Тема 7 Робота в відеорежимах Packed pixel graphics (PPG): вікна відеопам'яті, процес відображення відеопам'яті

Більшість функцій BIOS, у тому числі і функції VBE, незалежно від версії, розраховано на виконання в реальному (16-розрядному) режимі роботи мікропроцесора. Якщо задача виконується в захищеному (32-розрядном) режимі, то для звернення до функцій BIOS необхідний тимчасовий перехід в реальний режим роботи мікропроцесора. Це збільшує кількість допоміжних дій при виклику функції BIOS і уповільнює процес їх виконання. Розробники передбачили можливість безпосереднього виклику процедур, але розрахованих на виконання в захищеному режимі.

Переривання Int 10h, функції 4Fh, підфункції 0Ah: повертає адресу масиву.

Перед її викликом треба очистити регістр BL, а в AX занести наступні значення:

В es – сегмент масиву, розташованого в області BIOS у форматі для реального режиму (частіше всього код C000h);

в di – адреса (зсув) початку масиву в цьому сегменті;

в cx – розмір масиву в байтах;

Перші 3-і слова масиву es:[di+0], es:[di+2], es:[di+4] містять адреси (зсуви щодо початку масиву) точок входу в процедури, які дублюють функції 4F05h, 4F07h, 4F09h для захищеного режиму. Процедури повністю переміщувані, вони можуть виконуватися як безпосередньо в ROM BIOS так і в оперативній пам'яті, зрозуміло, після попереднього копіювання, для чого і потрібен розмір масиву, що повертається в регістрі cx.

Вказані процедури повинні викликатися як ближні, тобто без зміни сегментного регістра. Якщо задача використовує просту лінійну модель пам'яті, то доступ до області BIOS відбувається без зміни сегментного регістра і немає необхідності копіювати процедури в оперативну пам'ять. Якщо ж простір адресу сегментований, то процедури треба скопіювати в сегмент кодів. У такому разі їх виклик відбуватиметься без зміни сегментного регістра.

Фактично процедури не є повними аналогами функцій, які виконуються в реальному режимі. Є наступні відмінності:

- аналог 4F05h підтримує роботу тільки з одним вікном A.
- аналог 4F07h лише встановлює новий початок ділянки відеопам'яті, що відображається, причому замість номера рядка і стовпця при виклику указується повна (32-розрядна) адреса початку області, що відображається . Його старша частина поміщається в регістр dx, а молодша в cx.
- аналог 4F09h підтримує тільки основний набір регістрів DAC.

Окрім перерахованих функцій масив, який описується, може містити перелік номерів портів відео карти і адрес, які задача може використовувати для введення і виведення даних. Якщо такий список присутній, то його зсув щодо початку масиву вказаний в слові es:[di+6]. Якщо це слово очищено, то списку в масиві немає. (Програми в захищеному режимі можна почитати в статті Андріанова С.А в журналі "Мир ПК" [1,2], Internet - www.open systems.ru).

Додаткові функції VBE

Не дивлячись на невелике число функцій, їх склад виявився цілком достатнім. Автори версії VBE 3.0 (опубл. в вересні 1998г) не ввели жодної нової функції, а тільки розширили можливості існуючих з урахуванням новітніх досягнень розробників відео карт.

За допомогою функції 4F10h операційна система реалізує енергозберігаючі функції монітора згідно стандарту DPMS, також розробленому VESA (Управління станом монітора за цим стандартом, здійснюється шляхом відключення сигналів синхронізації, що і дозволяє зробити функція 4F10h).

За допомогою функції 4F13h можна управляти вбудованою акустичною системою мультимедійного монітора.

За допомогою функции4F15h реалізується підтримка моніторів Plug Play, що використовують для обміну даними спеціальний канал DDC.

Відеорежими packed pixel graphics (PPG).

При роботі в цих режимах код точки займає один байт і є номером рядка палітри, що містить опис кольору. В палітрі може бути описано тільки 256 кольорів. Робота з кольором відрізняється від побудови графічних об'єктів, тому її опис винесений в окремий розділ.

Робота з окремими точками

Комп'ютерна графіка, незалежно від її складності, зрештою зводиться до роботи з окремими точками зображення. В режимі PPG кожній точці екрану відповідає байт відеобуфера. При незмінній палітрі колір точки залежить від вмісту цього байта.

Команди для маніпуляції з точками

В графічних режимах VESA доступ до відеобуфера нічим не відрізняється від доступу до оперативної пам'яті. Тому для читання або зміни вмісту байтів використовують команди, які виконують пересилку, здвиг операндів, логічні, арифметичні та інші операції.

Починаючи з 386 мікропроцесора, вони можуть маніпулювати байтами, словами і подвійними словами. Відповідно, при роботі в режимах PPG однією командою можна обробити одну, дві або чотири підряд, розташованих на екрані точок.

Пересилка операндів є однією з найчастіших дій при роботі з відеопам'яттю. Її виконують команди mov та рядкові команди movs, stos, lods.

Їх основні властивості і відмінності наступні.

Команда **mov** двоадресна, копіює вміст джерела в приймач.

Команда допускає різні способи адресації операндів, проте обидва операнди не можуть одночасно знаходиться в пам'яті. Для копіювання вмісту одного байта, слова або подвійного слова в інший байт, слово або подвійного слова потрібні 2-і команди пересилки, що використовують як посередник один з регістрів загального призначення.

Таблиця 1 Пересилка із пам'яті в пам'ять

Пересилка байта	Пересилка слова	Пересилка подвійного сл.
Mov al, fs:[di] Mov gs:[si], al	Mov ax, fs:[di] Mov gs:[si], ax	Mov eax, fs:[di] Mov gs:[si], eax

При записі операндів команди mov можна використовувати імена всіх сегментних регістрів (додані fs і gs). Якщо сегментний регістр не вказаний явно, то мається на увазі, що це ds. В сегментному регістрі повино знаходитися конкретне значення сегменту звичайної, розширеної або відеопам'яті. Зсув байта, слова або подвійного слова, в цьому сегменті частіше за все указується в індексному регістрі, ім'я якого береться в квадратні дужки (це ознака адреси). В таб. 1 при читанні з пам'яті повна адреса операнда задається в регістрах fs:di, а при записі в пам'ять – в регістрах gs:si.

Рядкові конструкції **lods, movs, stos** відрізняються від команди пересилки наступними особливостям :

- ім'я інструкції може містити додаткову п'яту букву – b, w, d, яка вказує на кількість байтів, що пересилаються (1, 2, 4).
- якщо ім'я інструкції містить 5-ть букв, то операнди не указуються їх місцезнаходження визначено за умовчанням і залежить від інструкції.
- після виконання відповідної пересилки вміст індексних регістрів, що містять адреси операндів, збільшуються або зменшуються.
- збільшення або зменшення адрес операндів залежить від стану спеціальної ознаки напряму пересилки.
- тільки перед рядковою командою може бути вказана спеціальна команда *rep*, що викликає її багаторазове повторення. Призначення інструкцій пересилки, імена яких складаються з п'яти букв, показані в таб. 2 . Місцезнаходження операндів у них фіксовано, і змінити його не можна. Один з операндів може знаходитися в регістрі-акумуляторі, а інший або обидва – в оперативній пам'яті. Остання буква імені інструкції (b, w, d) указує, який з цих регістрів є акумулятором – al, ax, cx. Адреса операнда, що знаходиться в оперативній пам'яті, задається в одній з двох пар регістрів – ds:si або es:di. Вміст цієї пари повинен бути визначений в задачі до виконання інструкцій пересилки.

Таблиця 2 Призначення рядкових операцій

Розмір операнда в байтах	Читання пам'яті в акумулятор Accum = ds:[ci]	Запис аккумуля-тора в пам'ять Es:[di] = accum	Пересилка із пам'яті в пам'ять Es:[di] = ds:[ci]
1	Lodsb	Stosb	Movsb
2	Lodsw	Stosw	Movsw
3	Lodsd	Stosd	Movsd

Напрям пересилки

Після виконання рядкової операції адреса, що знаходяться в індексному регістрі (або в двох регістрах) збільшується на 1 або зменшується. В першому випадку прийнято говорити про пересилку в прямому напрямі, а в другому в зворотному. Перед корекцією адреси мікропроцесор перевіряє стан прапора напряму, який зберігається в 7-ом розряді регістра прапорів. Якщо цей розряд обчищений, то вміст індексних регістрів збільшується на розмір операнда, а якщо він встановлений, то зменшується. Стан 7-го розряду регістра прапорів змінюють 2-і спеціальні команди cld і std. Перша (cld) очищає розряд, дозволяючи тим самим пересилку в прямому напрямі. Друга (std) встановлює розряд, дозволяючи пересилку у зворотному напрямі. Обидві команди не мають операндів.

Програмні цикли.

Циклом прийнято називати багаторазове повернення на початок групи команд до тих пір, поки не буде виконана задана умова. Нас цікавлять тільки цикли по лічильнику. Для управління циклом по лічильнику призначена спеціальна команда loop. Вона має 2-а оператора, але явно в команді указуються тільки один з них - мітка, на яку передається управління. Неявно loop використовує лічильник повторів, яким є вміст регістра cx. При кожному виконанні команди вміст cx зменшується на 1, а якщо результат відмінний від нуля, то управління передається на вказану в команді мітку. В протилежному випадку виконуватиметься команда, розташована після.

Приклад 1 Очистити повний сегмент пам'яті за допомогою команди mov. Код сегменту, що очищається, повинен знаходиться в регістрі es. Для прискорення при кожному повторі очищається 4 байти.

```
xor eax, eax ; очищення регістра eax
```

```
        xor   di, di              ; di = 0, адреса початку сегменту
         mov   cx, 16384          ; cx = 16384, лічильник повторів
lps:  mov   es:[di], eax          ; запис 4-х байтів в сегмент
        add   di, 04              ; di = di+4, корекція адреси
        loop   lps                ; управління повторами циклу
```

Команда loop може передавати управління на мітку, яка відділяється від неї не більше ніж на 128 байтів. Інакше буде видано повідомлення про помилку.

Для багаторазового повторення призначена команда rep, яку називають префіксом повторення.

Приклад 2 Мікропрограмний цикл очищення сегменту пам'яті.

```
        xor   eax, eax            ; очищення регістра eax
        xor   di, di              ;  адреса початку сегменту
         mov   cx, 16384          ; cx = 16384, лічильник повторів
        rep   stosd               ; очищення сегменту, вказаного в es
```

Видно, що мікропрограмний цикл виконується значно швидше, ніж програмний.

Інструкції loop і rep визначаються способом роботи з лічильником. Rep спочатку перевіряє вміст регістра cx, і якщо він відмінний від 0, виконує рядкову операцію, а потім зменшує вміст cx на 1.

Loop спочатку зменшує вміст cx на 1 і залежно від результату повторює або припиняє виконання циклу. Ця відмінність виявляється, якщо при вході в цикл регістр cx очищений. У такому разі вказана після rep операція не виконується жодного разу, а команда loop повторюватиме виконання 65536 разів.

При обміні даними з відеопам'яттю можна використовувати як звичайні, так і рядкові операції пересилки. Коли можливий вибір, перевагу слід віддавати рядковим операціям.

Вікна видеопам'яті

При роботі в режимах SVGA зображення, що знаходиться на екрані монітора містить велику кількість точок. Воно залежить від дозволяючої здатності встановленого режиму і дорівнює кількості точок в рядку помноженої на кількість рядків. Наприклад, при дозволі (640*480) на екрані знаходиться 307200 точок, а при дозволі (1600*1200) – 1 920 000 точок. Об'єм відеопам'яті, необхідний для зберігання вмісту екрану також залежить від розміру коду точки. В режимах PPG код точки займає 1 байт, тому об'єм відеопам'яті, співпадає з кількістю точок, що знаходяться на екрані, в режимах direct color він в 2 або в 4 рази більше. Отже, для роботи з графічними об'єктами в режимах SVGA потрібен доступ до великого простору відеопам'яті.

Сегментація пам'яті.

Незалежно від конкретного призначення команди IBM PC завжди мають доступ до обмеженого простору адрес, граничний розмір якого залежить від декількох чинників, у тому числі і від режиму роботи мікропроцесора. Починаючи з 386 моделі, мікропроцесори можуть працювати в реальному, віртуальному і захищеному режимах. В реальному і віртуальних режимах весь простір пам'яті ділиться на сегменти, граничний розмір яких складає 65536 байтів. Вказані в командах адреси операндів завжди відносяться до конкретного сегменту, і ні за яких умов не можуть виходити за його межі, - це викликає аварійну ситуацію. Значення сегменту зберігається в одному з сегментних регістрів, який або явно указується в імені операнда, або використовується за замовчанням.

Доступ до сегментів.

При роботі на IBM PC в реальному режимі для доступу до простору адрес, розташованих за межами сегменту, використовуються два різні способи, вибір яких залежить від типу пам'яті.

Перший спосіб полягає в тому, що в сегментний регістр записується абсолютна адреса початку потрібного сегменту. Специфічною особливістю сімейства IBM ПК є те, що при роботі з сегментами загальний об'єм простору , що адресується, не може перевищувати один мегабайт.

Отже, можливо використовування тільки 16 сегментів граничного розміру. Тому пряма вказівка адреси початку сегменту застосовується тільки при роботі з молодшою частиною оперативної пам'яті (перші 640Кбайт).

Другий спосіб полягає в тому, що сегмент виконує роль вікна, через яке доступна та або інша частина реального простору адрес. Вміст сегментного регістра при цьому незмінний, а доступна частина адрес змінюється по запитах задачі. Такий спосіб має на увазі існування спеціального пристрою і програмного забезпечення, яке підтримує роботу з потрібним простором адрес. На IBM ПК він застосовується для доступу до відеопам'яті і до розширеного простору оперативної пам'яті ПК.

Доступ до відеопам'яті.

Розглянемо, як організовується робота із всім простором відеопам'яті. Для звернення до нього використовується спеціальний сегмент, який прийнято називати відео сегментом. При роботі в графічних режимах він має код A000h, але краще узяти його точне значення з масиву Info (в поперед. роздлі). Код відео сегменту є просто ознакою звернення до відеопам'яті, а не до якого-небудь пристрою і не є частиною адреси відеопам'яті.

Пізнавши звернення до себе, відеоконтроллер одержує від процесора 16-розрядну адресу і додає її до старшої частини, що зберігається в одному з його внутрішніх регістрів. В результаті виходить повна (абсолютна) адреса комірки відеопам'яті, до якої звертається команда. Коміркою, як завжди, може бути байт, слово або подвійне слово. У внутрішньому регістрі відеоконтроллера зберігається число, яке прийнято називати номером вікна (або банка) відеопам'яті. У сучасних відеокарт розмір вікна фіксований і складає 65536 байтів, тому, знаючи об'єм відеопам'яті, можна обчислити кількість існуючих вікон. Одному мегабайту відеопам'яті відповідає 16 вікон, двом – 32 і т.д. При кожній зміні відеорежиму регістр, який містить номер вікна, очищається, тобто встановлюється нульове вікно відеопам'яті. Надалі поточний номер вікна залежить тільки від дій, що виконуються в задачі, яка може встановлювати нею будь-яке допустиме значення. При перемиканні вікон треба змінити вміст внутрішнього регістра відеоконтроллера, тому для виконання таких дій передбачена спеціальна процедура BIOS. Як вже говорилося в попередньому розділі, її виклик через переривання INT 10h не істотно уповільнює перемикання вікон, і стандарт VBE рекомендує пряме звернення обминаючи переривання INT 10h.

Приклад 3 Три підпрограми для роботи з відеовікнами.

Всі вони працюють із змінною Cur_win, що має розмір слова і містить номер поточного вікна. Три підпрограми призначено для установки вікна шляхом виклику процедури VMC. Підпрограми оформлені як внутрішні, тому вони повинні бути розташовані в тому ж сегменті пам'яті, в якому знаходиться основна програма. Звернення до них проводиться за допомогою call, в якій вказано ім'я підпрограми. Вхідним параметром для всіх трьох підпрограм є змінна Cur_win, початкове значення якої змінюється при установці наступного або попереднього вікна.

Ядром прикладу є підпрограма Set_Win. Вона поміщає поточне вікно в регістр dx, очищає регістр bx і викликає процедуру VMS для установки вказаного вікна.

Переривання Int 10h не використовується. Підпрограма Nxt Win встановлює наступне вікно. При її виконанні поточне значення змінної Cur_Win збільшується на одиницю, а потім виконується Set_Win. Підпрограма PrevWin встановлює попереднє вікно.
При її виконанні поточне значення змінної Cur_win зменшується на одиницю, потім викликається Set_Win.

```
        ; Установка наступного вікна
NxtWin : push ax                ; зберігаємо вміст ax
         Mov ax, GrUnit         ; читаємо одиницю приросту вікна
         Add Cur_win, ax        ; збільшуємо номер вікна
         Pop ax                 ; відновлює ax

        ; Установка вікна, вказаного в Cur_Win
SetWin : push Reg <ax,bx,dx>    ; збереження вмісту регістрів
         Xor bx, bx             ;ознака установки вікна
         Mov dx, Cur_win        ; встановлюється номер вікна
         Call [vmc]             ;звернення до підпрограми BIOS
         Pop Reg <dx,bx,ax>     ;відновлення вмісту регістрів
         Ret                    ;повернення з підпрограми

        ; Установка поточного вікна
PrevWin : push ax               ; збереження ax
          Mov ax, GrUnit        ; читаємо одиницю приросту вікна
          Sub Cur_win, ax       ; зменшуємо номер вікна
          Pop ax                ; відновлюємо ax
          jmp short SetWin      ; перехід на установку вікна
```

Процедура VMC розташована в "далекому сегменті", тобто в просторі адрес, що не належить задачі. В таких випадках для звернення до підпрограми використовується команда call, у якої операнд є подвійним словом, що містить повну адресу, сегмент і зміщення. Взяття імені змінної, в квадратні дужки вказує на те, що адресою процедури є не VMC, а значення, що зберігається в ній, яке встановлено наступним чином:

```
mov eax, es: [dI+och]
mov vmc, eax
```

Вказівка типу short в команді jmp примусить макроасемблер сформувати коротку команду (для переходу не більше ніж на 128 байт) код якої займає 2 байти. Pushreg і Popreg це макроси. Перший еквівалентний трьом командам : push ax, push bx, push dx, а другий – pop dx, pop bx, pop ax. В реальній програмі треба або замінити їх вказаними командами або помістити в початок тексту програми відповідні їм визначення (макросам).

Збереження в стеку однієї величини виконує одна команда, це ж відноситься і до відновлення. Існує спеціальний засіб, що дозволяє скоротити запис однотипних дій, що повторюються, і зробити її більш наочною. Таким засобом є *макроси*. При цьому, скорочується тільки початковий текст, а не кількість команд в тілі задачі.

Приклад 4 Збереження в стеку вмісту регістрів і відновлення із стеку раніше збережених регістрів.

```
    ; збереження в стеку регістрів, перерахованих в списку reg.
      PushReg macro reg     ; заголовок макроозначення
      Irp r,<reg>           ; початок оператора повторення
      Push r                ; заготівка повторюваної команди
      endm                  ; кінець оператора повторення
       endm                 ; кінець макроозначення
    ; Відновлення із стека регістрів перерахованих в списку.
```

```
PopReg macro reg       ; заголовок макроозначення
Irp r,<reg>   ; оператор повторення
Pop r         ; заготівка повторюваної команди
endm          ; кінець оператора повторення
 endm          ; кінець макроозначення
```

При визначенні макросів використовують спеціальні директиви. Текст будь-якого макросу починається директивою macro, перед нею указується ім'я макросу, а після неї список аргументів. Директива end m указує на кінець макроозначення. При виклику макросу директива irp повторюється стільки раз, скільки імен містить список reg. Назва кожного імені вибирається із списку і підставляється в команду push і pop замість параметра r. В макросах різноманітність імен не обмежена явно, допустимо вказівка будь-яких величин, які можуть бути операндами команд Push і Pop. Наприклад :

```
Pushreg <fs, gs, Cur_win>
```

Якщо задача складена коректно, то при її виконанні значення Cur_win повинне співпадати з номером вікна, що зберігається у відеоконтролері.

Зафарбування робочої області екрану

Для ілюстрації роботи з вікнами розглянемо підпрограму, яка послідовно заповнює частину відеопам'яті заданим кольором, що відображається на екрані, внаслідок чого весь екран забарвлюється в заданий колір. Текст підпрограми приведено на прикладі 5. Перед зверненням до підпрограми в байтах регістра eax треба вказати код вибраного кольору. Припустимо, що синьому кольору відповідає код 01. У такому разі для виклику підпрограми використовується наступні дві команди:

```
mov eax, 01010101h      ; запис коду 01 в байти регістра eax
call fillscr            ; виконання підпрограми fill scr
```

; продовження тексту основної програми

В тексті основної програми повинні бути описані макроозначення PushReg і PopReg і підпрограми для роботи з вікнами. Крім того до звернення до fillscr треба встановити один з відеорежимів PPG і визначити значення змінних horsize, Versize і Vbuff. (Про все було сказано в попер. розд.)

Приклад 5 Заповнення всього екрану заданим кольором.

```
Fillscr : PushReg <es, di, dx, cx, Cur_win, eax> ; збереження в стеці
          Mov Cur_win, 0        ; очищення змінної Cur_Win
          Call SetWin           ; установка нульового вікна
          Mov es, vbuff         ; es <= значення відео сегменту
          Xor di, di            ; 0 – початкова адреса відеосегменту
          Mov ax, Versize       ; ax <= кількість рядків (Versize)
          Mul Horsize           ; dx:ax = Versize* Horsize
          Mov cx, dx            ; cx = кількість повних вікон
          Mov dx, ax            ; dx = розмір останнього вікна
          Pop eax               ; відновлення вмісту eax
```

; Цикл запису в повністю заповнені вікна.

```
fl_lp  push cx          ; збереження лічильника сегменту
       mov cx, 16384    ; кількість повторів для rep
       rep stosd        ; запис в повне вікно
       call NxtWin      ; установка наступного вікна
          pop cx        ; відновлення лічильника сегментів
       loop fl_lp       ; управління повторами циклу
```

; Запис в частково заповнене вікно, якщо воно існує.

```
        Mov  cx,  dx          ; cx <= число  байтів, що залишилося
       Rep  stosb             ; запис  в  неповне  вікно

 ; Дії, пов'язані  із  завершенням  роботи  підпрограми
fl_out   pop  Cur_Win                  ; відновлення  Cur_Win
         call  Setwin                  ; відновлення  початкового  вікна
         PopReg <cx,  dx,  di,  es>    ; відновлення  регістрів
          Ret                          ; повернення  з  підпрограми
```

Перші 5-ть команд даного прикладу виконують допоміжну дію, а саме, збереження в стеці тих величин, які будуть пошкоджені при виконанні підпрограми, установку нульового вікна відеопам'яті, запис в регістр es коду відеосегменту і очищення регістра di. В основній частині підпрограми повністю і частково заповнені вікна обробляються по-різному. Тому треба заздалегідь обчислити кількість повних вікон, що містяться в робочій області пам'яті, і кількість байтів в частково заповненому вікні, якщо воно є. В прикладі ці величини обчислюються шляхом множення значень змінних: Versize і Horsize.

Один із співмножників команди повинен знаходиться в регістрі ax, а другий вказуватися в самій команді. Старша частина результату множення знаходиться в регістрі dx, а молодша – в ax. Таким чином, добуток дорівнює 65536*[dx] + [ax], (квадратні дужки означають вміст вказаних в них регістрів). Іншими словами, після множення в регістрі dx знаходиться кількість повних вікон, а в регістрі ax – кількість байтів в частково заповненому вікні. Після множення вміст регістра dx копіюється в cx, вміст регістра ax – в dx і відновлюється із стека зіпсований при множенні вміст регістра eax (код кольору точки). Далі дії виконуються в наступному порядку: спочатку зафарбовуються точки повністю заповнених вікон, а потім точки частково заповненого вікна, якщо такі є.

Фарбування точок повністю заповнених вікон відбувається в циклі, що має мітку fl_lp. Він повторюється стільки разів, скільки повних вікон містить робоча область екрану. Заповнення вікна виконує мікропрограмний цикл rep stosd, для його прискорення рядкова операція (stosd) записує у відеопам'ять відразу 4 байти. Тому, заздалегідь в регістрі cx задається 16384 повтори цієї операції. Після заповнення всього сегменту відбувається звернення до підпрограми NxtWin для установки наступного вікна. Потім із стека відновлюється зміст регістра cx і команда loop управляє повторами циклу.

Після виходу з циклу fl_lp в регістр cx копіюється з dx кількість байтів, що залишилася, і виконується мікропрограмний цикл rep stosb.

Заключні дії виконує фрагмент, що має мітку fl_out. В ньому відновлюються збережені в стеку величини, початкове вікно відеопам'яті і відбувається повернення з підпрограми. В даному прикладі не вимагалося спеціальної перевірки досягнення межі сегменту і необхідності перемикання вікна.

Точки та їх адреси.

Для того, щоб записана у відеопам'ять точка з'явилася в потрібному місці екрану, треба зв'язати її розташування на екрані з адресою відеопам'яті, по якій проводиться запис. Розташування точки на екрані прийнято вказувати за допомогою координат, що задаються у вигляді номерів рядка і стовпця.

Процес відображення відеопам'яті.

При роботі в графічних режимах відеоконтроллер безперервно виводить на екран монітора вміст області відеопам'яті, що відображається . Зображення на екрані будується в напрямі зліва направо і зверху вниз, а коди точок вибираються з відеопам'яті в порядку збільшення їх адрес. Адреси пам'яті традиційно починаються з нуля, тому

якщо точки на екрані теж пронумерувати, починаючи з нуля, то порядковий номер точки співпадатиме з порядковим номером комірки відеопам'яті.

Розмір комірки відеопам'яті (кода точки) залежить від встановленого відеорежиму і може змінюватися від одного до чотирьох байтів. (це найбільш спрощена схема, т.я стандарт VESA дозволяє переміщати початок області, що відображається, і змінювати логічний розмір рядка). В режимах PPG код точки займає 1 байт, тому її адреса у відеопам'яті співпадає з порядковим номером.

Зв'язок адрес з координатами.

Порядковий номер точки (N) обчислюється по значеннях координат, що задаються у вигляді номерів рядка (row) і стовпця (column), на перетині яких вона розташована. Номер рядка і стовпця починається з 0. Тому, для обчислення порядкового номера точки потрібно номер рядка помножити на кількість точок в рядку (Horsize) і до добутку додати номер стовпця.

N = row* Horsize + column

При роботі у відеорежимах PPG отриманий результат є абсолютною адресою байта пам'яті**,** що містить код точки. З урахуванням сегментації відеопам'яті з нього треба виділити номер вікна і адресу (зміщення) в цьому вікні.

При множенні за допомогою команди mul результат автоматично ділиться на 2-і потрібні нам частини. Номер вікна знаходиться в регістрі dx, а адреса – в регістрі ax. Адреса не вимагає ніякої корекції, а номер вікна треба помножити на одиницю приросту значення вікна (GrUnit), яка залежить від особливостей відеоконтролера (спосіб її визначення описаний в 3 розділі). Якщо задача працює із сторінками відеопам'яті, то до результату множення номера вікна на GrUnit додається значення базового вікна (Base_win).

При роботі в графічних режимах VGA такі обчислення виконують 2-і спеціальні функції переривання INT 10h. Функція 0Ch записує, а 0Dh зчитує код точки по заданих номерах сторінки, рядка і стовпця. Проте для режимів SVGA ці функції не придатні оскільки реалізований в них алгоритм розрахований на розташування точок в межах одного сегменту відеопам'яті. Тому, треба скласти власну підпрограму для обчислення адрес точок відеопам'яті.

Підпрограма CallWin

В прикладі 6 приведений текст підпрограми, яка обчислює адресу точки наступним чином. Обчислена адреса вікна присвоюється змінній Cur_win, вікно встановлюється і вказується поточним. Зміщення (адреса) в цьому вікні розміщується в регістр di. Перед зверненням до підпрограми в регістрах cx і dx вказується, відповідно, номер стовпця і рядка.

Приклад 6 Обчислення та встановка вікна та адреси точки.

```
Callwin: push reg <dx, ax>     ; збереження регістрів dx, ax
         Mov ax, horsize       ; в ax – розмір рядка
         Mul dx                ; множемо на номер рядка
         Add ax, cx            ; додаємо номер стовпця
         Adc dx,00             ; враховуємо можливість переповнювання
         Mov di, ax            ; копіюємо адресу в регістр di
         Mov ax, GrUnit        ; одиниця приросту вікна
         Mul dl                ; множемо на номер вікна
         Add ax, Base_Win      ; ! тільки при роботі із сторінками
         Mov Cur_Win, ax       ; копіюємо вікно в Cur_Win
         Pop reg <ax, dx>      ; відновлюємо регістри
         Jmp SetWin            ; установка вікна на вихід
```

Декілька слів про особливість команди множення (mul). При її записі явно вказується тільки один операнд, другий вибирається з акумулятора (al, ax ,eax), куди його треба заздалегідь помістити. Команда може множити байти, слова або подвійні слова, розмір співмножників визначається за розміром (типом) вказаного в команді операнда. Залежно від розміру співмножників добуток може містити 16, 32 або 64 розряди і відповідно знаходитися в регістрах ax, в регістрах dx і ax, або в регістрах edx і eax. В даному прикладі перша команда (mul dx) перемножує слова, тому добуток розташований в регістрах dx і ax, а друга (mul dl) перемножує байти, тому результат займає тільки регістр ax. Для того, щоб з адресою точки можна було працювати, треба встановити обчислене вікно відеопам'яті. Тому тут номер вікна записується в Cur_win і відбувається перехід на процедуру SetWin, яка встановлює вікно. Зміщення у вікні поміщається в регістр di для того, щоб його можна було використовувати для запису кодів точки за допомогою рядкової операції stos. В даному прикладі немає команди повернення з підпрограми (ret). Вона не потрібна, тому що процедура установки вікна не викликається командою call SetWin, а відбувається безумовний перехід на її початок (jmp SetWin). Повернення на модуль, що викликається, виконає процедура SetWin. (Тут немає гострої необхідності даної заміни. Вона наведена як приклад того, як можна виключити непотрібні дії).

При заміні команди call командою jmp треба стежити за тим, щоб у верхівці стека знаходилася адреса повернення на той модуль, що викликається.

Вказівка номерів стовпця і рядка в регістрах cx і dx вибрана для сумісності з драйвером "миша". При кожному переміщенні миші доводиться обчислювати нову адресу початку малюнка курсора, це і пояснює вибір вказаних регістрів.

Використовування CallWin.

В прикладі 7 показано, як можна помістити точку білого кольору в центр екрану. Вважаємо, що змінні Horsize і Versize містять кількість точок на екрані по горизонталі і вертикалі, білий колір має код 0Fh, а регістр es містить код сегменту відеобуфера.

Приклад 7

Виведення білої точки в центрі екрану.

```
        Mov dx, versize      ; код-во крапок по вертикалі
        Shr dx, 01           ; зменшуємо в 2-а рази (здвиг на 1 розряд
                                  ; вправо)
        Mov cx, horsize      ; код-во крапок по горизонталі
        Shr cx, 01           ; зменшуємо в 2-а рази
        Call CallWin         ; встановлюємо вікно і адресу
        Mov ax, 0Fh          ; поміщаємо в al код білого кольору
        Mov es:[di], al      ; малюємо точку
    ; продовження програми
```

Адреси кожної точки обчислюється тим або іншим способом при будь-якій роботі з графічними об'єктами – від виведення на екран наперед підготовленого малюнка до побудови складних геометричних фігур. Тому, ефективність будь-якого алгоритму, призначеного для роботи з графікою, багато в чому залежить від того, як організована робота з адресами точок. Найбільший час займає обчислення адреси кожної точки по значеннях її координат. Тому, процедури типу CallWin використовуються тільки для знаходження адрес опорних точок, починаючи з якими проводиться побудова зображення. Наприклад, такою точкою може бути лівий верхній кут прямокутної області, в якій повинен розташовуватися малюнок. Адреси решти точок зображення обчислюються спрощеним способом, в основі яких лежать рекурентні співвідношення, які зв'язують значення координат або адрес поточної і наступної точок.

Зміщення точки та адреси.

Суміжні точки розташовані на екрані монітора поряд одна з одною. Якщо опорна точка не лежить на межі екрану, то її оточують вісім суміжних точок. Їх розташування показано в таб. 3 де опорною є точка з номером 0. В правій частині таблиці приведені прирости адрес відеопам'яті складних точок щодо адреси опорної точки. Для обчислення адреси складної точки до опорної адреси додається зсув, вказаний у відповідному елементі таблиці. Буква h позначає змінну Horsize.

Таблиця 3. Розташування адрес суміжних точок.

Розташування точок			Приріст адрес		
8	6	7	-1-h	-h	1-h
5	0	1	-1	0	1
4	2	3	h-1	h	h+1

Команди, які виконують рядкові операції обов'язково змінюють вміст індексного регістра, тому після їх виконання він містить адресу не поточної, а наступної точки. Аналогічна ситуація виникає і після виконання циклу побудови рядка з використанням звичайних команд. Тому при складанні програми треба розібратися, яка саме адреса буде корегуватися, можливо, вона вже збільшена на 1.

При застосуванні рекурентних формул обчислювальна адреса може виходити за межі поточного вікна.