Тема 8 Побудова геометричних фігур: прямі лінії зліва направо, справа наліво, малювання прямокутників

Геометричні фігури є самими підходящими об'єктами для першого знайомства з КГ. Способи їх побудови залежать тільки від природи самої фігури і від відеорежиму, що використовується.

Прямі лінії

Прямі лінії бувають горизонтальні, вертикальні і похилі, від цього залежать способи їх малювання. Лінії на екрані не завжди є рівними, в більшості випадків вони східчасті. Рівною можуть бути тільки лінії, кут нахилу яких рівний нулю або кратний 45 градусам. При інших кутах нахилу лінія стає східчастою. В даному розділі ми малюватимемо рівні лінії, ілюструючи різні способи роботи з адресами точок.

Малювання лінії зліва направо.

В прикладі 8 приведені 2-а варіанти підпрограми для малювання горизонтальної лінії в напрямі зліва направо. Перед їх викликом повинно бути встановлено вікно відеопам'яті, що містить першу точку прямої, а її адреса в цьому вікні вказана в регістрі di. В регістрах cx і al поміщається, відповідно, кількість точок в лінії (довжина прямої) та їх код (колір). Тут (в прикладі) передбачається, що регістр es містить адресу відеосегменту (значення змінної Vbuff).

Приклад

Підпрограми для малювання горизонтальної лінії.

```
        ; перший варіант, використовується команда пересилки.
hor line : mov  es:[di], al   ; запис коду точки у відеобуфер
           inc  di            ; збільшення адреси на 1
           jne  @F            ; перехід, якщо не 0- перев. прапора zF
           call NxtWin        ; установка наступного вікна
   @@ :    loop hor line      ; управління повторами циклу
           ret                ; повернення з підпрограми

        ; другий варіант, використовується рядкова операція.
hor line : stos b             ; запис коду точки у відеобуфер
           or di, di          ; початок нового сегменту
           jne @F             ; ні
           call NxtWin        ; установка наступного вікна
   @@:     loop hor line      ; управління повторами циклу
           ret                ;повернення з підпрограми
```

Відмінність між варіантами підпрограми в прикладі 8 полягає в тому, що в одному випадку для запису коду точки у відеопам'ять використана звичайна команда пересилки, а в другому рядкова, яка сама збільшує вміст регістра di на 1. При корекції значення адреси може вийти за межу сегменту. В такому випадку потрібно встановити наступне вікно. Вміру запису кодів точок у відеопам'ять вміст регістра di зростає до значення 65535 (код 0FFFFh). Якщо до цієї величини додати 1, то регістр di виявиться очищеним.

В даному прикладі вперше використані локальні мітки. Опишемо правила роботи з ними. Всі локальні мітки мають ім'я @@, після якого ставиться двокрапка. В командах переходів або розгалужень замість назви локальної мітки застосовується оператори @F або @B

Оператор @F (перехід вперед) вказується, якщо локальна мітка розташована нижче по тексту. (Оператор @B – якщо локальна мітка розташована вище по тексту). Знайшовши

одного з цих операторів Макроассемблер шукає в потрібному напрямку найближчу локальну мітку. Кількість локальних міток в програмі не обмежена.

Малювання лінії справа на ліво.

Намалюємо горизонтальну лінію справа наліво. Виклик 2-ох варіантів підпрограм відрізняється від викликів в прикладі 8 тим, що початкове вікно відеопам'яті і адреса в регістрі di відповідає правій точці прямої.

Приклад 9. Малювання горизонтальної лінії справа наліво.

```
; перший варіант, використовується п команда пересилки;
invline: mov es:[di],al     ; запис коду крапки у відеобуфері;
         sub di, 01         ; зменшення адреси на 1;
         jnc @F             ; перехід, якщо немає перенесення;
         call PrevWin       ; установка попереднього вікна;
@@:      loop invline       ; управління повторами циклу;
         ret                ; повернення з підпрограми;

      ; другий варіант, використовується рядкова операція ;
invline: std                ; установка прапора напряму DF;
  invlp: stosb      ; запис коду точки у відеобуфер;
         cmp di, -1         ; початок нового сегменту?
         jne @F             ; ні;
         call PrevWin ; установка попереднього вікна;
@@:      loop invline   ; управління повторами циклу;
         cld                ; очищення прапора напряму;
         ret                ; повернення з підпрограми;
```

В прикладі 9 після запису в відеопам'ять вміст регістра di зменшується на 1, тому кожна наступна точка розташовується на екрані зліва від попередньої. Якщо при черговому зменшенні адреси буде пройдена нижня межа сегменту, то треба встановити попереднє вікно відеопам'яті. Нижньою межею поточного сегменту є нульова адреса. При її зменшенні на 1 отримаємо негативний результат, що має код 0FFFFh, який є старшою адресою попереднього сегменту. Контроль поточної адреси виконується по-різному. В першому варіанті для цього перевіряється стан С-разряда регістра прапорів після операції віднімання. При відніманні одиниці з нуля він буде встановлений, що приведе до виклику підпрограми PrevWin. В другому варіанті віднімання виконує рядкова команда, що не виробляє ознаки, тому перевіряється код результату і якщо він рівний „-1“, то викликається підпрограма PrevWin.

В першому варіанті замість команди:

```
sub di, 01
```

не можна використовувати:

```
dec di
```

оскільки остання не виробляє ознаку перенесення.

Приведені підпрограми (приклади 8-9) достатньо прості, але це не найшвидший спосіб малювання горизонтальної лінії. Перевірка поточної адреси в процесі малювання не потрібна, і виконуючі її команди можна виключити з тіла циклу запису точок. Вона (перевірка) повинна виконуватися перед циклом малювання, а не в самому циклі.

Прискорення циклу малювання

В прикладі 10 приведена підпрограма для швидкого малювання горизонтальної лінії в напрямі зліва направо. При зверненні до неї вхідні параметри задаються так само, як для підпрограм прикладу 8

Приклад Підпрограма швидкого малювання горизонтальної лінії.

```
horline: 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        ; кількість точок в поточному вікні;
         rep stosb       ; малюємо всю пряму або її початок;
         or di, di       ; адреса в межах поточного вікна?;
         jne @F          ; так, лінія намальована повністю;
        call NxtWin; установка наступного вікна;
         mov cx, dx  ; кількість не намальованих точок;
         rep stosb       ; малюємо залишок лінії;
@@:   pop dx            ; відновлюємо вміст dx;
         ret             ; повернення з підпрограми;
```

Лінія може поміщатися в поточному вікні повністю або частково. Для перевірки цього в прикладі 10 поточна адреса копіюється в регістр dx і до нього додається розмір лінії. Якщо при цьому не відбулося переповнювання, то лінія повністю поміщається в поточному вікні і регістр dx треба очистити. Якщо при складанні відбулося переповнювання, то команда jc @F виключає очищення регістра dx, оскільки в ньому знаходиться кількість точок залишку, який буде намальований після зміни вікна. Команда sub cx,dx віднімає залишок (або 0) із загального числа точок і у такий спосіб визначає кількість повторів першого мікропрограмного циклу. Наступна команда:

rep stosb

малює частину лінії, розташовану в початковому вікні відеопам'яті.

Потім перевіряється поточна адреса відеопам'яті і якщо вона відрізняється від нуля, то лінія намальована повністю і відбувається вихід з підпрограми. Якщо ж поточна адреса рівна нулю, то встановлюється наступне вікно відеопам'яті, в регістр CX копіюється вміст регістра DX і виконується команда:

rep stosb,

яка малює залишок прямої. Після цього відбувається вихід з підпрограми. Оскільки, підпрограма працює з регістром dx, то його початковий вміст зберігається в стеці і відновлюється при виході.

В даному прикладі перед установкою вікна перевіряється поточна адреса відеопам'яті:

(or di,di)

а не розмір залишку рядка (вміст регістра dx). Розглянемо результат. Якщо пряма містить N точок і повністю поміщається в поточному вікні, то при її побудові по підпрограмах прикладу 8 буде виконано 4N команд, а при побудові по підпрограмі 10 всього 10 команд (не рахуючи ret). Однією з них є команда:

(ret stosb)

яка записує N байтів у відеопам'ять. Від неї залежить час, який витратили на малювання лінії. Можна вважати, що ми скоротили цей час в 4 рази в порівнянні з прикладом 8 і це цілком виправдовує збільшення розміру підпрограми прикладу 10.

Додаткові можливості прискорення

Для подальшого прискорення процесу малювання в мікропрограмному циклі потрібно записувати коди відразу двох або чотирьох точок. Для запису точок парами замість команди

rep stosb:

треба використовувати команду

rep stosw:

заздалегідь зменшивши вміст регістра cx в два рази шляхом зсуву на один розряд вправо. Якщо в регістрі cx знаходилося непарне число, то при такому зсуві молодша одиниця коду

потрапить в с-разряд регістра прапорів (ознака переповнення). Після зсуву треба перевірити стан с-разряду і записати додаткову точку, якщо він встановлений. Для скорочення запису циклу в два рази в прикладі 10 кожну команду:

rep stosb

треба замінити наступною групою команд.

Приклад 11. Заміна команди rep stosb на rep stosw:

```
        shr cx, 01; зменшуємо кількість точок в 2 рази;
        jnc @F    ; обхід наступної команди (stosb);
        stosb     ; запис додаткової точки;
@@: rep stosw ; основний цикл запису по 2 байти;
```

Для запису кодів чотирьох точок при кожному зверненні до відеопам'яті потрібно використовувати команду:

rep stosd

заздалегідь зменшивши вміст регістра dx в чотири рази. В тих випадках, коли вміст cx не кратно чотирьом, треба додатково малювати 1, 2 або 3 точки. Для спрощення дій які виконуються, вміст cx змінюється в два прийоми, спочатку він зменшується в двічі, і якщо отримана ознака перенесення, то малюється одна додаткова точка. Потім він повторно зменшується в два рази і якщо знову отримана ознака перенесення, то малюються дві додаткові точки. Після цього можна використовувати команду rep stosd. Спосіб виконання цих дій показаний в прикладі 12.

Приклад. Заміна команди reb stosb на rep stosd:

```
        shr cx, 01; зменшуємо кількість точок в два рази;
        jnc @F    ; обхід наступної команди (stosb);
        stosb     ; запис додаткової точки;
@@: shr cx, 01; зменшуємо кількість точок в два рази;
        jnc @F    ; обхід наступної команди;
        stosw     ; запис двох додаткових точок;
@@: rep stosd ; основний цикл запису по 4-и байта;
```

При використані прискорених варіантів малювання код кольору точки потрібно записати в обидва байти регістра ax або в чотири байти регістра eax.

Зауваження: Описані підпрограми малюють одноколірні лінії. Різнокольорова лінія – це вже малюнок. Для того, щоб її точки (або групи точок) мали різні кольори, недостатньо простої зміни їх кодів в процесі малювання. Повинна бути підготовлена і встановлена палітра кольорів, відповідних кодів точок. (Про все це в третьому розділі цієї глави і п'ятій главі.)

Малювання рівних ліній.

Рівні лінії не містять східців, вони можуть бути горизонтальними, вертикальними або похилими під кутом, кратним 45^o. При їх побудові адреси суміжних точок відрізняються на деяку постійну величину.

Наприклад, у вертикальних ліній її модуль рівний значенню змінної Horsize.

В прикладі 13 приведений текст підпрограми, яка малює рівні лінії за умови, що адреси їх точок монотонно зростають. Перед її викликом треба встановити вікно відеопам'яті, в якому розташована допоміжна точка, а її адреса вказана в регістрах di. Кількість точок (довжина лінії) і їх коди (кольори), що виводяться, поміщаються, відповідно, в регістри cx і al. Крім того, в регістр bx записується приріст адреси кожної точки.

Приклад Підпрограма для малювання гладких ліній.

```
Anyline: mov es:[di],dl; запис ходу точки у відеобуфер;
         add di,bx     ; корекція поточної адреси;
```

```
        jnc @F          ; адреса в межах вікна;
        call NxtWin     ; установка наступного вікна;
@@:     lopp anyline    ; управління з повторами;
        rep             ; повернення з підпрограм;
```

При малюванні рівних ліній використовувати операцію stosb для запису кодів точок у відеопам'ять недоцільно.

Розглянемо, що саме можна намалювати за допомогою даної підпрограми. Позначимо константу переадресації, значення якої записується в регістр вх, буквою к та домовимось, що вона може бути лише додатнім числом. В таблиці 3. позитивні прирости адрес суміжних точок можуть мати чотири значення: 1, Horsize, Horsize+1, Horsize-1. Якщо в якості константи к використовувати ці значення, то відповідно будуть намальовані горизонтальна і дві похилі прямі. Останні є діагоналлю квадрата, сторона якого містить кількість точок, вказану в регістрі сх.

Підпрограма 13. дозволяє малювати пунктирні лінії. Наприклад, якщо задавати к = 2, к= 2*Horsize, к = 2*(Horsize+1), к = 2*(Horsize-1), то будуть намальовані пунктирні лінії, у яких відстань між сусідніми точками рівна 2. Проте така можливість є побічним ефектом і не представляє особливого інтересу.

Якщо Вам треба намалювати вертикальну лінію, то в прикладі 13. необхідно замінити команду:

```
add di, bx
```

на:

```
add di, Horsize
```

і змінити ім'я підпрограми.

Довільні лінії

При малюванні довільних ліній, у відмінності від рівних обчислюється приріст адреси в кожній точці. В цьому випадку при переміщенні від точки до точки значення однієї координати (х або у) збільшується на 1, а значення іншої або збільшується на 1, або залишається незмінним. В результаті на екрані виникають сходинки, розмір і кількість яких, за інших рівних умов, залежать від кута нахилу.

Розглянемо, що робить підпрограма, призначена для малювання лінії, що проходить через дві точки. Задаються координати двох крайніх точок лінії x1, y1, x2, y2. Виконання підпрограми починається з аналізу значень координат. Якщо x1> x2, то проводиться перестановка значень координат в пам'яті так, щоб x2 >x1. Потім початок координат переноситься в точку x1,y1, що дозволить працювати з приростами адрес відносно цієї точки. При виконанні подальших дій враховуються наступні величини:

DX = x2 – x1;

DY = y2 – y1;

Якщо DX = 0 або DY = 0, то задана вертикальна або горизонтальна лінія, яка будується звичайним способом. В протилежному випадку вибирається спосіб побудови лінії. Якщо DX > DY, то лінія будується відносно осі х. Це значить, що при переході від точки до точки приріст значення координати х рівно 1, а приріст координати у дорівнює DY/DX, причому 0 < DY/DX< 1.

Якщо DX < DY, то лінія будується відносно осі у. Це значить, що при переході від точки до точки приріст значення координати у рівно 1, а приріст координати х рівно DX/DY, причому 0 < DX/DУ< 1.

Якщо DX = DY, то лінія рівна, наприклад з кутом нахилу 45^o, спосіб її побудови вибирається розробником.

Лінія малюється на екрані послідовно точка за точкою, тому значення координат в будь-якій точці дорівнює сумі приростів їх значень всіх попередніх і в даній точці. В одних випадках ці суми обчислюються явно, в інших неявно, але без них лінію намалювати неможливо. Значення однієї координати (х або у) є цілими числами і не вимагають

додаткових перетворень. Значення іншої координати (y або x) можуть мати дробову частину, тому їх треба перетворювати в цілі числа.

Приріст значень координат перетвориться в приріст адреси, і чергова точка виводиться на екран. Знак приросту залежить від взаємного розташування крайніх точок лінії, тому в підпрограмах враховується можливість як додатніх, так і відє'мних приростів адрес.

Опис алгоритмів малювання ліній і приклади підпрограм є в книзі Уїлтон Р. Відеосистеми персональних комп'ютерів IBM і PC PC/2, призначених для роботи у відеорежимах VGA.

Прямокутники.

При малюванні прямокутників можна переслідувати дві різні мети– замальовування (заповненя) прямокутної області екрану вибраним кольором або малювання сторін (контуру) прямокутника. Перша задача більш загальна, тому розглянемо способи її рішення.

Смуга заданого кольору

Припустимо, що ширина прямокутника рівна ширині робочої області екрану (Horsize), а її висота (товщина) складає N точок. В прикладі приведений текст підпрограми, яка послідовно малює задану кількість горизонтальних ліній завдовжки Horsize, внаслідок чого виходить смуга потрібної висоти.

Перед викликом підпрограми треба встановити вікно відеопам'яті, в якому знаходиться лівий верхній кут смуги, а її адреса в цьому вікні записується в регістр di. Кількість рядків в смузі вказується в регістрі cx. Для малювання рядків підходить будь-який варіант підпрограми horline, описаної в попередньому розділі. Залежно від того, який з них вибраний код кольору указується тільки в регістрі al, в обох байтах регістра ax або в чотирьох байтах регістра eax.

Приклад Замальовування прямокутної смуги.

```
Stripe : Push Reg  <di,cx,Cur_win>      ; збереження di, cx, Cur_win
  Fillbar:  Push  cx                     ; збереження поточного значення
            Mov  cx,  horsize            ; завдання розміру рядка
            Call  horline                ; виведення чергового рядка
            Pop  cx                      ; відновлення лічильника рядків
            Loop  fillbar                ; управління виведення рядка
            Pop Reg  <Cur-win,  cx,  di>  ; відновлення
            Call  SetWin                 ; відновлення початкового вікна
            ret
```

Виконання підпрограми прикладу 14 починається із збереження в стеку вмісту регістрів di, cx і змінної Cur_win. Замальовування проводиться в циклі, має мітку fillbar. Регістр cx використовується в цьому циклі як лічильник. Крім того, в ньому передається розмір рядка для підпрограми horline. Тому на початку циклу вміст cx зберігається в стеці і відновлюється після повернення з horline. Завдяки цьому команда loop fillbar працює з тією величиною ,яка була вказана в регістрі cx при зверненні до підпрограми stripe.

Після завершення циклу fillbar відновлюються збережені в стеку величини і початкове вікно відеопам'яті, для чого викликається підпрограма SetWin. Ret – повернення з підпрограми. В даному прикладі ширина смуги дорівнювала ширині робочої області екрану. У такому разі після запису останньої точки поточного рядка автоматично відбувається перехід на початок наступного рядка, для цього не потрібні ніякі додаткові дії підпрограми.

Якщо ж ширина прямокутної області менше horsize, то адресу початку наступного рядка повинна встановлювати підпрограма.

Обчислення адреси рядка.

Перші точки рядків прямокутної області знаходяться в одному стовпці, тому в режимах PPG адреси початку суміжних рядків розрізняються на величину horsize. Отже, якщо адресу початку поточного рядка збільшити на horsize, то одержимо адресу початку наступного рядка

Можна зберегти адресу початку поточного рядка і в потрібний момент збільшити його на Horsize. Це не найпростіший спосіб, тому що, якщо при обчисленні адреси початку наступного рядка відбувається переповнювання, то треба встановити нове вікно відеопам'яті. Але воно вже могло бути змінено при малюванні поточного рядка і повторна зміна неприпустима. Тому, перш ніж встановити нове вікно, треба перевірити, чи не змінилося воно при малюванні рядка. Таким чином, цей спосіб поганий тим, що невідомо, якому вікну відеопам'яті відповідає збережена адреса і потрібні додаткові перевірки для з'ясування цієї обставини. Зайві перевірки можна виключити, якщо змінити спосіб корекції так, щоб використовувалося поточне значення адреси, яке явно відповідає встановленому вікну відеопам'яті. Після виведення останньої точки рядка поточна адреса більша адреси її першої точки на ширину прямокутної області. Отже, додавши до нього значення змінної horsize, зменшене на ширину прямокутної області, отримаємо адресу початку наступного рядка.

Якщо при складанні відбувається переповнювання, то встановлюється наступне вікно відеопам'яті без перевірки яких-небудь додаткових умов.

Замальовування прямокутної області.

В прикладі 15 приведена підпрограма, що замальовує заданим кольором прямокутну область довільного розміру. Перед її викликом адреса лівого верхнього кута прямокутника поміщається в регістр di, встановлюється вікно відеопам'яті, якому належить ця адреса. Ширина прямокутника (кількість точок в рядку) поміщається в регістр dx, а висота (кількість рядків або точок по вертикалі) поміщається в регістр cx. Задання коду кольору точок залежить від того, який варіант підпрограми horline використовуватимемо. Horline може записуватися у відеопам'ять байти, слова і подвійні слова, відповідно один і той же код кольору вказується в регістрі al, в обох байтах регістра ax або в чотирьох байтах регістра eax.

Приклад Підпрограма замальовування прямокутної області.

```
Retngl: Push Reg <bx, cx, di, Cur_win>   ; збережемо в стеку
        Mov bx, horsize                  ; копіюємо horsize в регістр вх
        Sub bx, dx                       ; віднімаємо ширину прямокутника
        Fillar : push cx                 ; зберегли лічильник повторів
           Mov cx, dx                    ; завдання розміру рядка
           Call horline                  ; малюємо лінію
           Pop cx                        ; відновлення з лічильника
           Add di, bx                    ; адреса початку наступного рядка
           Jnc @F                        ; адреса в межах вікна
           Call NxtWin                   ;перехід до наступного вікна
    @@ : loop fillar                     ; управління повторами циклу
        pop reg <Cur_Win, di, cx, bx>    ; відновлення із стека
        call SetWin                      ; відновлення початкового вікна
        ret
```

Виконання прикладу 15 починається із збереження в стеку вмісту регістрів і змінної Cur_Win, що використовуються. Після цього за допомогою двох команд в регістрі вх

формується константа для корекції адрес рядків. Вона рівна різниці між значеннями змінної Horsize і шириною прямокутника, вказаної в dx перед викликом підпрограми.

Замалювання прямокутної області проводиться в циклі, що має мітку fillar. Він відрізняється від циклу fillbar (в попередньому прикладі) тим, що після малювання кожного рядка проводиться корекція поточної адреси (команда add di, bx). Якщо при додаванні не відбувається переповнювання результату, то нове значення адреси знаходиться в межах сегменту і команда

JNC @F

виключає виклик процедури NxtWin.

У разі переповнювання умовний перехід не виконуються, і відбувається установка наступного вікна. Після виходу з циклу fillar, перед поверненням в модуль що викликається, відновлюються збережені в стеку величини і збережене вікно відеопам'яті.

Малювання контуру прямокутника.

Контур прямокутника складається з 2-ох вертикальних і 2-ох горизонтальних ліній, тому для його малювання знадобляться підпрограми horline і anyline.

Вважатимемо, що допоміжною точкою є верхній лівий кут контуру, адреса якого відома. Якщо грані малювати в такому порядку – верхня, права, нижня, ліва, то обчислювати адресу початку граней взагалі не буде потрібно. Але у такому разі буде потрібно не дві, а чотири підпрограми. При малюванні верхньої і правої граней адреси відеопам'яті будуть змінюватися в природному порядку у бік їх збільшення і можна використовувати підпрограми horline і anyline. При малюванні ж нижньої і лівої грані адреси відеопам'яті будуть змінюватися у бік їх зменшення і знадобиться ще дві підпрограми. Які малюють лінії у зворотньому напрямі. Щоб обмежитися двома підпрограмами, необхідно змінити послідовність малювання граней.

Спочатку малюємо верхню і праву грані, потім повертаємось в лівий верхній кут контуру прямокутника, і малюємо ліву і нижню грані. Можна спочатку намалювати ліву і нижню грані, а потім верхню і праву. В обох випадках при малюванні граней, адреси відеопам'яті змінюється у бік їх збільшення.

В прикладі 16 приведена підпрограма, яка малює спочатку верхню і праву, а потім ліву і нижню грані. Вхідні параметри для неї співпадають з параметрами підпрограми 15

Приклад Підпрограма малювання контуру прямокутника.

```
Round: Push Reg  <bx,  Cur_win,  cx,  di >   ; збереження
       Mov  cx,  dx                          ; cx = ширина  прямокутника
       Dec  cx                               ; зменшуємо  ширину  на 1
       Call  horline                         ; малюємо  верхню  грань
       Mov  bx,horsize                       ; вх = horsize
       Pop  cx                               ; відновлює вміст cx
       Push  cx                              ; і зберігаємо  його  в  стеку
       Call  anyline                         ; малюємо  праву  грань
       Pop Reg  <di,  cx,  Cur_win >         ; відновлюємо початковий стан
       Push Reg  <Cur_win,  cx,  di >        ; заповнюємо  знову
       Call  SetWin                          ; встановлюємо  початкове  вікно
       Dec  cx                               ; зменшуємо  висоту  на  1
       Call  anyline                         ; малюємо  ліву  грань
       Mov  cx,  dx                          ; cx = ширина  прямокутника
       Call  horline                         ; малюємо  нижню  грань
       Pop Reg  <di,  cx,  Cur_Win,  bx>     ; відновлення  початкового стану
       Call  SetWin                          ; відновлення  початкового  вікна
       Ret                                   ; повернення
```

Виконання прикладу 16 починається із збереження в стеку змінної Cur_Win і регістрів bx, cx, di. При виклику підпрограми змінна Cur_Win і регістр di задають адресу лівого верхнього кута контуру прямокутника, а в регістрі cx указується висота прямокутника (кількість точок по вертикалі).

При малюванні верхньої грані її розмір скорочується на одну точку, а для того, щоб при поверненні з підпрограми horline в регістрі di знаходиться адреса першої точки правої грані. Праву грань малює підпрограма anyline, тому в регістр bx треба занести значення horsize, а із стека відновиться і тут же знову зберегти в ньому вміст регістра cx. Після повернення з підпрограми anyline будуть намальовані верхня і права грані. Тепер треба повернуться в лівий верхній кут контуру прямокутника, відновивши початковий стан, збережений в стеку, і наново зберегти його для використовування при виході з підпрограми. Крім того, відновлюється початкове вікно, оскільки воно могло змінитися при малюванні.

При малюванні лівої грані її розмір скорочується на 1, завдяки цьому при поверненні з підпрограми anyline регістр di містить адресу першої точки нижньої грані. В даному випадку записувати в регістр bx значення Horsize не вимагається, оскільки воно було записано туди раніше. Після повернення з підпрограми anyline в регістр cx указується ширина прямокутника, і підпрограма horline малює нижню замикаючу грань. Залишається відновити збережені величини, початкове вікно відеопам'яті і виконати повернення з підпрограми.

Описана програма малює прямокутний контур, ширина граней якого рівна одній точці. Якщо грані повинні бути більш широкими, то можна намалювати декілька вкладених прямокутних контурів так, щоб отримати грані потрібної ширини. Можна зробити інакше – намалювати дві вкладені прямокутні області. Спочатку малюється більша область, колір якої співпадає з кольором граней, а потім в ній менша область, яка має колір внутрішньої частини прямокутника.