Тема 9 Побудова малюнків: варіанти побудови рядків, відтворення нестиглих малюнків

На відміну від геометричних фігур малюнки не створюються в процесі виконання задачі, а готуються наперед і зберігаються у файлах, на зовнішніх носіях. На жаль (або на щастя) не існує єдиного стандарту структури таких файлів, але існують спеціальні програми для їх перетворення з одного стандарту (формату) в інший. Крім того, таке перетворення виконують всі поширені графічні редактори.

В структурі файлу, що містить точковий малюнок, можна виділити три основні частини : заголовок, палітру і образ малюнка.

Заголовок розташовується на початку файлу і містить вичерпну інформацію, необхідну для виведення малюнку на екран або на друк.

Палітра знаходиться після заголовка або після образу малюнку. Вона містить коди що використовуються в малюнку кольорів. Опису призначення палітри і способів роботи з нею присв'ячений наступний розділ.

Образ малюнка містить коди точок, які утворюють малюнок. Адреса його початку (зміщення) у файлі, звичайно, вказується в заголовку. В деяких випадках перед побудовою або в процесі побудови малюнка можливо буде необхідно перетворення його образу.

Перед тим, як виводити малюнок на екран, весь файл або його частину треба зчитати в оперативну пам'ять, виділити із заголовка всі необхідні величини і встановити палітру кольорів, що використовуються. Вважатимемо, що всі підготовчі дії виконані, і образ малюнка знаходиться в оперативній пам'яті. Таке припущення дозволить розглянути графічні аспекти побудови малюнків в чистому вигляді.

Варіанти побудови рядків.

Точкові малюнки, незалежно від їх конкретного змісту, завжди займають на екрані прямокутну область, а їх образи зберігаються у файлах у вигляді послідовності рядків однакового розміру.

Тому побудова малюнка зводиться до послідовної побудови його рядків на екрані монітора.

Рядки малюнків відрізняються від ліній геометричних фігур тим, що їх образи існують в оперативній або у відеопам'яті. Тому в простих випадках треба просто перемістити коди точок із одного місця пам'яті в інше. В даному розділі ми працюватимемо з рядками, у яких код точок займає 1 байт.

Побудова рядка зліва направо.

В прикладі 17 наведений текст підпрограми, що виконує копіювання образу рядка з оперативної пам'яті у відеопам'ять. В результаті на екрані з'явиться зображення рядка. Копіювання проводиться в прямому напрямі, тобто у бік збільшення адрес.

Перед зверненням до підпрограми встановлюються вікно відеопам'яті, в якому повинні розташовуватися точки рядка, а в регістрі di вказується адреса першої (лівої) точки. Крім того, пара регістрів fs:si повинна містити адресу початку рядка в оперативній пам'яті, fs – сегмент, а si – зміщення (відносна адреса) в цьому сегменті.

Розмір рядка (кількість точок) поміщається в регістр cx. Нагадаємо, що es повинен містити код відеосегменту (значення змінної vbuff).

Приклад 17 Побудова рядка 256 кольорового малюнка.

```
Drawline : mov  byte ptr [di],  fs: [si]  ; запис коду  точки  у  відеобуфер
           Or  di,  di                    ; початок  нового  сегменту?
           Jne  @F                        ; ні, обхід  команди  call  NxtWin
           Call  NxtWin                   ; установка  наступного  вікна
     @@ :  loop  drawline                 ; управління  повторами  циклу
```

```
        ret
```

Якщо порівняти другий варіант прикладу 8 і приклад 17, то побачимо, що вони відрізняються тільки однією командою. Замість stosb, яка записує у відеопам'ять вміст регістра al, в даному випадку використовується рядкова операція movs, що копіює у відеопам'ять байти оперативної пам'яті. В записі рядкової операції сегментний регістр приймача (es) вказувати не обов'язково, а сегментний регістр джерела (fs) можна змінити на свій розсуд.

Підпрограма прикладу 17 виводить точки в порядку збільшення їх адрес. Якщо виникне необхідність виводити точки в зворотньому порядку (справа наліво) у бік зменшення адрес, то можна використовувати приклад 9, замінявши в ньому команду lodsb на команду movs byte ptr [di], fs : [si].

Прискорення циклу побудови.

Для прискорення побудови рядка використовується мікропрограмний цикл пересилки, аналогічний описаному в прикладі 10. Вихідні параметри вказуються так само, як для підпрограми прикладу 17.

Приклад Прискорена побудова рядка малюнка.

```
Drawline : push dx                          ; збереження вмісту регістра dx
           Mov dx, di                       ; копіювання адреси в регістр dx
           Add dx, cx                       ; сума поточної адреси і кількості
                                            ; точок
           Jc @F                            ; пряма розташована в двох вікнах
           Xor dx, dx                       ; очищення регістра dx
    @@ : sub cx, dx                         ; обчислюємо кількість точок в
                                            ; поточному вікні
           rep movs byte ptr [di], fs:[si]  ; будуємо рядок або його частину
           or di, di                        ; адреса в межах поточного вікна?
           jne dhl_out                      ;так, рядок побудований повністю
           call NxtWin                      ; установка наступного вікна
           mov cx, dx                       ; кількість не побудованих точок
           rep movs byte ptr [di], fs:[si]  ; будуємо залишок рядка
dhl_out : pop dx                            ; відновимо вміст dx
           ret
```

Для подальшого прискорення виконання побудови рядка потрібно використовувати пересилку одночасно двох або чотирьох байтів. На прикладах 11 і 12 показані зміни, які дозволять це зробити при малюванні лінії. Аналогічні зміни треба внести і в приклад 18, не забувши замінити рядкові операції наступними:

Stosb на movs byte ptr [di], fs:[si]
Stosw на movs word ptr [di], fs:[si]
Stosd на movs dword ptr [di], fs:[si]

Слід зазначити, що якщо образ малюнка знаходиться у файлі, то час, затрачений на читання і попередні дії, значно більший чистого часу, який витратили для побудови всіх його рядків. У такому разі доцільність прискорення побудови рядків вельми проблемна.

Крім 256 – кольорових малюнків досить часто використовують 16-и і 2-кольорові. Вони використовуються, наприклад, Windows із її додатками для оформлення робочої області екрану та іншої мети. Такі малюнки зберігаються в упакованому вигляді і перед записом у відеопам'ять повинні бути розпаковані. Спосіб упаковки є загальноприйнятим і не залежить від конкретного стандарту, в якому підготовлений файл. При розпаковуванні треба врахувати, що залежно від кількості точок в початковому рядку останній байт упакованого рядка може бути заповнений частково.

Розпаковування 16-кольорових рядків.

Код 16-кольорового малюнка займає 4 розряди, і для скорочення розміру файлу в одному байті розташовуються коди двох точок. Перед записом у відеопам'ять коди точок, що знаходилися в одному байті треба розташувати в молодших розрядах двох різних байтів. Ця операція достатньо проста, і її доцільно виконати в процесі побудови рядка. Тим паче, що при попередньому розпаковуванні малюнка його розмір збільшується в два рази. Підпрограма для розпаковування рядка в процесі побудови 16-кольорового малюнка приведена в прикладі 19. Перед зверненням до неї треба встановити вікно відеопам'яті, в якому повинні розташовуватися точки рядка, а адресу першої точки вказати в регістрі di. Пара регістрів fs:si повинна містити адресу оперативної пам'яті, починаючи з якої зберігається упакований рядок. В регістрі cx вказується кількість точок в рядку.

Приклад Підпрограма побудови рядка 16-кольорового малюнка.

```
Drwlin 4 : mov al, fs:[si]        ; читаємо в al код пари точок
           Shr al, 04             ; виділяємо код старшої точки
           Stosb                  ; записуємо його у відеопам'ять
           Or di, di              ; початок нового сегменту?
           Jne @F                 ; ні, обхід команди NxtWin
           Call NxtWin            ; установка наступного вікна
    @@ : lods byte ptr fs:[si]    ; повторне читання коду пари точок
           dec cx                 ; cx = cx - 1
           je dr4ret              ; якщо cx = 0, то кінець рядка
           and dl,0Fh             ; виділяємо код молодшої точки
           stosb                  ; записуємо його у відеопам'ять
           or di, di              ; початок нового сегменту?
           jne @F                 ; ні, обхід команди call NxtWin
           call NxtWin            ; установка наступного вікна
    @@ : loop drwlin 4            ; управління повторами циклу
dr4ret :  ret                     ; вихід
```

В прикладі 19 спочатку виділяється і записується у відеопам'ять код старшої тетради чергового байта упакованого рядка, а потім код його молодшої тетради. Після запису коду старшої тетради вміст cx зменшується на 1 і перевіряється значення результату. Якщо воно рівне нулю, то побудова рядка завершена. Вказані дії потрібні, тому що в останньому байті упакованого рядка при непарній кількості точок в рядку буде заповнена тільки одна старша тетрада.

Після кожного запису у відеопам'ять перевіряється значення адреси, що зберігається в регістрі di. Якщо вона дорівнює нулю, то відбувся вихід за межі сегменту і треба змінити поточне вікно відеопам'яті. Якщо адреса не рівна нулю, то виконання команди call NxtWin виключається. Остання команда loop управляє повторами циклу, якщо рядок містить парну кількість точок, то саме вона завершить виконання підпрограми. При непарній кількості крапок в рядку виконання підпрограму завершить команда je dr4ret.

Розпаковування 2-кольорових рядків.

Якщо при побудові малюнка використано тільки два кольори, наприклад чорний і білий, то код точки поміщається в одному розряді і може приймати тільки два значення 0 і 1. В таких випадках для скорочення розмірів файлу в одному байті розташовується коди восьми точок. В старшому розряді байта знаходиться код першої точки, а в молодшому - останній, тому виділяти коди точок, треба починаючи із старшого розряду. В залежності від кількості точок в рядку останній байт може бути заповнений частково. Не слід вважати, що двокольорові малюнки обов'язково чорно-білі – їх кольори залежать від кодів, що знаходяться в палітрі, яка додається до файлу.

Підпрограма для розпаковування рядка в процесі побудови 2-кольорового малюнка наведена в прикладі 20. Перед її викликом встановлюється вікно відеопам'яті, в якому повинен розташовуватися рядок, який будується, а адреса першої точки поміщається в регістрі di.Пара регістрів fs:si повинна містити адресу оперативної пам'яті, починаючи з якої зберігається упакований рядок. В регістрі cx вказується кількість точок в рядку.

Приклад Підпрограма побудови рядка 2-кольорового малюнку.

```
Drwlin1 : push bx                  ; зберігаємо вміст вх
          Mov bx, cx               ; вх = сх (к-ть точок в рядку)
Lpdrwll : lods byte ptr fs :[si]   ; читаємо в al код чергового байта
          Mov ah, al               ; копіюємо коди з al в ah
          Mov cx, 08               ; кількість повторів циклу розпаковки
Out8pnt : xor al, al               ; очищаємо регістр al
          Shl ah, 01               ; зсовуємо ah на розряд вліво
          Adc al, 00               ; додаємо переповнення до al
          Stosb                    ; записуємо код чергової точки
          Or di, di                ; початок нового сегменту?
          Jne @F                   ; ні, обходимо слід. команди
          Call NxtWin              ; установка наступного вікна
@@ : dec bx                        ; вх = вх - 1
          je drlret                ; якщо вх = 0, то рядок побудований
          loop out8pnt             ; управління внутрішнім циклом
          jmp short      ; на обробку наступного байта
drtret : pop bx                    ; відновлення вмісту вх
          ret                      ; вихід
```

Підпрограма прикладу 20 представлена двома вкладеними циклами. Ім'я зовнішнього циклу lpdrwll, а внутрішнього – out8pnt. Зовнішній цикл прочитує черговий байт образу рядка, копіює його в регістр ah і задає кількість повторів внутрішнього циклу. У внутрішньому циклі проводиться розпаковування чергової групи точок і запис їх кодів у відеопам'яті. Розпаковування виконують три перші команди внутрішнього циклу. Перша з них очищає регістр al, друга виконує зсув вмісту регістра ah на розряд вліво. При зсуві старший розряд регістра ah переноситься в С – розряд регістра прапорів, тому якщо він містить одиницю, то виробляється ознака переповнювання. Третя команда

```
Adc al, 00
```

Додає вміст С – розряду до регістра al. В результаті, в залежності від коду чергової точки, в регістрі al опиниться 0 або 1. Отриманий код точки команда stosb записує у відеопам'ять. Потім перевіряється поточна адреса відеопам'яті і при необхідності встановлюється наступне вікно відеопам'яті.

Після запису кожної точки вміст регістра вх зменшується на 1 і якщо він виявиться рівним нулю, то відбувається перехід на мітку drlret для завершення підпрограми. В іншому випадку команда loop керує виведенням восьми точок. Після цього відбувається короткий безумовний перехід на початок зовнішнього циклу для обробки наступного байта.

Читання рядка з відеопам'яті.

У всіх описаних вище підпрограмах проводилася копіювання вмісту оперативної пам'яті у відеопам'яті. На практиці порівняно часто доводиться вирішувати зворотну задачу, тобто копіювати вміст відеопам'яті в оперативну пам'ять. Наприклад, це може знадобитися для збереження початкового фону перед побудовою малюнка. В прикладі 21 приведена підпрограма, що виконує копіювання рядка з відеопам'яті в оперативну пам'ять. При її виклику адреси задаються так: пара регістрів es:di містить адресу

відеопам'ять, а пара fs:si – адресу оперативної пам'яті. Заздалегідь встановлюється вікно, в якому розташований початок який рядка копіюється, а її розмір вказується в регістрі cx.

Приклад Копіювання рядка з відеопам'яті в ОЗУ.

```
Readlin : mov al, es : [di] ; читання чергового байта відеопам'яті
          Mov fs : [si],al  ; запис коду точки в ОЗУ
            Inc si          ; збільшення адреси ОЗУ
            Inc di          ; збільшення адреси відеопам'яті
            Jne @F          ; адреса в межах вікна
            Call NxtWin     ;перехід до наступного вікна
@@ :        loop readlin    ; управління внутрішнім циклом
            ret             ; повернення
```

В прикладі 4 .21 використані звичайні команди пересилки, тому черговий байт спочатку прочитується з відеопам'яті в регістр al, а потім вміст al копіюється в оперативну пам'ять. Після цього вміст регістрів si і di збільшується на 1 і перевіряється значення нової адреси відеопам'яті. Якщо вона виявиться рівною нулю, то виконується команда call NxtWin, внаслідок чого встановлюється наступне вікно відеопам'яті. Команда loop readlin повторює виконання циклу до тих пір, поки не будуть скопійовані всі байти рядка. В розглянутому варіанті підпрограми, якщо не відбувається зміна вікна, то при пересилці одного байта виконується 6 команд. Така кількість допоміжних дій істотно уповільнює пересилку, що буде особливо відчутно при збереженні великих об'ємів відеопам'яті. Для скорочення допоміжних дій пересилку потрібно виконувати за допомогою рядкової операції movs.

Поліпшення циклу копіювання.

У операції movs функціонально призначення індексних регістрів di, si та сегментного регістра es. Тому для застосування рядкової операції треба змінити розташування адреси джерела і приймача. Пара регістрів fs:si повинна містити адресу відеопам'яті, а пара es:di - адресу оперативної пам'яті, але для зручності краще зберегти одноманітний спосіб розташування адрес і тимчасово змінювати його в самій підпрограмі пересилки. В прикладі 22 показано, як переставляти адреси джерела і приймача в тілі підпрограми на час виконання циклу пересилки. При зверненні до підпрограми цього прикладу регістри es : di, як завжди, повинні містити адресу відеопам'яті, а регістри fs:si – адресу оперативної пам'яті.

Приклад Копіювання рядка з відеопам'яті в оперативну пам'ять.

```
; пересилка вхідних параметрів
 readlinl : push fs                ; збережемо вміст fs
            pop es                 ; виштовхуємо його в es
            mov fs, vbuff          ; fs = vbuff (код відеосегменту)
            xchg di, si            ; перестановка вмісту di і si

; Цикл копіювання рядка з відеопам'яті в оперативну пам'ять
  readlp : mov byte ptr [di], fs : [si]  ; копіювання чергового байта
           or si, si                     ; адреса в межах сегменту?
           jne @F                        ; так, обхід наступній команді
           call NxtWin                   ; установка наступного вікна
           loop readlp                   ; управління повторами циклу

; Відновлення початкового розташування вхідних параметрів
           push es                       ; зберігаємо вміст es
           pop fs                        ; зберігаємо вміст fs
           mov es, vbuff                 ; виштовхуємо вміст fs в es
           xchg di, si                   ; пересилка вмісту di та si
```

```
ret                              ; повернення
```

В прикладі 22 перед виконанням циклу пересилки вміст регістрів fs копіюється в регістри es через стек, в fs розміщається код відеосегменту (вміст змінної vbuff) і переставляється вміст індексних регістрів di та si. Так отримують потрібні адреси джерела і приймача. Цикл пересилки має мітку readlp, він відрізняється від аналогічного циклу прикладу 17 тільки тим, що замість команди

```
Or di, di
```

використовується команда

```
or si, si
```

тому ми не повторюватимемо його опис. Після пересилки відновлюється початкове розташування вхідних параметрів в сегментних і індексних регістрах і відбувається вихід з підпрограми.

В прикладі 22 цикл readlp містить на дві команди менше ніж цикл підпрограми прикладу 21, топто кількість допоміжних команд скоротилася на третю частину. Це небагато, але з'явилася можливість подальшого прискорення процесу копіювання за рахунок використання мікропрограмного циклу пересилки.

Вище підкреслювалося, що якщо малюнок відтворюється з файлу, то проблема прискорення запису у відеопам'ять не така актуальна. Якщо ж малюнок зберігається в оперативній пам'яті або відновлюється з образу, збереженого в пам'яті, то від часу, який витрачається на копіювання з одного виду пам'яті в іншу залежить швидкодія задачі. У такому разі має сенс збільшувати розмір підпрограми для прискорення її роботи з відеопам'яттю.

Тепер перейдемо до розгляду способів побудову завершених малюнків.

Відтворення не стислих малюнків.

Рядки образу малюнка можуть зберігається у файлі в прямому або зворотньому порядку. В першому випадку вони розташовані за збільшенням номерів, тобто спочатку у файлі записані точки першого рядка, потім другого і так аж до останнього. В другому випадку вони розташовані по зменьшенню номерів, тобто спочатку у файлі записані точки останнього рядка, потім передостаннього і так до першого. Перший спосіб зберігання образу малюнка застосовується, наприклад, у файлах відповідних стандарту PCX, а другий – у файлах, відповідних стандарту BMP. Розпізнати належність файлу до цих стандартів можна по їх типу(розширенню), який співпадає з назвою стандарту. Послідовність розташування рядків у файлі визначає логіку роботи з їх адресами при відтворенні образу малюнка на екрані. В даному розділі розглянемо побудову малюнків, у яких рядки розташовані у файлі або в оперативній пам'яті, в природному порядку.

Побудова малюнка невеликого розміру.

Важливою характеристикою, впливаючою на вибір варіанту побудови малюнка, є його розмір. Ми розглянемо малюнки, образ яких поміщається в одному сегменті оперативної пам'яті, тобто їх розмір не перевищує 65536 байт.

Текст підпрограми, що виконує побудову малюнка, образ якої цілком поміщається в одному сегменті оперативної пам'яті, а рядки розташовані в природному порядку, приведені в прикладі 23. Перед зверненням до підпрограми повинне бути встановлено вікно відеопам'яті, що містить лівий верхній кут малюнка, а адреса цього кута вказана в регістрах es:di. Адресу початку образу малюнка в оперативній пам'яті задає пара fs:si. В регістрах dx і cx вказується ширина і висота малюнка.

Приклад Робота з прямокутною областю невеликого розміру.

```
Draw :  Push Reg <di, si, cx, bx, Cur_win> ; збереження початкових величин
        Mov bx, horsize                    ; копіюємо в bx значення horsize
```

```
        Sub  bx,  dx                          ; віднімаємо з нього ширину
                                              ; малюнка
Drwount : push  cx                            ; зберігаємо лічильник повторів
          Mov  cx,  dx                        ; задаємо розмір рядка  малюнка
          Call  drawline                      ; !! або call bp, пояснення в стеці
          Pop  cx                             ; відновлюємо лічильник повторів
          Add  di,  bx                        ; адреса початку наступного рядка
          Jnc  @F                             ; адреса в межах сегменту?
          Call  NxtWin                        ; установка наступного вікна
  @@ :  loop  drwount                         ; управління повторами циклу
          Pop Reg <Cur_win,  bx,  cx,  si,  di>  ; відновлюємо початкові
                                              ; величини
          call  SetWin                        ; відновлення початкового вікна
          ret                                 ; повернення
```

Побудова малюнка відрізняється від заповнення прямокутної області тим, що код кожної точки, що виводиться, вибирається з оперативної пам'яті, а не із регістра – акумулятора. Тому тексти прикладів 15 і 23 розрізняються тільки ім'ям підпрограми, яка викликається в циклі побудови: horline в 15 і drawline в 23.

Виконання прикладу 23 починається із збереження в стеку тих величин, які можуть змінитися в процесі побудови. Потім дві команди формують константу для переадресації рядків.

Цикл побудови малюнка має ім'я drwount. Він починається із збереження в стеку значення лічильника повторів (регістра cx) і запису в нього розміру рядка. Потім відбувається виклик підпрограми drawline для виведення на екран чергового рядка малюнка. Після повернення з підпрограми відновлюється збережене в стеку значення лічильника повторів і обчислюється адреса початку наступного рядка. Якщо при додаванні буде отримана ознака переповнювання, то відбудеться звернення до підпрограми NxtWin для установки наступного вікна відеопам'яті. Остання команда циклу loop повторює його виконання до тих пір, поки не будуть побудовані всі рядки малюнка. Після виходу з циклу відновлюються збережені в стеку величини, початкове вікно відеопам'яті і відбувається повернення на модуль, що викликається.

Вибір допоміжної підпрограми.

В розділі 3.1. було описано декілька варіантів підпрограм побудови рядка, кожний з яких застосовується в конкретних випадках. Всі вони сумісні по розташуванню вхідних параметрів в регістрах. Тому в прикладі 23 можна підставити ім'я будь-якої з них, але змінювати кожного разу текст або використовувати декілька варіантів прикладу 23 що відрізняються ім'ям допоміжної підпрограми, не доцільно. Простіше ввести додатковий параметр, який є адресою підпрограми, що викликається . Краще всього його задавати в регістрі bp, у вигляді адреси потрібної підпрограми, а в тексті прикладу 23 замість команди call drawline записувати call bp, це вказано в коментарі. Якщо допоміжні підпрограми включені в текст задачі, то для формування адреси використовується команда lea, наприклад:

```
Lea  bp,  drawline  ; для  малювання  256-х кольорових малюнків
Lea  bp,  drwlin4   ; для  малювання  16-х кольорових малюнків
Lea  bp,  drwlin1   ; для  малювання  2-х кольорових малюнків
Lea  bp,  horline   ; для заповнення  прямокутної області
```

Таким чином, ми отримаємо універсальну процедуру, що дозволяє :

- заповнювати прямокутні області довільного розміру;
- малювати малюнки невеликого розміру;
- зберігати в оперативній пам'яті вміст прямокутної області;
- відновлювати збережений раніше вміст прямокутної області

При малюванні, збереженні або відновленні вмісту відеопам'яті прямокутна область може містити не більше ніж 65535 точок. Це єдине обмеження описаної процедури.

Особливості роботи з великими малюнками.

Великі малюнки не поміщаються в одному сегменті оперативної пам'яті, тому їх доводиться зчитувати і виводити на екран по частинах. В цьому випадку основний час затрачується не на малювання рядків, а на читання даних з файлу в оперативну пам'ять. Чим більше розмір порції даних, зчитаних за одне звернення до файлу, тим менше повторних звернень до процедури читання і тим швидше побудований малюнок. Стандартні засоби DOS, наприклад функція 3Fh переривання int 21h, дозволяють зчитати за одне звернення до диска від 1 до 65535 байт.

Розмір зчитаної порції.

Поки образ малюнка поміщається в одному сегменті, не потрібен контроль значень адрес оперативної пам'яті. Проте, при зчитуванні частини образу великого малюнка в сегменті може виявитися тільки частина останнього рядка і при її побудові без контролю значень адрес оперативної пам'яті відбудеться збій в роботі задачі. Контролювати адреси після виведення кожної точки малюнка явно не раціонально, краще при зверненні до диска зчитувати максимально можливу кількість повних рядків, що поміщаються в сегменті. Щоб встановити його, треба число 65535 розділити на розмір рядка малюнка. Після ділення результат множеться на розмір рядка, в результаті отримуємо розмір порції для читання в байтах. Остання порція, напевно, виявиться меншого розміру і це треба врахувати при побудові малюнка.

Нові змінні.

```
Swp0ffs   dw  0   ; адреса(зміщення) в буфері обміну
SwpSeg    dw  0   ; значення сегменту, що містить буфер обміну
Iwidth    dw  0   ; ширина рядка малюнка
Ineight   dw  0   ; кількість рядків в малюнку
Numbyte   dw  0   ; кількість байтів в порції даних, що зчитуються
Part      dw  0   ; кількість рядків в зчитуванній порції даних, що зчитуються
Remline   dw  0   ; кількість рядків, що залишаються невиведеними
```

При побудові малюнка використовується ці нові змінні, які повинні бути описані в розділі даних. Перші чотири з них є параметрами підпрограми. Змінні swp0ffs і swpSeg вказують повну адресу буфера обміну, в який зчитуються дані з файлу. Як резервується простір оперативної пам'яті, описано в главі 8. Значення змінних Iwidth і Iheght одержують при обробці заголовку файлу, що містить образ малюнка, який будується . Значення решти змінних формує сама підпрограма BigDraw.

Текст підпрограми, яка виконує побудову малюнка довільного розміру, приведений в прикладі 2 Перед її викликом в регістрі di треба вказати адресу лівої верхньої точки малюнка у відеопам'яті і встановити вікно, якому належить ця адреса. Регістр es повинен містити код відеосегменту (вміст змінної Vbuff).

Приклад Побудова малюнка довільного розміру.

```
BigDraw pusha                       ; зберігаємо стандартні регістри
        Push Reg <fs, cur_win>      ; зберігаємо fs і Cur_Win
        Mov fs, SwpSeg              ; fs=сегмент буфера обміну
        Mov Swp0ffs, 0              ; нульова адреса в буфері обміну
        Xor dx, dx                  ; старша частина діленого dx=0
        Mov ax, -1                  ; молодша частина діленого ax=65535
        Div Iwidth                  ; ax=65535/iwidth
```

```
            Mov  part,  ax          ; число  рядків  в  порції  даних
            Mul  iwidth             ; розмір  порції  даних в байті
            Mov  numbyte,  cx       ; зберігаємо  його  в  numbyte
            Mov  ax,  Iheight       ; копіюємо  кількість  рядків  в  малюнок
            Mov  remline,  ax       ; в лічильник невиведених  рядків
            Mov  bx,  horsize       ; bx = horsize
            Sub  bx,  iwidth        ; bx = horsize - iwidth
New Part:  mov  cx,  numbyte        ;вказуємо  розмір  порції  для  читання
            Call  readf             ; читання чергової  порції  даних
           Jnc  sucread             ; читання  без  помилок
; тут повинна виконатися дія при помилці читання.
 Sucread :  mov  cx,  part          ;cx = стандартна  кількість  рядків
            Cmp  remline,  cx       ; залишилося  менше  рядків?
            Jae  @F                 ; ні,  обходимо  команду  пересилки
            Mov  cx,  remline       ; cx = число  рядків, що залишилося
     @@ :  sub  remline,  cx        ; зменшуємо число  рядків, що залишилося
            xor  si,  si            ; адреса  початку  в  буфері обміну
  drwout  : push  cx                ;  зберігаємо  лічильник повторів
            mov  cx,  iwidth        ; задаємо  розмір  рядка  малюнка
            call  drawline          ; побудова  чергового  рядка
            pop  cx                 ; відновлюємо  лічильник  повторів
            add  di,  bx            ; адреса  початку  наступного  рядка
            jnc  @F                 ; адреса в межах сегменту
            call  NxtWin            ; установка  наступного  вікна
     @@ : loop  drwout              ; управління  повторами  циклу
            cmp  remline,  0        ; всі  рядки  виведені ?
            jne  New  Part          ; ні,  продовжуємо  побудову
            Pop Reg <Cur_win, fs>   ; відновлення  Cur_win  і  fs
            popa                    ; відновлення всіх  регістрів
            call  SetWin             ; відновлення  початкового вікна
            ret                     ; вихід
```

Виконання підпрограми починається з підготовчих дій. Дві перші команди зберігають в стеку вміст регістрів і значення змінної Cur_win, що використовуються . Потім в регістр fs поміщається сегмент буфера обміну, а змінна Swp0ffs очищається для розташування зчитаних з файлу даних з початку буфера обміну. Наступні вісім команд формують значення змінних part, numbyte, remline, остання є лічильником ще невиведених рядків, тому її початкове значення рівне висоті малюнка. Залишається сформувати в регістрі bx константу

Horsize – iwidth

для корекції адрес рядків у відеопам'яті.

Побудову малюнка виконують два вкладені цикли. Зовнішній має мітку NewPart, а внутрішній - drwount. В зовнішньому циклі проводиться читання з файлу чергової порції даних, уточнення розміру зчитаної порції (вмісту регістра cx), залишеної кількості рядків (змінна remline), та очищення регістра si. Після цього виконується внутрішній цикл, що виводить на екран зчитану порцію рядків. Внутрішній цикл повністю співпадає з аналогічним циклом прикладу 23. По закінченню роботи внутрішнього циклу перевіряється значення змінної remline і якщо воно відмінне від нуля, то відбувається повернення на початок зовнішнього циклу. Якщо всі рядки виведені на екран, то відновлюється збережені в стеку величини, початкове вікно відеопам'яті і відбувається повернення на модуль що викликається .