

Материалы к занятию 25 марта 2020

Тема: Использование тригонометрических функций, использование команд сопроцессора, прикладная задача (соответствующие программы находятся в папке 20_03_18, а также прилагаются в архиве с этим текстом).

На первом этапе рассмотрим построение окружности с использованием основного набора команд процессора (прилагаемая программа circle0.asm – см. ниже).

```
; Написать программу, которая строит окружность
Assume CS: Code;, DS: Code
Code    SEGMENT
        .286
        org 100h
Start:   jmp     start1
; Переменные
mode db      04      ; Видеорежим      (05,06      (один
цвет),0dh,0eh,10h, 12h)
col      db      2 ; Цвет
x0       dw      100; X центра
y0       dw      100; Y центра
x        dw      ? ;Текущая X на окружности
y        dw      ? ;Текущая Y на окружности
alfa     db      1 ; Начальный угол
rad      db      60 ; Радиус
start1:
; Установка видеорежима
        mov      ah,0 ; Функция установки видеорежима
        mov      al,mode ; Видеорежим
        int      10h
st2:     call     circle
        call     kbin
        int      20h
; Построение окружности радиусом RAD
; из центра X0, Y0
circle proc near
; Цикл по углу
cir1:    mov      al,alfa ;Угол
        call     cos      ;(-128 - +127)
        rol      al,1      ;Значение cos*2, знак в CF
        jc       cir2      ;Переход, если cos<0
;Здесь положительный cos
;Вычисление следующего X
        mov      dl,rad ;Радиус
        mul      dl      ;dX=R*cos(alfa)
        xor      al,al ;Нормировка в байте
```

```

        xchg     al,ah    ;dX->AX
        add      ax,x0    ;AX=X0+dX
        mov      x,ax     ;X=X0+dX
        jmp      cir3     ;На вычисление Y
;Здесь отрицательный cos
cir2:    neg      al      ;Инверсия cos
        mov      dl,rad   ;Радиус
        mul      dl      ;dX=R*cos(alfa)
        xor      al,al    ;Нормировка в байте
        xchg     al,ah    ;dX -> AX
        mov      bx,ax    ;BX=dX
        mov      ax,x0    ;X центра
        sub      ax,bx    ;AX=X0-dX
        mov      x,ax     ;X=X0-dX
;Вычисление Y
cir3:    mov      al,alfa  ;Угол
        call     sin      ;(-128 - +127)
        rol      al,1     ;Значение sin*2, знак в CF
        jc       cir4     ;Переход, если sin<0
;Здесь sin положительный
        mov      dl,rad   ;Радиус
        mul      dl      ;dY = R*sin(alfa)
        xor      al,al    ;Нормировка в байте
        xchg     al,ah    ;AX=dY
        add      ax,y0    ;AX=Y0+dY
        mov      y,ax     ;Y=Y0+dY
        jmp      cir5     ;Переход на построение точки
;Здесь синус отрицательный
cir4:    neg      al      ;Инверсия sin
        mov      dl,rad   ;Радиус
        mul      dl      ;dY=R*sin(alfa)
        xor      al,al    ;Нормировка в байте
        xchg     al,ah    ;AX=dY
        mov      bl,al    ;BL=dY
        mov      ax,y0    ;AX=Y0
        sub      al,bl    ;AX=Y0-dY
        mov      y,ax     ;Y=Y0-dY
cir5:    call     pix     ;Вызов проц.построения точки
;Следующее значение угла
        inc      alfa    ;Инкремент угла (1=5,625 град.)
        cmp      alfa,65  ;Всего 64 угловых сдвига
        jz       cir6     ;Дошли до конца
        jmp      cir1     ;Еще не конец, на начало цикла
cir6:    ret
circle  endp

```

```

        ;Ввод с клавиатуры (стандартный)
kbin    proc    near
        mov     ah,0    ; Функция 0
        int     16h    ; клавиатурного прерывания
        ret
kbin    endp
;Построение точки функцией BIOS
pix:pusha                ;Сохранение всех РОН
        mov     ah,0ch  ;Функция построение точки
        mov     dx,y;Номер граф. строки
        mov     cx,x;Номер граф. столбца
        xor     bx,bx    ;Текущая граф. страница
        mov     al,col   ;Цвет
        int     10h    ;Видеоперерывание
        popa          ;Восстановление всех РОН
        ret           ;Возврат
sin      proc    near
; вход AL - угол (0 - 63)
; выход AL - SIN (-128 - 127)
        lea     bx,sintab
        xlat
        ret
sin      endp
cos      proc    near
; вход AL - угол (0 - 63)
; выход AL - COS (-128 - 127)
        lea     bx,cosstab
        xlat
        ret
cos      endp
; Тригонометрические таблицы
SINTAB                                     db
0,12,24,37,48,60,71,81,90,98,106,112,118,122,125,127
COSTAB                                     db
127,127,125,122,118,112,106,98,90,81,71,60,49,37,25,12
db
0,244,232,219,208,196,186,175,166,158,150,144,138,134,1
31,129
db
129,129,131,134,138,144,150,157,166,175,185,196,207,219
,231,244
db
0,12,24,36,48,60,70,80,90,98,106,112,118,122,125,127
db 127 ; Лишнее 65 значение для COS
code    ends

```

END Start ; Указание точки входа в программу

При разборе программы не будем останавливаться на стандартных элементах (assume, объявление сегмента и прочих стандартных функциональных элементах). Здесь отметим лишь, что в инструкции assume указан лишь сегмент кода, а сегмент данных закомментирован. Так тоже можно, это лишь немного усложняет задачу компилятора.

Сразу после определения сегмента кода написана инструкция .286 (можно было написать .186, .386, .486 и т. д.), которая позволяет использовать команды до 285 процессора включительно. Но она нужна лишь для использования команд **pusha** и **popa** (сохранения и восстановления всех регистров общего назначения в процедуре **pix**, строки 97 и 104).

Затем идет перескок через переменные (**Start: jmp start1**). Напомню, что перед этой (первой в программе настоящей командой) стоит метка, которая будет указана в качестве точки входа в программу (**end start**) в строке 128.

Первая переменная – номер видеорежима (отметим, что режимы 4, 5 и 6 – режимы **CGA** с разрешением 320 x 200 точек и четырьмя цветами 0-3, режим **0dh** – режим **EGA** с разрешением 320 x 200 точек и 16 цветами, режим **0eh** – режим **EGA** с разрешением 640 x 350 точек и 16 цветами, режим **10h** – режим **EGA** с разрешением 640 x 350 точек и 4 цветами, режим **12h** – режим **VGA** с разрешением 640 x 480 и 16 цветами – все это есть в **rhelp**'е на сайте).

Далее идет переменная **col** – цвет (при экспериментах не забывайте, что количество возможных цветов в различных режимах различное).

Затем идут координаты центра строящейся окружности (**x0** и **y0**) и текущие координаты на окружности (**x** и **y**).

Затем идет переменная начальный угол (**alfa**), с которого начинает строиться окружность. При **alfa**=1 окружность строится полностью. При **alfa**=32, будет построено половина окружности. Здесь следует сказать, что **alfa** задается в условных единицах (от 0 до 64), то есть каждая единица дает приращение угла на 5,625 градусов. Количество шагов выбрано равным степени двойки (2⁶) исходя из элементарного удобства.

Последняя переменная **rad** – радиус строящейся окружности.

Понятно, что все линейные размеры (координаты и радиус) задаются в графических точках экрана (напомню, что экранные координаты начинаются в левом верхнем углу экрана и с приращением **X** и **Y** идет перемещение вправо и вниз).

Затем идет метка **start1**: собственно начала программы.

В первую очередь, устанавливается видеорежим соответствующей функцией **BIOS**. В исходном варианте там стоит режим **12h** (**VGA** с разрешением 640 x 480 и 16 цветами).

Затем идет вызов процедуры построения окружности (**call circle**). Следом идет вызов процедуры ввода с клавиатуры (**call kbin** – с ожиданием, чтобы увидеть картинку), и затем выход из программы.

На рисунке 1 схематически показано, как на экране строится окружность. Имеется в виду исходный видеорежим **VGA 640x480**. Для других видеорежимов правый и нижний края экрана будут иметь другие максимальные значения.

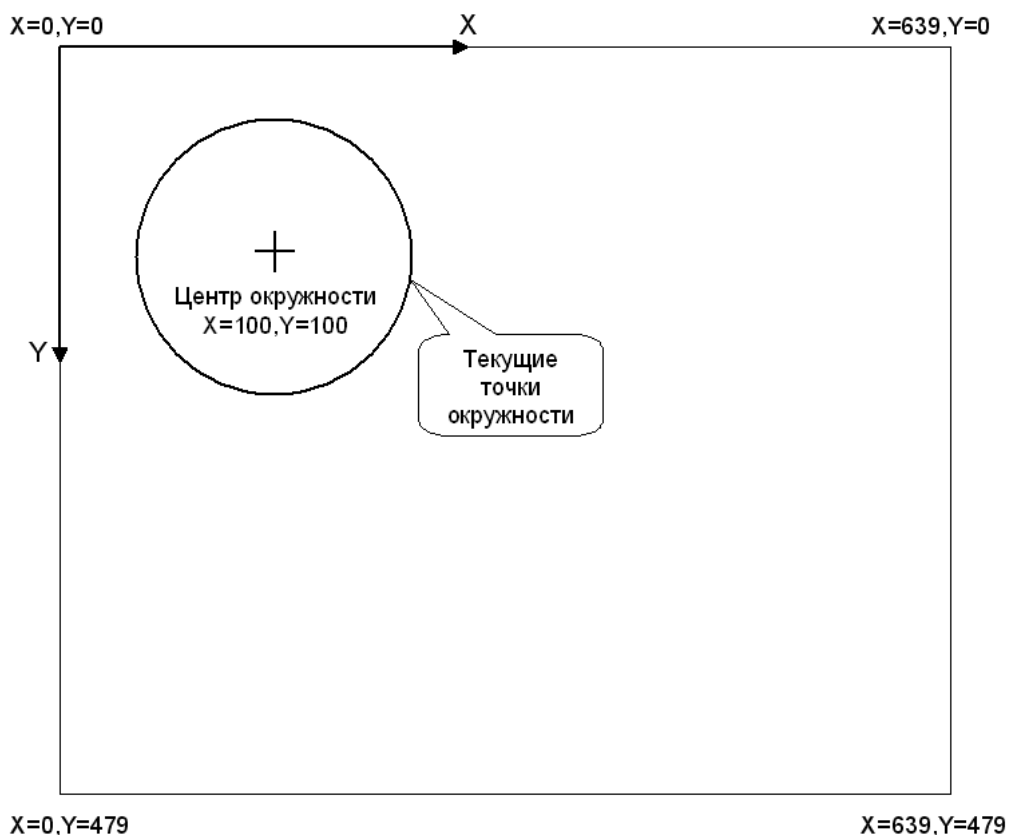


Рисунок 1 – Схема графического экрана в режиме VGA 640x480 точек (пикселей)

Процедура построения окружности (**circle proc near**) начинается с вычисления первой точки окружности (**cir1: mov al,alfa**) – метка нужна для замыкания цикла по значениям угла. В **AL** загружается текущее (в первый момент начальное, заданное в переменной) значение угла (не забываем, что угол здесь выражается в единицах по 5,625 градусов). Затем вызывается процедура вычисления косинуса (**call cos**). Как будет видно далее, косинус вычисляется табличным методом с помощью таблицы, которая была предварительно составлена. Значение косинуса имеет размер 1 байт, поэтому диапазон его представления от -128 до $+127$. Так как синус может быть, как положительным, так и отрицательным, следующие две команды определяют знак синуса. Команда **rol al,1** сдвигает значение в **AL** циклически на один разряд влево. При этом значение знака (если минус, то 1) сдвигается во флаг переноса (так как сдвиг циклический), модуль значения синуса при этом нормируется относительно байта (то есть, единичному значению синуса будет соответствовать значение 255). Следующая команда (**jc cir2**) выполняет переход, если значение синуса отрицательное.

Затем выполняются вычисления приращения **X** и **Y** по схеме, изображенной на рисунке 2.

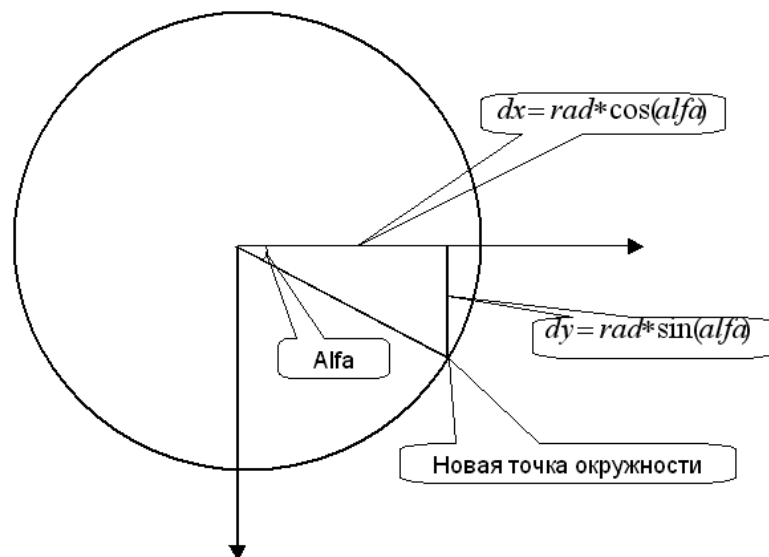


Рисунок 2 – Схема вычисления координат новой точки окружности

Команды строк с 34 по 39 выполняют вычисления **dX**, показанные выше, и записывают вычисленное значение в переменную **X**. Здесь стоит пояснить команды в строках 36 и 37. После умножения радиуса на косинус угла в аккумуляторе получается 16-разрядное значение. При единичном значении косинуса получается значение радиуса, умноженное на 255 (так как единичному значению косинуса соответствует именно это значение). То есть, практически в старшем байте мы имеем радиус. При других значениях косинуса в старшем байте будет радиус, умноженный на соответствующее значение косинуса. Для получения этого значения в младшем байте аккумулятора мы обнуляем младший его байт (строка 36), а затем меняем местами значения старшего и младшего байтов (строка 37). В итоге в **AL**, а следовательно и в **AX** получается значение 8-разрядное вычисленное значение (так как старший байт аккумулятора обнулен). Затем командой **jmp cir3** (строка 40) выполняется переход на аналогичное вычисление **dY**.

Если косинус был отрицательным, то после перехода на **cir2:** (строка 42) выполняется инверсия косинуса (то есть, взятие модуля от значения) командой **neg al**. Затем выполняются вычисления, соответствующие положительному значению косинуса, с той только разницей, что вычисленное значение **dX** не добавляется к **X0**, а вычитается из него (**sub ax,bx** – строка 49). Следует отметить, что схема вычислений немного отличается от предыдущей. Так как в предыдущем случае **dX** вычислялся в аккумуляторе, то затем к нему просто добавлялся **X0**. В случае с вычитанием, необходимо из **X0** вычитать **dX**, поэтому используется промежуточное хранилище для **dX** (регистр **BX**), затем в аккумулятор загружается **X0**, а из него затем вычитается содержимое регистра **BX**.

Затем в строке 52 идет метка **cir3:**, на которую происходит переход со строки 40 при вычислении с положительным косинусом, или непосредственно после вычислений с отрицательным.

Здесь по той же схеме, что и для **dX**, вычисляется **dY** (при положительном или отрицательном значении синуса). Обе ветки

завершаются присвоением переменной **Y** значения **dY** и переходом к строке 74 (с меткой **cir5**), в которой вызывается процедура построения очередной точки окружности на экране.

Затем (строка 77 – команда **cmp alfa,65**) производится сравнение текущего значения угла (условного) со числом 65. Если значение угла =65, значит ничего строить уже не надо (у нас всего 64 значения тригонометрических функций). В этом случае выполняется переход на выход (строка 78 – команда **jz cir6**).

Если переход не выполняется, значит конец еще не достигнут, выполняется переход на начало цикла по углу (строка 79 – команда **jmp cir1**).

Далее в строке 81 выход из процедуры **ret** (не забываем, что компилятор заменит эту команду на **retn**, так как процедура ближняя).

Ниже расположена стандартная процедура ввода с клавиатуры с ожиданием (функция 0 клавиатурного прерывания **int 16h**). Ее мы уже неоднократно рассматривали.

Затем идет процедура вывода графической точки на экран. Прежде всего в стеке сохраняются все регистры общего назначения (для этого нужна была **.286**). Затем используется биосовская функция вывода графической точки на экран (**0ch**). В ней в **DX** указывается номер графической строки строящейся точки (вертикаль, от 0 до 479), в **CX** – номер графического столбца (горизонталь, от 0 до 639), в **BX** – текущая графическая страница (0), в **AL** цвет точки.

Ниже расположены процедуры получения синуса и косинуса по аргументу (углу). Здесь используется очень полезная для табличной перекодировки команда (**xlat**). Для нее в регистр **BX** помещается адрес начала таблицы перекодировки (строка 101 – команда **lea bx,sintab**), а в **AL** помещается номер элемента таблицы (в нашем случае это значение угла в условных единицах). После выполнения команды **xlat** (строка 102) в **AL** оказывается значение синуса, которое находилось на указанном месте таблицы.

Следом идет аналогичная процедура вычисления значения косинуса.

После процедур идут тригонометрические таблицы. Как видно из листинга, таблица косинусов начинается на 16 значений позже, чем таблица синусов. Связано это с тем, что $\cos(\alpha) = \sin(\alpha + \frac{\pi}{2})$, то есть косинус – это синус, сдвинутый на 90° (четверть от 360° , что соответствует 16 значениям из 64). Создавалась таблица очень просто: первое значение соответствует синусу 0 град., второе – синусу 5,625 град., третье – синусу $2 \times 5,625 = 11,250$ град и т. д. Все эти значения надо было вычислить заранее.

Результат работы этой программы в **DosBox**'е показан на рисунке 3.

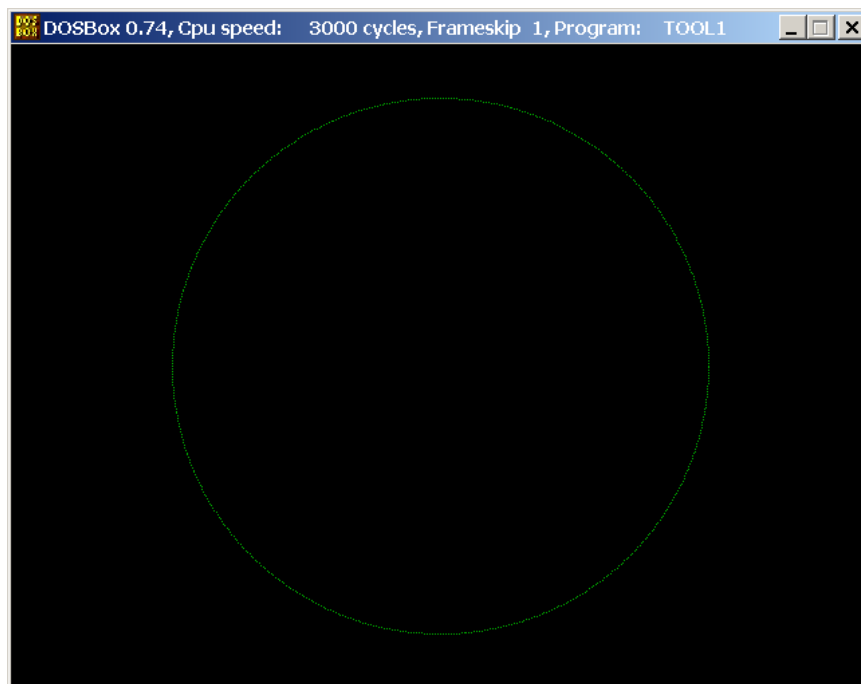


Рисунок 3 – Построение окружности функциями BIOS

Эксперименты, которые можно выполнить с программой:

- попробовать поменять координаты центра окружности (**x0, y0**),
- попробовать поменять значение радиуса (**rad**),
- попробовать поменять значение начального угла (**alfa**); напомним, что при **alfa=32** изобразится только верхняя половина окружности (почему?),
- попробовать поменять видеорежим и/или цвет (**mode, col**), при этом учитывайте, сколько цветов поддерживает тот или иной режим.

Использование сопроцессора

Существенно удобнее использовать тригонометрические функции, реализуемые сопроцессором. Система команд сопроцессора наиболее внятно описана здесь: <https://prog-cpp.ru/asm-coprocessor-command/>

Приведенная ниже программа находится в одном архиве с этим текстом в каталоге **Circle01** (основной модуль **circle01.asm**, включаемый модуль – **mac1.asm**).

```
;Построение окружности радиусом
;с использованием команд сопроцессора mod_c      equ
           12h;12h графический 640*480 16 цветов
Assume CS: Code, DS: Code
Code SEGMENT
    .486
    org 100h
include mac1.asm ; Подключение макроса
Start:    jmp begin
ry  dd    1.0 ;масштабные коэффициенты
rx  dd    1.0;0.87
```



```

x0 dd 320.0 ;середина по X и Y
y0 dd 240.0
rad dd 200.0 ;Радиус окружности
alpha dd 0.0 ;Угол
delta dd 0.01 ;величина изменения
tmp dd 0 ;temp
xr dw 0 ;координаты выводимой точки
yr dw 0
begin:
    call mode; Установка видеорежима
    PutCircle rad
    mov ah,0 ; Ожидание ввода с клавиатуры
    int 16h
    int 20h ; Выход из программы
modeprocnear;
    mov ah,0; Функция установки видеорежима
    mov al,mod_c; Значение режима
    int 10h
    ret
modeendp
Code ends
end Start

```

Подключаемый файл макросов.

```

;Макрос вывода окружности радиусом r на экран
PutCircle macro r
    finit ;инициализация сопроцессора
;x=round(rx*cos(alpha))
    mov cx,629 ;Счетчик цикла
l1: fld alpha ;Загрузка alpha в вершину стека
    fcos;Вычисление косинуса и помещение в вершину
стека
    fld r
    fmul
    fld rx ;Загрузка rx в вершину стека
    fmul ;Умножение вершины стека на следующее в
стеке
    ;значение и оставление в вершине стека
    frndint ;Округление вершины стека до целого
    fld x0 ;Загрузка середины X в стек
    fadd ;Сложение вершины стека со следующим
значением
    fistp word ptr xr ;Сохранение вершины стека в xr
;y=round(ry*sin(alpha))
    fld alpha ;Загрузка alpha в вершину стека

```

```

        fsin                ;Вычисление синуса и помещение в
вершину стека
fld r
fmul
        fld ry              ;Загрузка rx в вершину стека
        fmul                ;Умножение вершины стека на следующее в
стеке
        ;значение и оставление в вершине стека
        frndint             ;Округление вершины стека до целого
        fstp tmp            ;Сохранение вершины стека в tmp
        fld y0              ;Загрузка середины Y в стек
        fsub tmp            ;Вычитание из вершины стека tmp
(выч.знач.Y)
        fistp word ptr yr   ;Сохранение вершины стека в yr
        PutPix xr,yr,2      ;выводим точку зеленым цветом
        fld delta           ;вычисляем новое значение alpha
        fld alpha
        fadd
        fstp alpha
    loop    l1              ;цикл по cx
endm
;-----
;Макрос вывода графической точки относительно осей
;Параметры - граф.коорд. X, Y, цвет
PutPix macro x,y,col
    pusha
    mov ah,0ch              ;Вывод граф.точки на экран
    mov al,col              ;Цвет
    mov bh,0h               ;Текущая видеостраница
    mov cx,x;Коорд.X (горизонталь)
    mov dx,y;Коорд. центра
    int 10h                 ;Выполнение функции
    popa
endm

```

Уже по приведенным листингам видно, что текст программы существенно короче, чем предыдущая. Первое, что бросается в глаза – отсутствие таблиц синусов и косинусов (все средства для вычисления тригонометрических функций есть в сопроцессоре).

Разберем сначала основной модуль (**circle01.asm**).

Начинается она с определения константы (**mod_c equ 12h**), определяющей используемый видеорежим. Напомним, что есть два способа назначения констант **equ** и **=**. Отличаются они тем, что **equ** присваивает константу навсегда (в пределах программы), а константа, назначенная при помощи «=», может переопределяться. Кроме того, с помощью **equ** можно присваивать числовые и символьные значения, а с помощью «=» – только

числовые. Расположение этого определения константы в первой же строке иллюстрирует тот факт, что это присвоение может находиться в любом месте программы.

Далее идут стандартные строки 4-7, которые объяснения не требуют (единственное отличие от предыдущих программ – здесь в строке 6 стоит **.486**, а не **.286**, как ранее, но это не имеет значения (так больше понравилось)).

Дальше идет подключение файла с макросами (строка 8 – **include mac1.asm**).

На строке 10 (**Start: jmp begin**) находится метка точки входа в программу и команда перескока через переменные.

Далее, в строках 10-19 определены переменные, аналогичные переменным в предыдущей программе, но есть некоторые отличия.

Во-первых, появились масштабные коэффициенты. Для чего они нужны. Если Вы проделали эксперименты с предыдущей программой, то заметили, что окружность похожа на собственно окружность только в режиме, соответствующем пропорциям используемого монитора (у меня это **VGA 640x480**). В остальных режимах получается скорее эллипс, вытянутый по вертикали или горизонтали. Масштабные коэффициенты **ry** и **rx** позволяют ввести раздельную коррекцию по осям. Следует также обратить внимание на то, что переменные (строки 10-17) определены как двойное слово (один из стандартов представления чисел с плавающей точкой в сопроцессоре), а из значения (в том числе, и значения масштабных коэффициентов) указаны с плавающей точкой.

Переменные в строках с 12 по 15 те же, что и в предыдущей программе (за исключением формата **dd** и значений с плавающей точкой) – координаты центра окружности, радиус, начальный угол.

Далее идет переменная, которой не было в предыдущей программе – **delta** – приращение угла в цикле. В предыдущей программе мы были привязаны к сформированной нами же таблице синусов-косинусов, поэтому шаг был естественным (1). Здесь мы к таблицам не привязаны, поэтому можем дать любой, как угодно маленький, шаг (в данном случае это 0,01), что обеспечит достаточно плавную окружность на экране.

Далее определена вспомогательная переменная **tmp**, необходимая из-за того, что в сопроцессоре все операции выполняются в стеке, и иногда приходится хранить промежуточные значения.

Последние две переменные – координаты выводимой точки **xr** и **yr** (аналогичные переменным **x** и **y** в предыдущей программе), которые определены как целые значения формата слово (так как вся экранная графика целочисленная).

Далее начинается тело самой программы, начинающееся с метки **begin:** (строка 20).

Основной текст очень короткий:

- вызов установки видеорежима (строка 21 – **call mode**),
- собственно построение окружности (строка 22 – **PutCircle rad**), где выполняется соответствующий макрос с параметром **rad** (радиус),

- стандартное ожидание нажатия клавиши, чтобы увидеть картинку (функция 0 клавиатурного прерывания **int 16h**),
- выход из программы.

Далее определена процедура установки видеорежима. Она полностью совпадает с такой же процедурой в предыдущей программе, за исключением переменной **mod_c**, поэтому описывать ее здесь смысла нет.

Теперь начинается самое интересное. Подключаемый файл макросов.

Первым расположен макрос построения окружности радиусом **r**.

Название макроса **PutCircle macro r** (строка 3). Название макроса **PutCircle** выбирается произвольно (желательно сз латиницы), **macro** – ключевое слово, обозначающее начало описания макроса, **r** – формальный параметр, вместо которого при вызове макроса подставляется фактически указанное значение (в нашем случае **rad** – строка 22 основного файла программы).

Далее пошли команды работы с сопроцессором (см. ссылку выше). Эти команды легко отличаются из-за наличия буквы **f** (означает «плавающий») в начале команды.

Первая команда (строка 6 – **finit**) – инициализация сопроцессора, при которой сбрасываются все указатели и очищаются все регистры.

Затем в строке комментария (строка 5 – **;x=round(rx*cos(alpha))**) поясняются вычисления (они ничем не отличаются от вычислений в предыдущей программе, за исключением формата используемых значений).

Затем инициализируется счетчик цикла (строка 6 – **mov cx,629**), определяющий количество строящихся точек окружности. Счетчик цикла **CX**, так как для организации цикла будет использована команда **loop** (строка 36). Количество точек выбрано таким, чтобы окружность замкнулась. Значение выбиралось приблизительно. Если количество точек задано больше, чем нужно, то повторяющиеся точки совпадут с построенными ранее, а на длительность вычислений это практически не влияет (если, конечно, не задать безумно большое количество точек) из-за высокой производительности современных процессоров.

Для понимания дальнейшего изложения следует пояснить, что все операции в сопроцессоре выполняются в стеке. Вершина стека сопроцессора в каком-то смысле эквивалентна аккумулятору обычного процессора (в смысле значимости). Если команда двухоперандная, то первый операнд по умолчанию (если не указан в команде) располагается в вершине стека сопроцессора — регистре **ST(0)**, и на его место после выполнения команды записывается результат. Второй операнд может быть расположен либо в памяти, либо в другом регистре стека сопроцессора. По умолчанию в качестве второго операнда используется регистр **ST(1)**.

Далее в строке 7 (**fld alpha**) начинается цикл построения точек окружности.

Первая команда в цикле **fld alpha** – загрузка в стек (его вершину) значения угла.

Следующая команда (строка 8 – **fcos**) вычисление косинуса (в качестве аргумента используется значение угла в вершине стека, а результат – значение вычисленного косинуса – помещается в ту же вершину стека)

Следующая команда (строка 9 – **fld r**) помещает радиус в вершину стека **ST(0)**, перемещая значение косинуса из вершины стека в следующую позицию **ST(1)**.

Команда в строке 10 (**fmul**) без указания операндов, поэтому производится умножение радиуса, расположенного в вершине стека на косинус, расположенный в следующей позиции. Результат остается в вершине стека.

В строке 11 в стек загружается коэффициент масштабирования по **X** (**fld rx**), а в строке 12 – умножение этого коэффициента на полученный ранее результат (сдвинутый при загрузке коэффициента в следующую позицию).

Строка 14 – загрузка координаты **x0** центра окружности в стек (со сдвигом результата предыдущего действия в следующую позицию).

Строка 15 **fadd** – сложение результата предыдущего умножения с координатой **x0** (по умолчанию суммируются вершина стека **ST(0)** и следующая позиция **ST(1)**, так как операнды в команде не указаны). В итоге мы получили в вершине стека полную координату **X** текущей точки окружности (но в плавающем формате).

Поэтому в следующей строке (16) выполняется команда **frndint** округление значения (в вершине стека) до целого.

После этого в строке 17 (**fistp word ptr xr**) мы с чистой совестью сохраняем целое значение координаты **X** в предназначенную для этого переменную **xr**. Определение **word ptr** необходимо для того, что в сопроцессоре значение представлено двойным словом, а нам в **xr** надо слово.

После этого выполняются вычисления, связанные с определением координаты **y** (строка 18 – **;y=round(ry*sin(alpha))**).

Все выполняется таким же образом, что и для координаты **x**, с тем же отличием, что и в предыдущей программе. Так как нам надо из координаты **y0** вычитать вычисленное значение, используется вспомогательная переменная **tmp** (в точности по тому же алгоритму, что и в предыдущей программе).

В итоге в строке 30 выполняется сохранение полученной целой экранной координаты (длиной в слово) в переменной **yr** (**fistp word ptr yr**).

Настало время строить полученную точку на экране. Для этого в строке 31 используется макрос **PutPix xr,yr,2** с тремя параметрами – координата **x** (**xr**), координата **y** (**yr**) и цвет (2). Сам макрос мы разберем ниже.

Затем в стек загружается шаг изменения угла **delta** (строка 32 – команда **fld delta**), после этого загружается значение угла **alpha** (строка 33 – команда **fld alpha**), не забывая при этом, что значение приращения при этом сдвигается в следующую позицию **ST(1)**.

Затем к строке 34 значение угла в вершине стека суммируется со значением приращения из **ST(1)**, и результат остается в вершине стека.

Следующей командой **fstp alpha** в строке 35 новое значение угла сохраняется в переменной **alpha**.

Последняя команда в строке 36 – замыкание цикла (**loop 11**). Вспомним, что делает команда **loop**: уменьшает значение **CX** на единицу, затем, если в **CX** не нуль, выполняется переход на указанный адрес (**11**), в противном случае команда пропускается. В итоге цикл построения точек окружности будет повторен столько раз, какое значение было в **CX** перед началом цикла (629).

Макрос построения точки на экране ничем не отличается от соответствующего кода предыдущей программы (строки 89-96). Отличие лишь в том, что там это – вызываемая процедура, поэтому в ней есть команда возврата (**ret**), здесь же это подставляемый макрос, поэтому команда возврата здесь не нужна.

Результат работы описанной программы показан на рисунке 4.

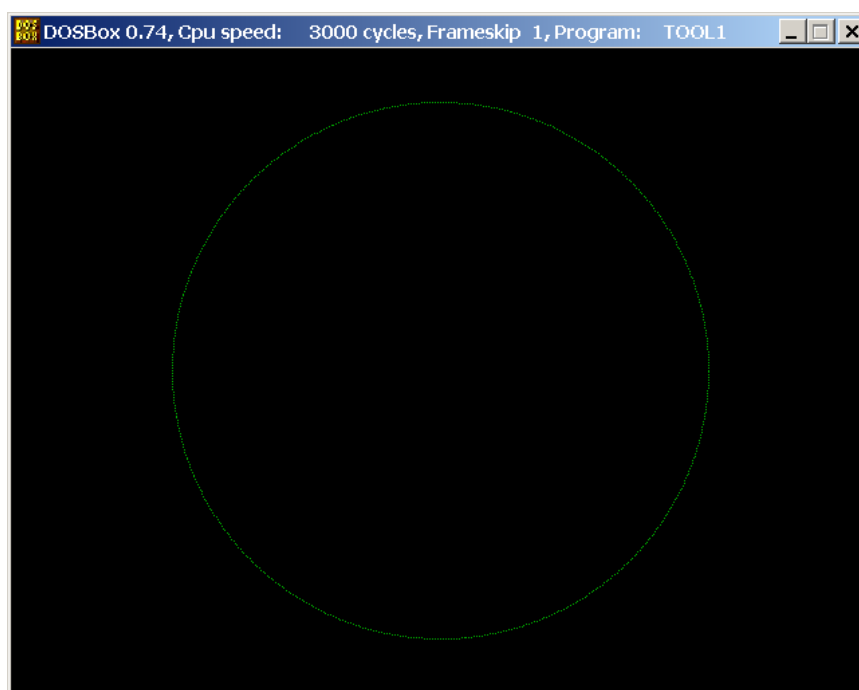


Рисунок 4 – Результат построения окружности с использованием команд сопроцессора

С этой программой можно проделать все те эксперименты, которые описаны для предыдущей программы. При этом не надо забывать, что соответствующие параметры здесь (координаты центра окружности, радиус, угол и его приращение – **delta**) имеют плавающий формат с десятичной точкой. Интересен эксперимент, которого нельзя было провести с предыдущей программой – изменить приращение угла и, соответственно, количество точек окружности. На рисунке 5 показан результат эксперимента, в котором **delta=6.0**, а количество точек (строка 6 в **mac.asm**) равна 120.

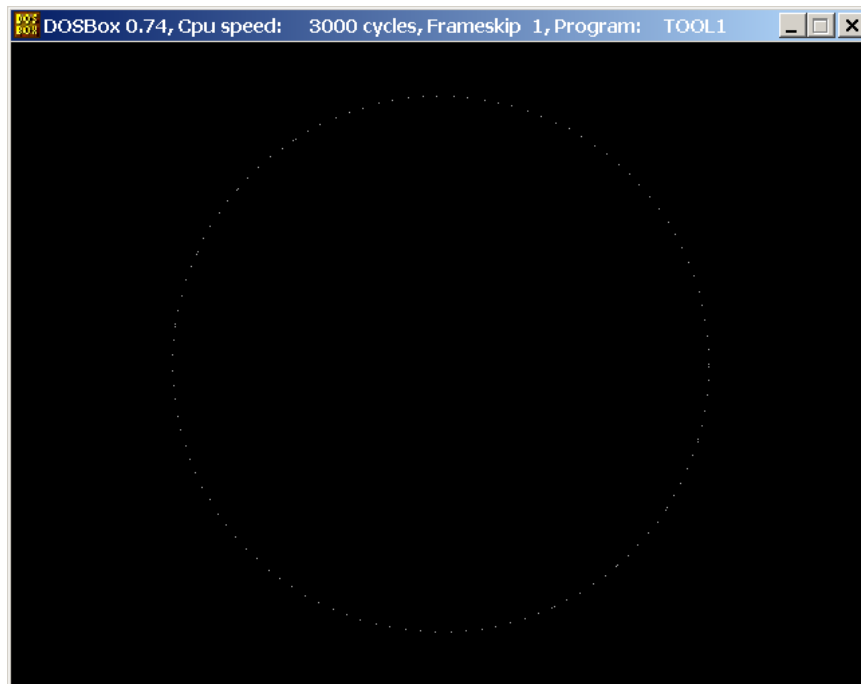


Рисунок 5 – Результат эксперимента, в котором **delta**=6.0, а количество точек (строка 6 в **mac.asm**) равно 120

В заключение приведу программу, решающую прикладную задачу: Проверка положения точки относительно окружности радиусом R и вписанного квадрата. Попробуйте разобрать (кому интересно) программу самостоятельно. Особое внимание стоит обратить на процедуры ввода значений (здесь используется уже известная Вам функция ввода в буфер) с последующей разборкой введенных символов. Интересна также процедура проверки положения точки относительно окружности и квадрата. Алгоритм проверки в синтаксисе языка Си показан ниже.

/*

За пределами окружности - 0

Между квадратом и окружностью - 1

Внутри квадрата - 2

*/

int PointCheck1(double x,double y,double R)

```
{
    return (x*x+y*y>R*R) ? 0 : ((fabs(x)+fabs(y)>R)?1:2);
}
```

int PointCheck2(double x,double y,double R)

```
{
    if(x*x+y*y>R*R)
        return 0;
    return (fabs(x)+fabs(y)>R)?1:2;
}
```

```

}
*****
int PointCheck3(double x,double y,double R)
{
    if(x*x+y*y>R*R)
        return 0;
    if(fabs(x)+fabs(y)>R)
        return 1;
    return 2;
}

```

Стоит также обратить внимание на построение осей и точек, а также на вывод текста в графическом режиме.

```

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 1, Program: TOOL1
Kira cyrillic/latin keyboard driver Version 3.0 10-27-92
Copyright (C) Serguey Fomin 1991-1992
Usage: KIRA [<command>]
<command>
/c - make CONFIGURATION
/u - unload driver.
For switch between RUS/LAT use RShift+LShift.
Kira installed.
C:\>\nc\nc.exe
The Norton Commander, Copyright (C) 1986, 88, 89, Peter Norton Computing, Inc.
C:\20_03_18>tool1.com
Введите радиус окружности в диапазоне от 0 до 200
150
Введите координату X в диапазоне от -220 до 220
90
Введите координату Y в диапазоне от -310 до 310
-90
Точка между квадратом и окружностью
Чтобы посмотреть, нажмите любую клавишу

```

Рисунок 6 – Результат работы программы в тексте

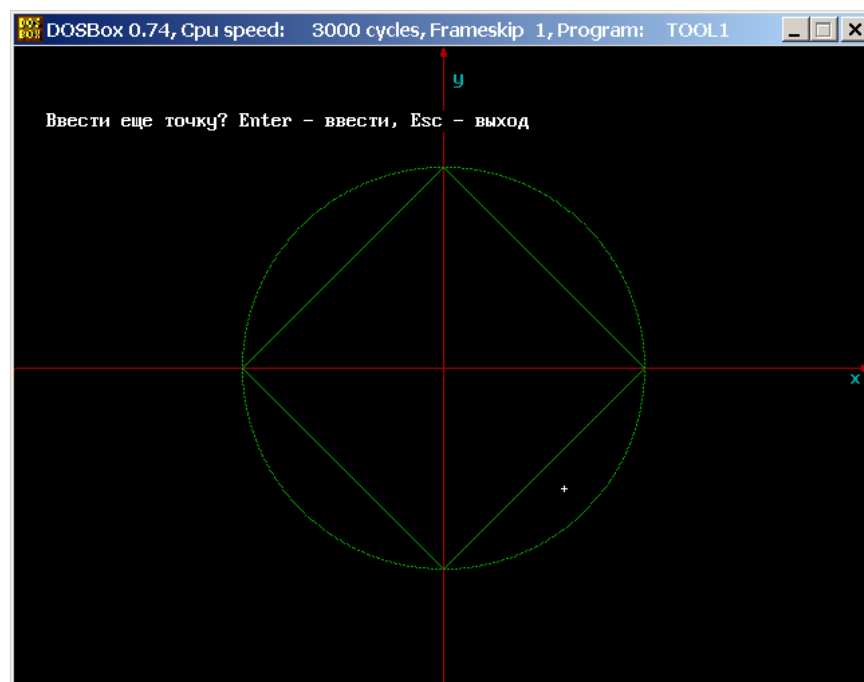


Рисунок 7 – Графическое представление работы прикладной программы

Программа со всеми модулями находится на сайте в каталоге 20_03_18.
Результаты работы программы показаны на рисунках 6 и 7.

Следует обратить внимание на то, что при запуске этой программы на виртуальной машине Oracle будут проблемы с русификацией текста в графическом режиме (см. рисунок 7). Во избежания неудобств в этом случае следует заменить текст "Ввести еще точку? Enter - ввести, Esc - выход\$" на тот же текст, набранный латиницей или на соответствующий перевод.

Желаю успехов. С вопросами обращайтесь по почте.

Основной модуль

```
;Проверка положения точки относительно окружности
радиусом R
;и вписанного квадрата. Режим VGA 640*480
; Диапазон заданич координат X от -320 до +320,
; Y от -240 до +240
mod_c equ 12h;12h графический 640*480 16 цветов
Assume CS: Code, DS: Code
Code SEGMENT
.486
org 100h
include mac1.asm ; Подключение макросов
Start: jmp begin
ry dd 1.0 ;масштабные коэффициенты
rx dd 1.0;0.87
x0 dd 320.0 ;середина по X и Y
y0 dd 240.0
x dw 0;X и Y относительно
y dw 0;начала координат
signdb 0

rad dd 200.0 ;Радиус окружности
radi dw 200
x0i dw 320 ;Координаты
y0i dw 240 ;начала координат
alpha dd 0.0 ;Угол
delta dd 0.01 ;величина изменения
Rsqr dd 0 ;Квадрат радиуса
Xsq dd 0 ;Квадрат X
Ysq dd 0 ;Квадрат Ytmp dd 0 ;temp
tmp dd 0 ;temp
xr dw 0 ;координаты выводимой точки
yr dw 0
begin:
    call radius ; Ввод радиуса окружности
```

```

b1: callinX      ; Ввод координаты X
    calliny     ; Ввод координаты Y
    call comp   ; Проверка попадания точки
    xor ah,ah
    int 16h
    call mode   ; Установка видеорежима
    callpaint   ; Построение осей, окружности и
квадрата
    call point  ; Построение введенной точки
    callmore   ; Приглашение на еще одну точку
    mov ah,0    ; Ожидание ввода с клавиатуры
    int 16h
    cmp ah,1;Esc
    jnz b1     ;Если Esc, на новый ввод точки
    int 20h    ; Выход из программы
modeprocnear;
    mov ah,0; Функция установки видеорежима
    mov al,mod_c; Значение режима
    int 10h
    ret
modeendp
compprocnear
;Проверка  $x^2 + y^2 > r^2$ 
    mov ax,radi ;Радиус
    movsx eax,ax ;Расширение до 32 разр.со знаком
    imuleax ;Квадрат радиуса
    mov Rsq,eax ;В Rsq
    mov ax,x;X
    movsx eax,ax
    imuleax ;X квадрат
    mov Xsq,eax ;В Xsq
    mov ax,y;Y
    movsx eax,ax
    imuleax ;Y квадрат
    mov Ysq,eax ;В Ysq
    mov eax,Xsq ;В EAX квадрат X
    add eax,Ysq ;+ квадрат Y
    cmp eax,Rsq ;Сравнение суммы с радиусом
    jc c2 ;Внутри окружности
    jz c1 ;На окружности
;Здесь вне окружности
;Проверка  $|X|+|Y|<radi$ 
    mov ah,9
    lea dx,ct0 ;"Вне окружности"
    int 21h

```

```

        jmp c5 ;Выход
c1: mov ah,9
    lea dx,ct1 ;"На окружности"
    int 21h
    jmp c5 ;Выход
c2: ;Здесь точка внутри окружности
    mov ax,x ;Координата X
    test ax,ax ;Проверка
    jns c21 ;Переход, если не минус
    neg ax ;Инверсия - модуль
c21: mov di,ax ;Сохранение модуля X в DI
    mov ax,y ;Координата Y
    test ax,ax ;Проверка
    jns c22 ;Переход, если не минус
    neg ax ;Инверсия
c22: add ax,di ;Сложение модуля Y и модуля X
    cmp ax,radi ;Сравнение суммы с радиусом
    jc c4 ;Внутри квадрата
    jz c3 ;На квадрате
    mov ah,9
    lea dx,ct2 ;"Между квадратом и окружностью"
    int 21h
    jmp c5 ;Выход
c3: mov ah,9
    lea dx,ct3 ;"На квадрате"
    int 21h
    jmp c5 ;Выход
c4: mov ah,9
    lea dx,ct4 ;"Внутри квадрата"
    int 21h
C5: mov ah,9
    lea dx,ct5 ;"Посмотреть"
    int 21h
    ret
ct0 db 10,13,"Точка вне окружности",10,13,'$'
ct1 db 10,13,"Точка на окружности",10,13,'$'
ct2 db 10,13,"Точка между квадратом и
окружностью",10,13,'$'
ct3 db 10,13,"Точка на квадрате",10,13,'$'
ct4 db 10,13,"Точка внутри квадрата",10,13,'$'
ct5 db 10,13,"Чтобы посмотреть, нажмите любую
клавишу",10,13,'$'
compendp
Paint proc near
    АхеХ

```

```

    AxeY
    PutCircle rad
    PutSquere
    ret
paint    endp
Moreproc near
    mov ah,2;Установить позицию курсора
    mov bh,0;Текущая видеостраница
    mov dh,3;3 строка
    mov dl,3;3 столбец
    int 10h
    mov ah,9
    lea dx,mm
    int 21h
    ret
mm    db  "Ввести еще точку? Enter - ввести, Esc -
        выход$"
Moreendp
include mac2.asm;Подключение процедур
;*****
praxprocnear    ;    Печать    четырехзначного
шестнадцатеричного числа из AX
    pushbx
    pushcx
    pushax
    and ax,0f000h
    mov cl,12
    shr ax,cl
    callprss
    pop ax
    pushax
    and ax,0f00h
    mov cl,8
    shr ax,cl
    callprss
    pop ax
    pushax
    and ax,0f0h
    mov cl,4
    shr ax,cl
    callprss
    pop ax
    pushax
    and ax,0fh
    callprss

```

```

        pop ax
        pop cx
        pop bx
        ret
praxendp
; Печать двухзначного шестнадцатеричного числа из AL
pralprocnear
        pusha
        pushax
        mov cl,4
        shr al,cl
        callprss
        pop ax
        callprss
        popa
        ret
pralendp
; Печать одного hex символа из мл. тетр. al
prssprocnear
        and al,0fh
        add al,30h
        cmp al,39h
        jle print
        add al,7
; Печать 1 ASCII символа
print    procnear
        pusha
        mov bx,0fh
        mov ah,0eh
        int 10h
        popa
        ret
print    endp
prssendp
; Печать пробела
space    procnear
        mov al,32 ; Код пробела
        jmp print ; Печать 1 ASCII символа
space    endp
Code     ends
        end Start

Подключаемый модуль mac1.asm
;*****
;Макрос вывода горизонтальной оси X
АхеX     macro      ;Без параметров

```

```

local    iter;Локальная метка
        pusha
PutSymG 78,15,03h,78h ;X - обозначение оси
        mov cx,640    ;Счетчик цикла (она же координ. X)
iter:
        PutPix  cx,240,4h
        loop iter    ;Цикл по CX
        PutPix  637,241,4h ;Стрелка
        PutPix  637,239,4h
        PutPix  636,241,4h
        PutPix  636,239,4h
        PutPix  635,241,4h
        PutPix  635,239,4h
        PutPix  634,241,4h
        PutPix  634,239,4h
        PutPix  633,241,4h
        PutPix  633,239,4h
        PutPix  632,242,4h
        PutPix  632,238,4h
        PutPix  633,242,4h
        PutPix  633,238,4h
        PutPix  632,241,4h
        PutPix  632,239,4h
        PutPix  634,242,4h
        PutPix  634,238,4h
        popa
        endm

;-----
---
;Макрос вывода окружности радиусом r на экран
PutCircle macro r
finit ;инициализация сопроцессора
;x=round(rx*cos(alpha))
mov  cx,629
l1: fld alpha ;Загрузка alpha в вершину стека
    fcos;Вычисление косинуса и помещение в вершину
стека
fld  r
fmul
    fld rx ;Загрузка rx в вершину стека
    fmul ;Умножение вершины стека на следующее в
стеке
        ;значение и оставление в вершине стека
frndint ;Округление вершины стека до целого
fld  x0 ;Загрузка середины X в стек

```

```

        fadd                ;Сложение вершины стека со следующим
значение
        fistp word ptr xr ;Сохранение вершины стека в xr
;y=round(ry*sin(alpha))
        fld alpha          ;Загрузка alpha в вершину стека
        fsin                ;Вычисление синуса и помещение в
вершину стека
        fld r
        fmul
        fld ry              ;Загрузка rx в вершину стека
        fmul                ;Умножение вершины стека на следующее в
стеке
        ;значение и оставление в вершине стека
        frndint            ;Округление вершины стека до целого
        fstp tmp           ;Сохранение вершины стека в tmp
        fld y0              ;Загрузка середины Y в стек
        fsub tmp            ;Вычитание из вершины стека tmp
(выч.знач.Y)
        fistp word ptr yr ;Сохранение вершины стека в yr
        PutPix xr,yr,2 ;выводим точку зеленым цветом
        fld delta ;вычисляем новое значение alpha
        fld alpha
        fadd
        fstp alpha
        loop 11 ;цикл по cx
        endm

```

```

;-----
---
```

```

;Макрос вывода квадрата
```

```

PutSquare macro
```

```

        pusha
mov si,0 ;X
mov cx,radi
l2: mov ax,radi
    sub ax,si ;y=r-x
    mov bp,ax
push cx
    mov cx,si
    mov al,2
    PutPix1
    neg bp
    PutPix1
    neg cx
    PutPix1
    neg bp

```

```

        PutPix1
pop cx
inc si
loop    12    ;цикл по cx
popa
endm

;-----
---
;Макрос вывода оси Y в середине экрана
AxeY    macro    ;Без параметра
local   iter;Локальная метка
        pusha
        mov cx,480    ;Счетчик цикла
iter:    mov dx,cx    ;Она же координата Y
        PutPix 320,dx,4 ;Цвет зеленый
        loop iter;Цикл по CX
        PutPix 319,2,4 ;Стрелка
        PutPix 321,2,4
        PutPix 319,3,4
        PutPix 321,3,4
        PutPix 319,4,4
        PutPix 321,4,4
        PutPix 318,5,4
        PutPix 322,5,4
        PutPix 318,6,4
        PutPix 322,6,4
        PutPix 318,7,4
        PutPix 322,7,4
        PutPix 319,6,4
        PutPix 321,6,4
        PutPix 319,7,4
        PutPix 321,7,4
        PutPix 319,5,4
        PutPix 321,5,4
        PutSymG 29h,01,03,79h ;Y
        popa
        endm

;-----
---
;Макрос вывода символа в граф режиме (sym - ASCII код символа)
;Параметры - колонка, строка, цвет, код символа
PutSymG macro x,y,col,sym
        pusha    ;Сохранение всех регистров в стеке
        mov ah,02h    ;Установка позиции курсора

```



```

    mov bh,0h      ;Текущая видеостраница
    mov dh,y;Строка
    mov dl,x;Колонка
    int 10h ;Выполнение функции
    mov ah,09h     ;Запись символа/аттр.в поз.курсора
    mov al,sym     ;Символ (ASCII-код)
    mov bh,0h      ;Текущая видеостраница
    mov bl,col     ;Цвет
    mov cx,01h     ;Количество выводимых символов
    int 10h ;Выполнение функции
    popa;Восстановление всех регистров из стека
endm
;-----
---
;Макрос вывода графической точки относительно осей
;Параметры - граф.коорд. X, Y, цвет
PutPix macro x,y,col
    pusha
    mov ah,0ch     ;Вывод граф.точки на экран
    mov al,col     ;Цвет
    mov bh,0h      ;Текущая видеостраница
    mov cx,x;Коорд.X (горизонталь)
    mov dx,y;Коорд. центра
    int 10h ;Выполнение функции
    popa
endm
;-----
---
;Процедура вывода графической точки относительно осей
;Параметры - CX-X, DX-Y, AL-цвет
PutPix1 macro
    pusha
    mov ah,0ch     ;Вывод граф.точки на экран
    mov bh,0h      ;Текущая видеостраница
    add cx,x0i
    mov dx,y0i
    sub dx,bp
    int 10h ;Выполнение функции
    popa
endm
Подключаемый модуль mac2.asm
;*****
***
; Ввод радиуса
radius    proc near

```

```

inR: mov ah,9
      lea dx,trad
      int 21h ;Вывод приглашения
      call in3 ;Ввод трехзначного числа
      cmp word ptr num,201
      jc  inP ;Пер.на вывод
      mov ah,9
      lea dx,erd ;Сообщ.о выходе из диапазона
      int 21h
      jmp inR
erd db 10,13,'Превышен диапазон',10,13,'$'
inP: mov ax,num
      mov radi,ax
      finit ;Перевод целого радиуса в плав.формат
      fld radi
      fstp rad
      ret
traddb 10,13,'Введите радиус окружности в диапазоне от
0 до 200'
      db 10,13,'$'
mrr db 10,13,'За пределами диапазона',10,13,'$'
buf db 4,0,0,0,0,0,0,0,'$'
num dw 0 ;Введенное число
radius endp
in3 proc near; Процедура ввода трехзначного числа
n0:  mov num,0 ;Обнуление вводимого числа
      mov ah,0ah ;Ввод в буфер 3 символов
      lea dx,buf
      int 21h
      lea bx,buf
      mov al,[bx+1] ;Количество введенных символов
      cmp al,0 ;Сравнение кол-ва ввода с 0
      jnz ni ;Есть цифры
;Здесь не введены цифры
      mov ah,9
      lea dx,tno
      int 21h ;"Не введено число, повторите ввод"
      jmp n0 ;На повтор ввода
tno db 10,13,"Не введено число, повторите
ввод",10,13,'$'
ni:  xor ah,ah
      mov si,0 ;Смещение для текущего номера цифры
      cmp al,3 ;Сравнение кол-ва ввода с 3
      jz n3 ;Обработка 3 цифр
      cmp al,2 ;Сравнение кол-ва ввода с 2

```

```

        jz  n2          ;Обработка 2 цифр
        jmp n1          ;Проц.обр.1 цифры
n3:     mov al,[bx+2+si];3 цифра
        call cnum ;Проверка на цифру от 0 до 9
        jc  errnum     ;"Введена не цифра"
        sub al,30h     ;Перевод в цифру
        mov cl,100
        mul cl         ;В AX 3 цифра *100
        add num,ax     ;Добавл.в num 3 цифра *100
        inc si
n2:     mov al,[bx+2+si];3 цифра
        call cnum
        jc  errnum
        sub al,30h     ;Перевод в цифру
        mov cl,10
        mul cl         ;В AX 2 цифра *10
        add num,ax     ;Добавл.в num 2 цифра *10
        inc si
n1:     mov al,[bx+2+si];3 цифра
        call cnum
        jc  errnum
        sub al,30h     ;Перевод в цифру
        xor ah,ah      ;В AX 1 цифра
        add num,ax     ;Добавл.в num 1 цифра
        ret
errnum:
        mov ah,9
        lea dx,trn
        int 21h ;"Введена не цифра, повторите ввод"
        jmp n0 ;На повтор ввода
trn db 10,13,"Введена не цифра, повторите
ввод",10,13,'$'
cnum:
        cmp al,30h ;
        jc  trnum ;al,30h Ниже цифры
        cmp al,3ah
        jnc trnum ;al>=3ah Выше цифры
        clc ;Сброс флага переноса
        ret
trnum:  stc ;Взведение флага переноса
        ret
in3 endp
;*****
; Ввод координаты X
inX proc near

```

```

inX1:  mov sign,0
        mov num,0
        mov ah,9
        lea dx,tX
        int 21h ;Вывод приглашения
        call in3s ;Ввод трехзначного числа
        cmp word ptr num,221
        jc  inX2;Пер.на выход
        mov ah,9
        lea dx,eXd  ;Сообщ.о выходе из диапазона
        int 21h
        jmp inX1
eXd db 10,13,'Превышен диапазон',10,13,'$'
inX2:  mov ax,num
        cmp sign,0
        jz  inX3;Переход, если плюс
        neg ax      ;Инверсия, если минус
inX3:  mov x,ax;Координата X
        ret
tX db 10,13,'Введите координату X в диапазоне от -220
до 220'
        db 10,13,'$'
;mrrdb 10,13,'За пределами диапазона',10,13,'$'
buf1db 0,0,0,'$'
inX endp
in3sprocnear; Проц.ввода трехзн.числа со знаком
ns0:mov ah,0;Ввод с ожиданием
        int 16h ;Ввод первого символа
call os
        cmp al,'-'    ;Проверка на минус
        jz  ns1      ;Если минус
        cmp al,13     ;Проверка на Enter
        jnz ns2      ;Не Enter - символ в буфер
;Enter - не введены цифры
nsn: mov ah,9
        lea dx,tno
        int 21h ;"Не введено число, повторите ввод"
        jmp ns0 ;На повтор ввода
; Здесь минус
ns1:mov sign,1    ; Флаг минуса
        mov ah,0
        int 16h    ;Ввод второго символа
call os
ns2:
        lea bx,buf1 ;Буфер для числа со знаком

```

```

    mov [bx],al ;Запись первой цифры в буфер
    mov ah,0
    int 16h      ;Ввод второго символа
call os
    cmp al,13    ;Проверка на Enter
    jz  nsi      ;Enter - на выход
    mov si,2;Введен второй символ
    mov [bx+1],al ;Запись второй цифры в буфер
    mov ah,0
    int 16h      ;Ввод третьего символа
call os
    cmp al,13    ;Проверка на Enter
    jz  nsi      ;Enter - на выход
    mov si,3;Введен второй символ
    mov [bx+2],al ;Запись второй цифры в буфер
;*****
nsi:xor ah,ah
    mov ax,si
    mov word ptr num,0
    mov si,0 ;Смещение для текущего номера цифры
    cmp al,3 ;Сравнение кол-ва ввода с 3
    jz  nss3    ;Обработка 3 цифр
    cmp al,2 ;Сравнение кол-ва ввода с 2
    jz  nss2    ;Обработка 2 цифр
    jmp nss1    ;Проц.обр.1 цифры
nss3:  mov al,[bx+si];3 цифра
    call cnum ;Проверка на цифру от 0 до 9
    jc  errnums ;"Введена не цифра"
    sub al,30h  ;Перевод в цифру
    mov cl,100
    mul cl      ;В AX 3 цифра *100
    add num,ax  ;Добавл.в num 3 цифра *100
inc si
nss2:  mov al,[bx+si] ;3 цифра
    call cnum
    jc  errnums
    sub al,30h  ;Перевод в цифру
    mov cl,10
    mul cl      ;В AX 2 цифра *10
    add num,ax  ;Добавл.в num 2 цифра *10
inc si
nss1:  mov al,[bx+si] ;3 цифра
    call cnum
    jc  errnums
    sub al,30h  ;Перевод в цифру

```

```

        xor ah,ah      ;В AX 1 цифра
        add num,ax     ;Добавл.в num 1 цифра
        ret
errnums:
        mov ah,9
        lea dx,trn
        int 21h ;"Введена не цифра, повторите ввод"
        jmp ns0 ;На повтор ввода
;trndb      10,13,"Введена не цифра, повторите
ввод",10,13,'$'
        ret
os: mov ah,0eh
        int 10h
        ret
in3sendp

;*****
; Ввод координаты Y
inY procnear
inY1: mov sign,0
        mov num,0
        mov ah,9
        lea dx,tY
        int 21h ;Вывод приглашения
        call in3s ;Ввод трехзначного числа
        cmp word ptr num,311
        jc inY2;Пер.на выход
        mov ah,9
        lea dx,eYd ;Сообщ.о выходе из диапазона
        int 21h
        jmp inY1
eYd db 10,13,'Превышен диапазон',10,13,'$'
inY2: mov ax,num
        cmp sign,0
        jz inY3;Переход, если плюс
        neg ax ;Инверсия, если минус
inY3: mov y,ax;Координата X
        ret
tY db 10,13,'Введите координату Y в диапазоне от -310
до 310'
        db 10,13,'$'
;mrrdb 10,13,'За пределами диапазона',10,13,'$'
;buf1 db 0,0,0,'$'
inY endp
;*****

```

;Процедура построение точки по коорд.X и Y

```
Point    procnear
    mov ax,x
    add ax,x0i
    mov xr,ax
    mov ax,y0i
    sub ax,y
    mov yr,ax
    sub xr,2
    mov cx,5
Po1: PutPix xr,yr,15
    inc xr
    loop Po1 ;Цикл по X
    sub xr,3
    sub yr,2
    mov cx,5
Po2: PutPix xr,yr,15
    inc yr
    loop Po2
    ret
Point    endp
```