Тема 10 Особливості роботи в режимах Direct Color Colori колір в коді точки, кодування кольору, малювання ліній в режимах Hi-Color та True Color

Маніпуляції з графічними об'єктами у багатьох випадках залежать не тільки від розміру коду точки, але і від того, як розташовані базові кольори в цьому коді. Розмір коду точки і розташування в ньому базових кольорів залежать від відеорежиму. Стандарт VESA передбачає можливість визначення вказаних величин при виконанні задачі. Раніше ми говорили, що характеристики встановленого відеорежиму знаходяться в масиві Info, а їх перелік в таблиці 2.2. В масиві Info кількість розрядів в коді точки зберігається в байті 19h, а розташування базових кольорів для режимів direc color – в байтах 1Fh-26h. Коректно складена задача повинна використовувати ці величини для настройки на конкретний відеорежим.

Середня кількість кольорів.

Режими середнього кольорового дозволу прийнято називати Hi-Color. При їх встановлені можливі 2 способи кодування кольору, що розрізняються розмірами коду точки. Наприклад, в режимі 110h код точки займає 15 розрядів, в яких можна вказати 32768 (або 32К) різних комбінацій (кольорів), а в режимі 111h код точки займає 16 розрядів, в яких можна вказати 65536 (64К) різних комбінацій. Розрішення в обох випадках однакове (640*480) точок, розрізняється тільки розташування базових кольорів.

Кодування кольору

В режимах Hi-Color код точки займає одне слово, розташування базових кольорів в його розрядах показано в таблиці 1.

Таблиця 1. Розміщення базових кольорів в слові.

	Режим 32К кольорів															
	Червоний колір					Зелений колір					Синій колір					
F	E	D	З	B	A	9	8	7	6	5	4	3	2	1	0	

Режим 64К кольорів															
Червоний колір					Зелений колір						Синій колір				
F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0

В режимах 32К коди базових кольорів займають по 5 розрядів, старший розряд слова не використовується. В режимах 64К код зеленого кольору займає 6 розрядів, тому використовуються всі розряди слова. Наприклад, коди базових кольорів максимальної інтенсивності мають наступні значення:

Режим 32К: червоний – 7C00h, зелений – 3E00h, синій - 1Fh.
Режим 64К: червоний - 0F800h, зелений - 7E0h, синій - 1Fh.

Для того, щоб задача могла підтримувати обидва різновиди режимів Hi-Color, при роботі з кодом кольору треба враховувати вміст байтів 19h (розмір коду точки) та 1F-26h (розташування базових кольорів) масиву Info.

Порівняння з режимом PPG.

У відеорежимах PPG системна палітра дозволяє використовувати одночасно тільки 256 різних кольорів, а в режимах Hi-Color її розмір збільшується в 128 або 256 разів. В режимах PPG код базового кольору займає 6 розрядів, тому точка може мати один з 256 різних відтінків, в режимах Hi-Color – в 4 або в 8 раз менше. Таким чином, в режимах Hi-

Color, в порівнянні з режимами PPG, зменшується колірний дозвіл, але збільшується різноманітність кольорів, які можна одночасно побачити на екрані. Остання обставина є вирішальним доказом на користь режимів Hi-Color, але говорити при цьому про поліпшення якості передачі ніяк не можна.

Максимальний колірний дозвіл

Максимальний кольоровий дозвіл забезпечують відеорежими VESA-True Color або 24-bit Color (ця назва вказує розмір коду кольору, а не коду точки). В цих режимах базові кольори мають 256 градацій, а загальна палітра містить 256*256*256=16177216 (або 16М) кольорів.

Кодування кольору.

Код точки, звичайно, займає у відеопам'яті 32 розряди (4 байти або подвійне слово), окрім базових кольорів до нього входить додатковий (резервний) порожній байт. Розташування базових кольорів та пустого байта в розрядах подвійного слова показано в табл.2.

Таблиця 2. Розташування базових кольорів в 32-розрядному слові.

Байт 3 Розряди 24-31	Байт 2 Розряди 16-23	Байт 1 Розряди 8-15	Байт 0 Розряди 0-7
1F 18	17 10	F 8	7 0
Резервний байт	Червоний колір	Зелений колір	Синій колір

Байти оперативної і відеопам'яті розташовуються в порядку збільшення їх номерів (адрес). Саме в такій послідовності (зліва направо) вони виводяться на екран при перегляду вмісту пам'яті за допомогою різних редакторів. Адреса подвійного слова співпадає з адресою його нульового байта, тому першим в пам'яті зберігається код синього кольору, другим- зеленого, третім- червоного, а четвертий байт – порожній (резервний).

Код в пам'яті і в регістрі.

Байти регістрів прийнято нумерувати в напрямі справа наліво. На екран же вони виводяться, починаючи із старшого (зліва направо), тобто так, як розташовані десяткові розряди в загальноприйнятій формі запису чисел.

При порівнянні вмісту пам'яті і регістрів може скластися враження, що базові кольори переставлені. Насправді це не так, просто одні і ті ж дані представлені двома різними способами. Все це продемонстровано в таб. 3.

Таблиця 3. Розташування кодів кольору в пам'яті і в регістрі.

Назва кольору	Байти пам'яті				Байти пам'яті			
	0	1	2	3	3	2	1	0
Чорний	00	00	00	00	00	00	00	00
Синій	FF	00	00	00	00	00	00	FF
Зелений	00	FF	00	00	00	00	FF	00
Червоний	00	00	FF	00	00	FF	00	00
Білий	FF	FF	FF	00	00	FF	FF	FF

Після установки відеорежиму його основні характеристики можна прочитати в масиві Info. В ньому, починаючи із зсувом 1F, розташовано чотири пари байтів, що містять розмір коду і адресу його молодшого байта для червоного, зеленого, синього кольорів і резервного простору. Звичайно, в них знаходяться наступні коди: 08, 10h, 08, 08, 08, 00, 08, 18h, що відповідає табл.2.

Після установки режимів TRUE Color задача обов'язково повинна перевірити байт масиву Info із зсувом 19h, який містить розмір точки в бітах. Якщо цей байт містить код 20h, то відеокарта підтримує 32-розрядний код точки. Але не всі відеокарти встановлюють код такого розміру, одне виключеня описане далі.

Можливість роботи в режимах True-Color залежить від об'єму пам'яті, розташованої на відеокарті. При об'ємі відеопам'яті 1 Мбайт пам'яті доступні відеорежими 112h і 115h, які мають дозвіл (640*480) і (800*600) точок. При об'ємі видеопям'яті 4 Мбайта до них додається режим 118h з дозволом (1024*768) точок.

Порівняння з режимом Hi-Color.

В режимах True Color код базового кольору збільшився на три розряди, відповідно кількість градацій базових кольорів збільшилася в 8 разів, а загальна кількість кольорових відтінків – в 256 разів. Тому передача кольору в режимах True Color істотно поліпшується в порівнянні з режимами Hi-Color. Проте останні в 2 рази скорочують необхідний об'єм видеопям'яті і в стільки ж разів або більше прискорюють маніпуляції з графічними об'єктами. Тому у багатьох випадках, наприклад при роботі графічних акселераторів, основним є режим Hi-Color.

24-розрядний код точки.

Стандартом VESA не обумовлена обов'язкова наявність резервного байта в коді точки. Тому відеокарти, у яких він відсутній, а код точки займає всього 24 розряди, формально відповідає вимогам стандарту VBE 1.2.

Розташування базових кольорів в коді точки і їх розміри відповідають табл. 2, за винятком відсутнього порожнього байта.

Недоліки трьохбайтового коду.

У розглянутих раніше відеорежимах розмір коду точки співпадав з однією з одиниць вимірювання пам'яті – байт, слово, подвійне слово. Трьохбайтовий код точки породжує дві основні проблеми:

1. У віх без виключення команд мікропроцесорів Intel розмір операнда кратний ступеню двійки, тому обробити три байти однією командою не можна. У такому разі при обміні даними з відеопам'яттю доводиться спочатку обробляти слово, а потім байт або навпаки, що уповільнює процес обміну.
2. Розмір сегменту оперативної або відеопам'яті так само кратний ступені двійки. Тому в ньому не поміщається ціла кількість трьохбайтових точок. У однієї з них (першої або останньої) в поточному сегменті опиниться тільки частина коду,що відповідає одному або двом базовим кольорам. Це справжній подарунок! Він вимушує переглянути логіку маніпуляцій з точками, яка використовувалася дотепер, і в деяких випадках використовувати спеціальні підпрограми для запису кодів точок у відеопам'ять і їх читання з неї.

Підпрограми для запису і читання трьохбайтового коду точки приведені нижче. При їх складанні враховано наступне:

– код точки знаходиться в трьох молодших байтах регістра eax, причому базові кольори розташовані так, як показано в табл.2, а старший резервний байт не використовується;
– адреса відеопам'яті знаходиться в регістрі di, а поточне вікно задає змінна Cur_Win, при виконанні підпрограм початкове значення адреси і вікна не змінюється;
– код відеосегменту знаходиться в регістрі es;

– доступ до відеопам'яті відбувається через два вікна, вікно А використовується при записі, а вікно В – при читанні;
– якщо код точки поміщається в поточному вікні, то підпрограми повинні виконувати мінімум допоміжних дій.

Підпрограма запису коду точки

Текст підпрограми, що виконує запис кода точки зі регістра eax у відеопам'ять приведений в прикладі 1.

Приклад. Підпрограма запису 24- розрядного коду точки.

```
wrtpnt: push eax                ; збереження вмісту eax
        mov es:[di],al          ; запис першого байта коду точки
        shr eax, 08             ; зсув вмісту eax
        cmp di, -2              ; скільки байтів до кінця вікна?
        jae wrtpn1              ; 1 або 2 байта
        mov es:[di+1],ax        ; запис залишку коду точки
        pop eax                 ; відновлення вмісту eax
        ret                     ; вихід з підпрограми
wrtpn1: push Cur_win            ; збереження значення Cur_Win
        je wrtpn2               ; > до кінця вікна 2 байти
        call NxtWinA            ; установка наступного вікна
        mov es:[di+1],ax        ; запис залишку коду точки
        jmp SHORT wrtpn3        ; перехід на мітку wrtpn3
wrtpn2: mov es:[di+1], al       ; запис другого байта коду точки
        call NxtWinA            ; установка наступного вікна
        mov es:[di+2], ah       ; запис третього байта коду точки
wrtpn3: pop Cur_win             ; відновлення значення Cur_Win
        call SetWinA            ; відновлення початкового вікна
        pop eax                 ; відновлення eax
        ret                     ; вихід з підпрограми
```

Виконання прикладу починається із збереження в стеку коду точки, запису у відеопам'ять його молодшого байта і зсуву вмісту регістра eax на 8 розрядів вліво. Після зсуву 2 байти коду точки, що залишилися, знаходяться в реєстрі ax. Тепер треба перевірити, скільки байтів залишилося до кінця вікна, і вибрати спосіб запису залишку коду точки. Якщо залишилося більше 2 байтів, то команда jae не виконує перехід на мітку wrtpn1. Останні 2 байти коду точки записуються у відеопам'ять., відновлюється початковий код в регістрі eax і відбувається повернення з підпрограми.

Якщо до кінця залишилося 1 або 2 байта, то команда jae виконує перехід на мітку wrtpr1. На цій вітці підпрограми спочатку в стеку зберігається значення змінної Cur_win і знов перевіряється кількість байтів, що залишилася (команда push не змінює стан прапорів).

Якщо до кінця вікна залишилося 1 байт, то команда je не проводить перехід на мітку wrtpr2 і виконується наступна команда, що встановлює нове вікно відеопам'яті. Потім в це вікно записуються 2 байти коду точки, що залишилися, і відбувається безумовний перехід на мітку wrtpn3 для завершальних дій.

Якщо до кінця вікна залишилося рівно 2 байти, то команда je виконає перехід на мітку wrtpr2. В цьому випадку у відеопам'ять треба записати ще один байт коду точки, що знаходиться в регістрі al, викликати підпрограму для установки наступного вікна і записати в це вікно старший байт коду точки з регістра ah.

Фрагмент підпрограми, що починається з мітки wrtpn3, є загальним для випадків, коли до кінця буфера залишилося 1 або 2 байта. В ньому відновлюється початкове значення

змінної Cur_win, початкове вікно відеопам'яті і вміст регістра eax. Після цих дій відбувається повернення на модуль, що викликається.

Підпрограма читання коду точки.

Текст підпрограми приведений в прикладі 2. При її використанні в розділі даних задачі треба зарезервувати один байт, привласнивши йому ім'я Dot Buff.

Приклад Підпрограма читання 24-розрядного коду точки.

```
rdpnt:    xor   eax, eax        ; очищення регістра eax
          mov  al, es: [di]     ; читання молодшого байта коду точки
          mov  DotBuff, al      ; і його збереження в DotBuff
          cmp  di,-2            ; скільки байт до кінця вікна?
          jae    rdpnt2                 ; 1 або 2 байта
          mov  ax, es: [di+1]           ; читання старших байтів коду
rdpnt1:  sh1    eax, 08                 ; зсув вмісту eax вліво
          mov  al, DotBuff  ; додавання коду молодшого байта
          ret                   ; вихід з підпрограми
rdpnt2:  push Cur_win                 ; збереження значення Cur_Win
          je     rdpnt3                 ; до кінця вікна 2 байти
          call   NxtWinB              ; установка наступного вікна
          mov   ax, es: [di+1]          ; читання старших байтів коду точки
          jmp    SHORT rdpnt4       ; перехід на мітку rdpnt4
rdpnt3:   mov   al, es: [di+1]   ; читання 2-го байта коду точки
          call    NxtWinB    ; установка наступного вікна
          mov  ah, es:[di+2]            ; читання старшого байта точки
rdpnt4:   pop Cur_Win                   ; відновлення значення Cur_Win
          call SetWin B                 ; відновлення початкового вікна
          jmp SHORT rdpnt1          ; перехід на мітку rdprt 1
```

Виконання прикладу починається з очищення регістра eax. Це потрібно для очищення старшого байта коду, який формується. Потім молодший байт коду точки прочитується в регістр al і поміщається в DotBuff. Тепер потрібно перевірити, скільки байтів залишилося до кінця вікна, і вибрати спосіб читання старших байтів коду точки.

Якщо до кінця вікна залишилося більше 2 байтів, то перехід на мітку rdprt2 не відбувається, і виконується команда, наступна за jae rdprt2. Вона поміщає в регістр ax два старші байти коду точки. Вміст регістра eax зсовується на 8 розрядів вліво, а в молодший байт, що звільнився, копіюється вміст DotBuff і відбувається повернення з підпрограми на модуль, що викликається.

Якщо до кінця вікна залишилося менше ніж 3 байти, то команда jae виконує перехід на мітку rdpnt2. при цьому в стеці зберігається значення змінної Cur_Win, і якщо в буфері залишився 1 байт, то команда je не виконує перехід на мітку rdpnt3. В цьому випадку встановлюється наступне вікно відеопам'яті, в регістр ax прочитуються 2 старші байти коду точки, і відбувається безумовний перехід на мітку rdpnt4 для завершальних дій.

Якщо до кінця вікна залишилося 2 байти, то команда ja rdprt3 виконує перехід на мітку rdpnt3. В цьому випадку в регістр al спочатку записується середній код точки, після цього встановлюється наступне вікно відеопам'яті і в регістр ah зчитується старший байт коду точки.

Далі виконується фрагмент підпрограми, що має мітку rdprt В ньому відновлюються значення змінної Cur_win і початкове вікно відеопам'яті, після чого відбувається безумовний перехід на мітку rdprt1 для останнього формування коду і виходу з підпрограми.

Робота з двома вікнами.

В приведених вище прикладах використані імена підпрограм NxtWin і SetWin, з доданими до них буквам A і B.

Способи роботи з двома вікнами відеопам'яті описані в розділі Там мовилося про два варіанти перемикання вікон, одне з яких доступно тільки для читання, а інше тільки для запису. Перший варіант заснований на одночасному перемиканні вікон. Текст відповідної підпрограми SetWin приведений в прикладі 3. Другий варіант заснований на незалежному перемиканні вікон для запису і читання. В розділі 3 ... описано, як скласти дві групи підпрограм для незалежної роботи з вікнами. При цьому рекомендувалося додати до основних імен підпрограм букви A і B. Вибір одного з цих варіантів залежить від конкретних особливостей алгоритму перетворення графічного об'єкту.

Якщо при використанні підпрограм прикладів 1 та 2 необхідно працювати з двома різними вікнами, то треба використовувати два комплекти підпрограм з іменами, вказаними в прикладах 1 і 2. Якщо ж допустимо одночасне перемикання вікон, то в іменах підпрограм треба просто прибрати букви A і B.

Коли використовуються підпрограми.

Розташування коду точки в двох суміжних вікнах подія достатньо рідкісна. Якщо вести звіт від початку відеопам'яті, то в трьох підряд розташованих вікнах воно відбувається двічі. Наприклад, при установці режиму 112h на екрані поміщається 307200 точок, їх коди займають 15 неповних вікон відеопам'яті. З них тільки коди 10 точок розташовані в двох суміжних вікнах (10 випадків з 307200).

Якщо код точки розташований в одному вікні, то в описаних підпрограмах виконується 9 команд при записі і 10 при читанні (з урахуванням команди виклику підпрограми). Одним з можливих компромісів між розміром задачі і часом її виконання є звернення до підпрограм тільки в тих випадках, коли код точки знаходиться в двох суміжних вікнах.

Координати і адреси точок.

Для виведення точки заданого кольору в потрібне місце екрану треба зв'язати координати цього місця з адресою відеопам'яті, по якій повинен бути записаний код точки. Ми повертаємося до питання точки і їх адреси, але вже з урахуванням особливостей режимів direct color.

Нові змінні.

Після читання масиву Info задачі доступні дві величини, які мають відношення до розрішаючої здатності режиму.

1. Одна з них розташована в слові із зсувом 12h, вона вказує ширину екрану, виражену в точках. В главі 3 пропонувалося берегти копію цього слова в змінній Horsize, яка потім неодноразово використовувалася в прикладах.
2. Інша величина розташована в слові із зміщенням 10h, вона вказує, скільки байтів відеопам'яті відображається при виведенні рядка на екран. Інакше кажучи, це ширина рядка, помножена на розмір коду точки, виражений в байтах.

При роботі в режимах PPG обидві величини співпадають, тому ми використовували тільки першу з них. Тепер нам може знадобиться і друга величина, тому після читання масиву Info її значення треба привласнити тимчасовій bperline. В деяких випадках нам буде потрібен розмір коду точки в байтах. Такої величини в масивах Info немає, але байт із зсувом 19h містить кількість розрядів в коді точки. Якщо його зсунути на три розряди управо і результат перетворити в слово, то вийде потрібна нам змінна.

Якщо копію цієї змінної змістити ще на один розряд вправо, то вийде ще одна змінна, що містить кількість слів в коді точки. Імена і описи нових змінних наступні:

```
bpenline  dw    2560   ; розмір рядка який відображається на екрані в байтах
bytppnt   dw    0004   ; розмір коду точки, виражений в байтах
wrdppnt   dw    0002   ; розмір коду точки, виражений в словах
```

В цьому описі змінних вказані значення, які отримують при установці режиму 112h – True Color, 640*480 точок.

Перетворення координат в адресу.

Виконується перед початком роботи з більшістю графічних об'єктів. Формулу для обчислення адреси по заданих значеннях координат x і y можна записати в наступному вигляді:

$$Adress = (y * horsize + x) * bytppnt$$

При заміні множення на bytppnt зсувами дії виконувалися в тій послідовності, в якій вони вказані у формулі – спочатку множення, потім складання і, нарешті, зсув. Якщо ж зсув замінити множенням, то послідовність дій доведеться змінити. Результат дій взятий у дужках розташований в двох регістрах, dx містить його старшу частину, а ax – молодшу. Для множення подвійного слова (або вмісту двох регістрів) на значення змінної bytppnt, буде потрібно багато допоміжних дій. Щоб спростити обчислення в приведеній вище формулі, треба розкрити дужки і врахувати, що bperline = horsize * bytppnt, в результаті вийде наступний вираз:

$$Adress = y * bperline + x * bytppnt$$

Підпрограма Caladdr.

В прикладі 3. приведений текст підпрограми, що виконує обчислення по цій формулі. Перед її викликом в регістрі cx вказується номер стовпця (координата x), в регістрі dx – номер рядка (координата y). Обчислена адреса поміщається в регістри dx:ax, тобто в ax знаходиться значення вікна, а в dx – адреса (зсув) в цьому вікні.

Приклад 3. Універсальна підпрограма обчислення відеоадреси:

```
Caladdr: mov ax, bperline      ; ax   = розмір рядка в байтах;
         mul dx                ; dx:ax = У+bperline;
         push dx               ; зберігаємо старшу частину результату;
         xchg ax, cx           ; обмін вмісту регістрів;
         mul bytppnt           ; ax = X*bytppnt, dx=0;
         add ax, cx            ; обчислюємо молодшу частину адреси;
         mov dx, ax            ; зберігаємо її в регістрі dx;
         pop ax                ; ax =старша частина У*bperline;
         adc ax, 00            ; враховуємо можливість перенесення;
         mul byte ptr GrUnit ; ax = al*GrUnit;
         add ax, Base_win      ; якщо використовується базове вікно;
         ret                   ; повернення;
```

Вміст регістра dx (старшу частину добутку y*bperline) треба зберегти в стеці тому що воно буде зіпсовано при повторному множенні. Після другого множення і обчислення молодшої частини адреси, старша частина виштовхується з стеку в регістр ax. До неї додається одиниця перенесення, яка могла виникнути, якщо при виконанні команди add ax, cx сталося переповнення і був встановлений C- розряд регістра прапорів (ознака Carry). Команди пересилки і виштовхування із стеку не змінюють стан C- розряду. Тому, якщо він був встановлений, то adc ax,00 додасть одиницю до вмісту регістра ax.

Можна змінити текст програми 3 так, щоб обчислена адреса поверталася в регістрі di, значення вікна привласнювалося змінній Cur_Win і виконувалася установка вікна (Call SetWin). В результаті вийде варіант підпрограми CallWin, описаний в прикладі 6, вживаний в будь-яких відеорежимах VESA.

Інший варіант Caladdr.

В прикладі 4 показаний варіант підпрограми Caladdr, в якому замість множення x*bytppnt вміст регістра cx зсувається на к розрядів вліво. Залежно від відеорежиму в команді зсуву літеру к потрібно замінити цифрами 1 або 2, тобто це підпрограма не універсальна.

Приклад. Перерахунок координат в адресу з використанням зсуву

```
Caladdr:  mov ax, bperline     ; ax = розмір рядка в байтах
          mul dx               ; dx:ax = Y*bperline
          shle cx, k           ; к=1 для Hi-Color; к=2 для True Color
          add ax, cx           ; обчислюємо молодшу частину адреси
          adc dx, 00           ; враховуємо можливість перенесення
          xchg ax, dx          ; обмін вмісту регістрів
          mul byte ptr GrUint  ; ax=al*GrUnit
          add ax, Base_Win ; якщо використовується базове вікно
          ret                  ; повернення
```

В даному прикладі зміщується не результат множення, а тільки значення координати x (вміст регістра cx). Це можливо тому, що значення координати У (вміст dx) множеться не на Horsize, а на

bperline = horsize * bytppnt

Особливість операцій зсувів.

Дана особливість полягає в тому, що величина зсуву може або знаходиться в регістрі cl, або вказується безпосередньо в команді. Тому для автоматичного вибору величини зсуву в прикладі 4 замість двох підряд розташованих команд:

```
shl cx, k
add ax, cx
```

треба записати наступні:

```
mov bx, wrdppnt    ; bx= величина зсуву;
xchg bx, cx        ; обмін вмісту регістрів;
shl bx, cl         ; зсув значення координати x;
add ax, bx         ; обчислюємо молодшу частину адреси;
```

Якщо відеокарта в режимі True Color підтримує трьохбайтовий код точки, то замінювати множення зсувами не доцільно.

Підсумок: Перша з двох підпрограм універсальна, а друга спеціалізована. Першу краще складати при розробці бібліотечних модулів, особливо для мов високого рівня.

Координати і адреси суміжних точок.

Значення координат потрібні для обчислення базової адреси відеопам'яті, що відповідає якійсь опорній точці, як правило лівому верхньому куту, графічного об'єкту. В процесі роботи з об'єктом адреси решти точок обчислюються спрощеним способом, виходячи з поточного значення адреси, тобто враховується залежність приросту адрес від взаємного розташування точок на екрані.

Якщо базова точка не лежить на одній з чотирьох границь екрану, то її оточує вісім суміжних точок. В таблиці 4 показані прирости значень координат і адрес суміжних точок.

Базова точка має координати x і y а приріст її адрес дорівнює 0. В правій частині таблиці літера к відповідає змінній bytppnt, а літера W- змінній bperline.

Таблиця Приріст координат і адрес суміжних точок.

Приріст координат суміжних точок			Приріст їх адрес		
X-1, У-1	X, У-1	X+1, У-1	-K-W	-W	K-W
X-1, У	X, У	X+1, У	-K	0	К
X-1, У+1	X, У+1	X+1, У+1	W-K	W	W+K

З таблиці 4, зокрема витікає, що при переміщенні по горизонталі адреси точок зменшуються або збільшуються на значення змінної bytppnt. Якщо для роботи з кодами точок використовуються звичайні операції, то після їх виконання поточну адресу треба змінити на bytppnt. Якщо ж застосовуються рядкові операції, то вони автоматично змінюють поточну адресу на розмір операнда. При обробці точок в природному порядку, тобто у бік збільшення значень їх координат, рядкові операції збільшують адресу, а при обробці точок в зворотному порядку зменшують її. Таким чином, при послідовній обробці точок рядка для отримання адреси чергової точки достатньо простої переадресації операндів.

Адреса наступного рядка.

Якщо рядки графічного об'єкту обробляються послідовно один за одним, то після побудови поточного рядка, треба визначити адресу початку наступного рядка. В цьому випадку простої переадресації операндів недостатньо.

Перші точки рядків прямокутної області розташовані на екрані в одному стовпці. Значення координати x у них співпадають, а координати У розрізняються на величину, кратну значенню змінної bperline. Це треба враховувати при обчисленнях.

Адресу початку поточного рядка можна зберегти в спеціально виділеному місці і для переходу до наступного рядка збільшити або зменшити його значення на bperline. Проте невідомо, якому вікну належить збережена адреса, оскільки при обробці рядка могла відбутися зміна вікна. Тому, при такому способі обчислень буде потрібна спеціальна ознака перемикання вікна і аналіз стану.

Для спрощення обчислень треба використовувати адресу, отриману в кінці обробки поточного рядка. Вона явно належить встановленому вікну, а відрізняється від адреси початку рядка на ширину прямокутної області, виражену в байтах. Тому, якщо до нього додати значення змінної bperline, зменшене на ширину прямокутної області, то вийде адреса початку наступного рядка. Ширина прямокутної області задається у вигляді кількості точок, яку треба помножити на розмір коду точки. Приведені міркування можна записати у вигляді наступної формули обчислення константи корекції адреси (offsline позначає зсув рядка)

Offsline = bperline – widthrect * bytppnt = (horsize – widthrect) * bytppnt.

В цих формулах widthrect позначає ширину прямокутної області, виражену в точках. Обидва варіанти формули рівноцінні по кількості виконуваних дій. При роботі в режимах PPG bytppnt = 1 формула перетворюється на різницю (horsize-widthrect), яка обчислювалась в приведених раніше прикладах.

В загальному випадку при малюванні ліній і геометричних фігур приріст адрес суміжних точок не рівні одиниці, вони обчислюються по спеціальних алгоритмах.

Лінії, рядки і прямокутні області

Для запису кодів декількох точок використовуються спеціальні підпрограми, які малюють лінії геометричних фігур або виконують побудову рядків малюнків.

В режимах Direct color спосіб пересилки залежить від розміру коду точки і не залежить від розташування в ньому базових кольорів. Код точки може займати 2, 3, або 4 байти, а команди пересилки і рядкові операції працюють тільки із словами (2 байти) або з подвійними словами (4 байти).

Підпрограми для малювання ліній.

При оформленні робочої області екрану часто використовуються одноколірні горизонтальні і вертикальні лінії. Спосіб їх малювання залежить від кута нахилу та напряму лінії. Зображення на екрані дискретно по своїй природі, тому рівними є тільки лінії, нахилені під кутом кратним 45^o, при їх малюванні значення одній або обох координат змінюється від точки до точки на 1. В решті випадків приріст координат обчислюється по спеціальних алгоритмах. Від напряму лінії залежать способи переадресації операндів і зміни вікна відеопам'яті в тих випадках, коли значення адрес виходять за межі сегменту.

Перераховані особливості малювання ліній детально обговорювалося в розділі В даному розділі описано малювання одноколірних горизонтальних ліній в прямому напрямі (зліва направо). При цьому, основна увага приділяється впливу розміру коду точки на виконувані дії.

Базові варіанти підпрограм.

В Прикладі 5 наведено два варіанти підпрограм малювання лінії, що розрізняються способом пересилки. Перед їх викликом код кольору точки поміщається в регістр ax або eax, а розмір лінії (кількість точок) в cx. Початкова адреса відеопам'яті вказується в регістрі di, і встановлюється вікно відеопам'яті, якому належить ця адреса. Регістр es повинен містити код відеосегменту (значення змінної Vbuff).

Приклад Цикл малювання горизонтальної лінії в режимах Hi-Color

```
horline: mov es:[di], ax          ; для True Color – mov es[di],eax
         add di, bytppnt          ; переадресація операнда
         jne @F                   ; якщо не нуль, перехід
         call NxtWin  ; установка наступного вікна
@@:      loop horline ; управління повторами циклу
         ret                      ; повернення
```

Варіант 2; використовується рядкова операція

```
horline: stosw                    ; для True Color – stosd
         or di, di                ; початок нового сегмента
         jne @F                   ; ні
         call NxtWin              ; установка наступного вікна
@@:    loop horline          ; управління повторним циклом
         ret                      ; повернення
```

Текст прикладу 5 відрізняється від тексту прикладу 8 незначними змінами. В першому варіанті при переадресації використовується не 1, а значення змінної bytppnt, яке рівно 2 або

Для використовування підпрограм прикладу 5 в режимах True Color треба перші команди в обох варіантах замінити командами, вказаними в коментарях. В цих режимах перед викликом підпрограм код кольору точок поміщається в регістр eax, оскільки він займає 32 розряди.

Тут приведено два варіанти циклів. Команда пересилки зручна в тих випадках, коли переадресація не може виконуватися відразу після запису або читання коду точки або коли крок переадресації не співпадає з розміром коду точки, наприклад при малюванні вертикальних ліній. Якщо вказані умови не суттєві, то другий варіант циклу 5 більш підходе. Після компіляції він виявиться коротше першого на 3 байти, і виконуватиметься дещо швидше.

Прискорення циклу малювання.

В прикладі 5 після кожної переадресації виконується перевірка належності нового значення адреси поточному сегменту і, у разі потреби, встановлюється наступне вікно відеопам'яті. Імовірність того, що нове значення адреси вийде за межу сегменту, достатньо мала. Для скорочення кількості додаткових дій треба змінити спосіб перевірки.

Один з варіантів скорочення даремних дій полягає в тому, щоб перед початком малювання лінії перевіряти, поміщається вона в поточному сегменті чи виходить за його межі. Це виключає необхідність контролю адрес в циклі малювання. Якщо лінія цілком розташована в одному сегменті, то цикл малювання виконується один раз. В іншому випадку після малювання частини лінії, розташованої в поточному вікні, встановлюється наступне вікно і повторюється цикл малювання залишку лінії.

Підпрограма Twopart.

Текст прикладу 6 є своєрідним управляючим алгоритмом, який для виконання конкретних дій викликає допоміжну підпрограму baselp. В даному прикладі вона малює пряму лінію. Вхідні параметри підпрограми Twopart повністю відповідають параметрам підпрограми horline (приклад 5) і розташовані в тих же регістрах.

Приклад 6 Малювання ліні по частинах в режимах Hi-Color

```
Twopart: push dx            ; зберігаємо вміст регістра dx
         mov dx, di         ; копіюємо адресу в регістр dx
         shl cx, 01 ; для True Color – shr cx, 02
         add dx, cx ; сума початкової адреси і розміру лінії
         jc @F              ; пряма розташована в 2-х вікнах
         xor dx, dx ; очищення регістра dx
@@:      sub cx, dx  ; кількість точок в поточному вікні
         shr cx, 01 ; для True Color shr cx, 02
          call baselp       ; малюємо всю лінію або її першу частину
           or di, di  ; адреса в межах поточного вікна?
          jne hrl_exit       ; так, лінія намальована повністю
           call NxtWin       ; установка наступного вікна
           add cx, dx        ; кількість не намальованих точок
          je hrl_exit        ; лінія намальована повністю
           shr cx, 01        ; для True Color shr cx, 02
           call baselp       ; малюємо залишок лінії
hrl_exit:  pop dx            ; відновлення вмісту dx
           ret               ; повернення

; Підпрограма, що виконує основні дії
 baselp: mov es:[di], ax     ; для True Color mov es:[di], eax;
         add di, bytppnt ; переадресація адресантів;
```

```
        loop baselp         ; управління повтором циклу;
        ret                 ; повернення;
```

Виконання прикладу 6 починається із збереження в стеку вмісту регістра dx, оскільки він змінюється в підпрограмі. При вході регістр cx містить кількість мальованих точок, його треба перетворювати в кількість байтів за допомогою зсуву і скласти з початковою адресою відеопам'яті (в регістрі dx). Якщо при цьому відбувається переповнювання, то лінія не поміщається в поточному вікні і її треба малювати по частинах. В протилежному випадку регістр dx очищається. Після цього обчислюється кількість точок в першій частині і викликається підпрограма baselp, що малює початок лінії.

При першому виклику baselp може бути намальована вся лінія або тільки її перша частина. Це важливо знати для виконання подальших дій. Вони починаються з перевірки значення адреси, що знаходиться в регістрі di. Якщо при першому малюванні досягнута межа вікна, то в регістрі di знаходиться 0, але нульова адреса належить не поточному, а наступному вікну відеопам'яті. Тому, якщо регістр di очищений, то обов'язково треба змінити вікно відеопам'яті. Якщо ж вміст di відрізняється від нуля, то намальована вся лінія.

Після установки вікна сумується вміст регістрів cx і dx. Якщо сума дорівнює нулю, то лінія намальована цілком, а код її останньої точки був записаний в останнє слово сегменту. В протилежному випадку кількість байтів, отримана в регістрі cx, перетвориться в кількість точок і повторно викликається підпрограма baselp. Перед поверненням на модуль, що викликається, відновлюється початковий вміст регістра dx, відповідна команда має мітку hrl_exit.

Цінність даного прикладу полягає в тому, що він ілюструє спосіб обробки рядка графічного об'єкту по частинах. Основні дії локалізовані в підпрограмі baselp. Її можна змінити так, щоб замість запису у відеопам'ять вмісту регістра ax виконувалися інші дії, наприклад, інверсія кодів точок, пересилка кодів з відеопам'яті в оперативну і навпаки.

Прискорене малювання лінії

В підпрограмі baselp роботу з відеопам'яттю виконує одна команда, тому відразу після неї можлива переадресація операнда. Якщо при цьому крок адресації співпадає з розміром операнда, то замість команди пересилки можна використовувати рядкову операцію. В цьому випадку тіло циклу пересилки скорочується до однієї команди rep stosw, яку треба вставити замість call baselp. Це і показано в прикладі 7.

Приклад. Прискорене малювання лінії в режимах Hi-Color.

```
horline: push dx          ; зберігаємо вміст dx
         mov dx, di       ; копіюємо адресу в dx
         shl cx, 01       ; для True Color shrl cx, 02
         add dx, cx       ; пряма розташована в двох вікнах
         xor dx, dx       ; очищення регістра dx
hrl_1:   sub cx, dx       ; кількість точок в поточному вікні
         shr cx, 01       ; для True Color shl cx, 02
         rep stosw        ; для True Color stosd
         or di, di        ; адреса в межах поточного вікна?
        jne hrl_exit      ; так, лінія намальована повністю
        call NxtWin       ;установка наступного вікна
         mov cx, dx       ;  кількість не намальованих точок
         shr cx, 01       ; для True Color – shr cx, 02
         rep stosw        ; для True Color – stosd
hrl_exit: pop dx          ; відновлення вмісту dx
          ret             ; повернення
```

В даному прикладі в другій частині перевіряється тільки вміст регістра di і не перевіряється кількість точок, що залишилася. Це допустимо тому що якщо регістр cx очищений, то цикл rep stosw не виконуватиметься і попередня перевірка вмісту, cx не обов'язкова. І ще – код кольору лінії поміщається в регістр eax для режиму True Color.

Трьохбайтовий код точки.

Такий код не укладається в загальну схему по двом причинам. По–перше, розмір операндів команд не може бути рівний трьом байтам. По-друге, існують особливі точки, код яких розташований в двох суміжних сегментах. Тому потрібні спеціальні підпрограми, при складанні яких враховуються особливості трьохбайтових кодів. Дві такі підпрограми, виконуючі запис і читання коду точки, приведено в прикладах 1 і 2.

Складемо спеціалізовану підпрограму, яка самостійно обробляє більшість точок і викликає підпрограму wrtpnt тільки для запису кодів послідовних точок відеосегментів. Текст такої підпрограми приведений в прикладі 8, перед викликом адреса першої точки поміщається в регістри es:di, а код кольору точки в регістр eax. Це зроблено для сумісності з чотирьохбайтовими режимами.

Приклад. Малювання лінії, режим True Color, трьохбайтовий код

```
horline: push bx              ; зберігаємо вміст bx
         mov ebx, eax         ; копіюємо eax в ebx
         shr ebx, 16 ; bx= старші розряди коду точки
hrln_1:  cmp di, -3   ; це остання точка у вікні?
         jae hrln_3   ; так, особливий випадок
         stosw                ; запис двох молодших байтів коду
         mov es:[di], bl      ; запис старшого байта коду
hrln_2:  inc di               ; переадресація операнда
         loop hrln_1 ; управління повторами циклу
         pop bx               ; відновлюємо вміст bx
         ret                  ; повернення
hrln_3: call wrtpnt    ; запис коду особливої точки
        call NxtWinA         ; установка наступного вікна
       jmp short hrln_2 ; короткий перехід на hrln_2
```

Для спрощення дій, які виконуються в основному циклі прикладу 8, код точки треба розташувати в регістрах ax і bx. Початковий вміст регістра bx зберігається в стеку, а на його місце поміщається старша половина регістра eax, тому код червоного кольору опиняється в регістрі bl.

Основний цикл має мітку hrln_1 і починається з перевірки адреси. Якщо виявиться, що до кінця сегменту залишилося більш ніж 3 байти, то продовжується виконання основної частини циклу. Спочатку у відеопам'ять записуються коди двох молодших байтів точки, а потім код старшого байта. Далі проводиться переадресація операнда, і команда loop управляє повторами циклу. Після закінчення циклу відновлюється початковий вміст регістра bx і відбувається повернення на модуль, що викликається. Особливі точки обробляє фрагмент підпрограми, що має мітку hrln_3. Перехід на неї відбувається в тих випадках, коли до кінця відеосегменту залишається менше чотирьох байтів. При цьому викликається підпрограма wrtpnt для запису коду точки, встановлюється наступне вікно відеопам'яті і відбувається перехід на мітку hrln_2 для продовження циклу.

Підпрограми для побудови рядків.

Рядки відрізняються від ліній тим, що в оперативній пам'яті зберігається їх точковий образ. Він може бути отриманий в результаті читання файлу, якій містить малюнок, збереження зображення, що знаходилося на екрані або будь-яким іншим способом.

Якщо не вимагається додаткових перетворень кодів точок, то побудова рядка малюнка зводиться до пересилки заданої кількості байтів з оперативної у відеопам'ять або у зворотньому напрямі (для збереження вмісту відеопам'яті).

Для всіх варіантів підпрограми побудови рядка збережемо те розташування вхідних параметрів в регістрах, яке було прийняте в розділі 3.1. Адреса оперативної пам'яті (джерела) вказується в парі регістрів fs:di, а адреса відеопам'яті (приймача) – в регістрі di. Заздалегідь встановлюється вікно відеопам'яті, якому належить адреса першої точки (вказана в di). Регістр es повинен містити код сегменту відеобуфера, що зберігається в змінній Vbuff.

Початковий варіант підпрограми.

В прикладі 9 приведений варіант підпрограми, що ілюструє послідовність дій при побудові рядка.

Приклад 9 Цикл побудови рядка в режимі Hi-Color.

```
drawline:  mov ax, fs:[si]   ; для True Color mov eax, fs:[si]
           mov es:[di],ax    ; для True Color mov es:[di], eax
           add si, bytppnt   ; переадресація операнда джерела
           add di, bytppnt   ; переадресація операнда приймача
           jne @F            ; перехід, якщо не нуль
           call NxtWin   ; установка наступного вікна
@@:        loop drawline ; управління повторами циклу
           ret               ; повернення
```

Варіант з рядковою операцією.

Команди пересилки доцільно використовувати тільки в тих випадках, коли переадресацію операнда треба відкласти на деякий час або коли приріст адреси не співпадає з розмірами операнда.

При простому копіюванні рядків розмір операнда співпадає з кодом точки (якщо він не трьохбайтовий) і можлива корекція адреси відразу після запису точки. Тому в прикладі 9 команди пересилки сенс замінити рядковими операціями. Одна рядкова операція замінює чотири перші команди – дві пересилки і дві переадресації операндів. Змінений цикл побудови рядка приведений в прикладі 10. Його можна використовувати в тих випадках, коли код точки займає 2 або 4 байти.

Приклад. Поліпшений цикл побудови рядка в режимі Hi-Color

```
drawline: movs word ptr [di],fs:[si]    ; (dword для True Color)
          or di, di                     ; початок нового сегменту?
          jne @F                        ; ні
          call NxtWin                   ; установка наступного вікна
@@:       loop drawline                 ; управління повтором циклу
          ret                           ; повернення
```

Перша команда прикладу 10 змінна, спосіб її запису для пересилки 32- розрядних кодів (True Color) показаний в першому рядку

```
movs dword ptr [di], fs:[si]
```

Для використовування всіх переваг рядкової операції з циклу треба виключити перевірку значень адрес, тобто пересилати рядок по частині так, як це робилося в прикладі 7. Залежно від відеорежиму основні дії в ньому виконували команди rep stosw (Hi-Color) або rep stosd (True Color). При підстановці в текст прикладу 10 їх треба змінити так, як показано нижче rep stosw замінити rerp movs word ptr [di], fs:[si], rep stosd замінити rep movs dword ptr [di], fs:[si].

Універсальна підпрограма пересилки.

Заданий рядок, що містить N точок, код кожної з них займає M байтів. Цей рядок треба скопіювати з одного місця пам'яті в інше. Спочатку потрібно обчислити кількість байтів в рядку (L=N*M) і переслати L байтів з одного місця пам'яті в інше. В результаті множення ми позбулися розміру коду точки і при складанні підпрограми враховуєм тільки кількість байтів, що пересилаються.

Приклад 11 Універсальний (цикл пересилки) спосіб побудови рядків

```
drawline: push dx                ; збереження вмісту dx
          xchg ax, cx            ; обмін вмісту регістрів (ax = N)
          mul bytppnt            ; dx:ax = L = N*M
          xchg cx, ax            ; обмін вмісту регістрів (cx = L)
          pop dx                 ; відновлення вмісту dx
drawalt:  push dx                ; збереження вмісту dx
          mov dx, di             ; копіювання адреси в регістр dx
          add dx, cx       ; dx = початкова адреса +L
          jc @F                  ; пряма розташована в двох вікнах
          xor dx, dx       ; залишок в dx рівний нулю
@@:       sub sx, dx       ; кількість байтів в поточному вікні
          call moveto            ; будуємо перову частину рядка
          mov cx, dx             ; cx= залишок кількості байтів
          pop dx                 ; відновлення вмісту регістра dx
          or di, di              ; адреса в межах поточного вікна?
          jne d_exit       ; так, рядок побудований повністю
          call NxtWin            ; установка наступного вікна
moveto:   shr cx, 01       ; перетворимо байти в слова
          jnc@F                  ; парне число байтів
          movs byte ptr[di],fs:[si]; пересилка одного байта
@@:       shr cx, 01       ; перетворимо слова в подвійні слова
          jnc @F                 ; парне число слів
          movs word ptr[di],fs:[si];пересилка одного слова
@@:       je d_exit        ; пересилати більше чогось
          rep movs dword ptr[di],fs:[si]; основний цикл пересилки
        d_exit: ret        ; повернення з підпрограми
```

Приклад. Виконання починається з обчислення кількості байтів в рядку. При множенні використовуються регістри dx і ax, тому вміст dx зберігається в стеку, а вміст ax – за рахунок двократного використовування команди xchg. Добуток знаходиться в регістрах dx:ax. Вважатимемо, що воно менше 65536, тобто dx містить 0. Команда обміну xchg поміщає результат в cx, одночасно відновлюючи початковий стан ax, а із стека виштовхується початковий вміст регістра dx.

Якщо кількість байтів в рядку відома наперед, то наново обчислювати її не має сенсу. Вона указується в регістрі cx, а для виклику підпрограми використовується друга точка входу, що має ім'я drawalt.

Команда з міткою drawalt зберігає в стеку вміст регістра dx, потім в нього копіюється адреса відеопам'яті, яка збільшується на розмір рядка в байтах. Якщо при складанні відбудеться переповнювання (установка C-розряду), то команда jc @F виключає очищення dx. В іншому випадку рядок поміщається в поточному вікні і регістр dx очищається. Потім обчислюється кількість точок, що виводяться, і підпрограма moveto будує першу частину рядка.

Після побудови першої частини рядка в регістр cx копіюється вміст dx (залишок рядка). Регістр dx вивільнився і треба відновити його початковий стан. Для вибору

подальших дій перевіряється поточна адреса в регістрі di, якщо він відрізняється від нуля, то побудова рядка завершена і виконується команда ret. В протилежному випадку встановлюється наступне вікно відеопам'яті.

Программа moveto будує залишок лінії. Після її виконання завершиться робота основної підпрограми, оскільки у верхівці стека знаходиться адреса повернення на модуль, що викликається.

В підпрограмі moveto команда rep movs dword ptr[di],fs:[si] є головною, вона пересилає групу 32- розрядних слів. Проте кількість байтів в рядку не обов'язково кратно чотирьом. Тому потрібна попередня перевірка і пересилка від одного до трьох "зайвих байтів", так щоб залишок був кратний чотирьом.

Команда, що має мітку moveto, зсовує вміст регістра cx на розряд управо. Якщо він був непарним, то пересилається перший байт рядка, в протилежному випадку jnc OF виключає цю пересилку.

Потім вміст cx іще раз зміщується на розряд вправо. Якщо він був непарним, то пересилається слово (два байта рядка) в протилежному випадку jnc OF виключає цю пересилку.

В результаті виконання цих двох зміщень та пересилки "зайвих байтів" в рядку залишеться ціле число 32 розрядних слів, кількість яких знаходиться в регістрі cx. Якщо воно рівне 0, то здійсниться перехід на команду ret, в протилежному випадку виконується мікропрограмний цикл копіювання 32-розрядних слів. Після цього виконується команда ret.