ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Кафедра информационных систем и технологий

О.Л. Куляс, К.А. Никитин

Программирование на языке ASSEMBLER

Лабораторный практикум
по дисциплине
«ЭВМ и периферийные устройства»
(часть 2)

Самара
2016

УДК 004.43 (076)

К 907

Рекомендовано к изданию методическим советом ПГУТИ,
протокол № 8, от 14.04.2016 г.

Куляс, О.Л.
К 907 Программирование на языке ASSEMBLER: лабораторный практикум по дисциплине «ЭВМ и периферийные устройства» (часть 2) / О.Л. Куляс, К.А. Никитин. – Самара: ПГУТИ, 2016. – 78 с.

Лабораторный практикум предназначен для бакалавров направления 09.03.01 – «Информатика и вычислительная техника», изучающих курс «ЭВМ и периферийные устройства». Двухсеместровый цикл лабораторных работ включает 12 работ (7 работ в 1-й части и 5 работ во 2-й), которые позволяют освоить основы программирования на языке ASSEMBLER. Каждая лабораторная работа содержит достаточный теоретический материал, поэтапно вводящий студентов в мир программирования на языке Ассемблера, сведения и задания, необходимые для практического выполнения работы, список литературы, рекомендуемой для дополнительного изучения, а также контрольные вопросы для проверки усвоения изученного.

Лабораторный практикум можно использовать не только студентам, указанного направления подготовки, но и всем желающим самостоятельно овладеть основами программирования на языке ASSEMBLER.

©, Куляс О.Л., 2016

Оглавление

Лабораторная работа №8
Графические операции в текстовом режиме дисплея

1 Цель работы

Получение практических навыков использования системных прерываний BIOS и DOS для создания графических изображений на экране дисплея.

2 Теоретический материал

2.1 Вывод изображений на экран

Для вывода изображений на экран используются видеоадаптеры (видеокарты), которые подключаются к системе через разъемы расширения и формируют сигналы, управляющие работой монитора. Существуют два принципиально разных режима работы видеоадаптеров – текстовый и графический.

В тестовом режиме изображение состоит из литер расширенного набора ASCII, формируемых знакогенератором. При этом из квазиграфических символов, которые входят в набор ASCII, возможно построение примитивных рисунков.

В графическом режиме изображение строится попиксельно, что позволяет формировать на экране сложные изображения и надписи разных размеров и конфигураций.

Видеопамять компьютера (оперативная память для хранения изображений) физически расположена на плате видеоадаптера. В современных видеоадаптерах ее емкость может достигать сотен Мбайт. Однако логически эта память является частью адресного пространства процессора. Диапазон адресов **A0000h…AFFFFh**, размером **64К**, отводится под графический видеобуфер, а диапазон **B8000h…BFFFFh**, размером **32К**, отводится под текстовый видеобуфер видеопамяти. Обращаясь по этим адресам, можно записывать и выводить на экран графическую или текстовую информацию.

И в графическом, и в текстовом режимах видеоадаптеры организованы так, что определенному адресу в видеопамяти соответствует определенное место на экране монитора. Аппаратура видеоадаптера периодически (с частотой кадров) считывает содержимое видеопамяти и отображает его на экране, формируя изображение.

В графическом режиме каждой точке экрана соответствует своя ячейка видеопамяти, в которой храниться информация о цвете пиксела. Адресуемым элементом является пиксел, позиция которого определяется номерами столбца и строки в системе координат экрана.

В текстовом режиме ячейка видеопамяти хранит информацию о

символе, который занимает на экране знакоместо определенного формата. Информация о символе включает **код символа** (1 байт) и **атрибуты символа** (1 байт). К атрибутам символа относят цвет символа, цвет фона, мигание. Знакоместо состоит из матрицы точек, количество которых может быть разным: 8x8, 9x14, 9x16. Изображение символа считывается из знакогенератора, который представляет собой запоминающее устройство (ОЗУ или ПЗУ). Систему координат экрана в текстовом режиме можно представит рис. 8.1, на котором показан экран размером 80x25 символов. В этом случае адресуемым элементом является знакоместо, которое определяется номерами столбца и строки.

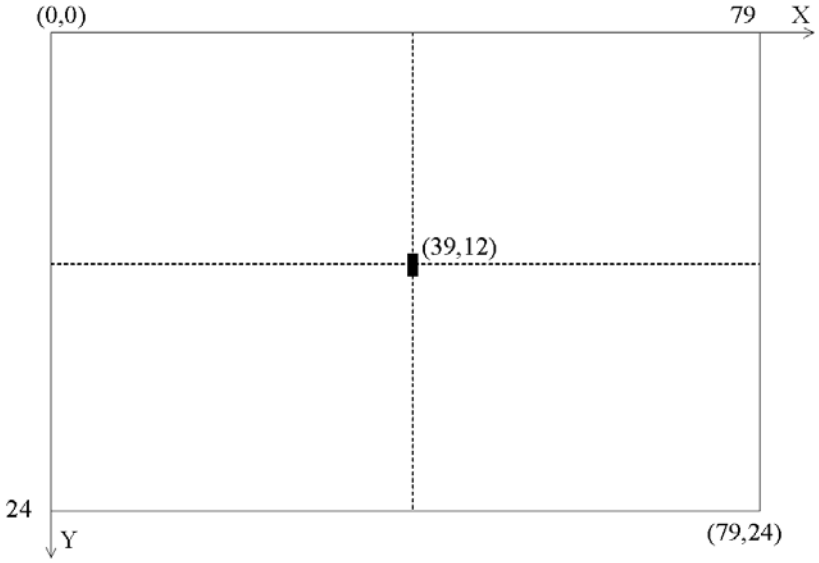

Рис. 8.1 – Система координат экрана в текстовом режиме

2.2 Видеоадаптеры и режимы их работы

Приведем список основных видеоадаптеров в хронологическом порядке их появления.

– **MDA** (Monochrome Display Adapter) – монохромный адаптер, применяемый в первых PC. Режим – текстовый, 4 цвета.

– **HGC** (Hercules Graphic Controller) – монохромный адаптер фирмы Hercules. Режим – текстовый, графический, 4 цвета.

– **CGA (**Color Graphic Adapter) – цветной графический адаптер. Режимы – текстовый (16 цветов), графический (4 цвета), разрешение до 320x200.

– **EGA** (Enhanced Graphics Adapter) – расширенный графический адаптер. Режим – текстовый, графический, 16 цветов, разрешение до 640x320. Поддерживает режимы MDA, CGA.

– **VGA** (Video Graphics Array) – видео графический адаптер. Режим текстовый, графический, 16 цветов (256 цветов с разрешением 320x200), разрешение до 640x480. Поддерживает режимы CGA, EGA.

– **SVGA** (Super VGA) – режим текстовый, графический. Число цветов от 16 до 32 млн. Разрешение до 1280x1024. Поддерживают режимы CGA, EGA, VGA.

Как видно из приведенного списка, большинство видеоадаптеров могут работать как в текстовых, так и в графических режимах. Существует несколько стандартных режимов с номерами 0…13h (таблица 8.1), поддерживаемых практически всеми современными адаптерами.

Таблица 8.1

Режимы работы видеоадаптеров

Режим работы	Тип режима	Количество цветов	Разрешение	Размер символов
0, 1h	Текстовый цветной	16	40x25	8x8
2, 3h	Текстовый цветной	16	80x25	8x8
4, 5h	Графический цветной	4	320x200	
6h	Графический цветной	2	340x200	
7h	Текстовый монохромный	2	80x25	9x14
0Dh	Графический цветной	16	320x200	
0Eh	Графический цветной	16	640x200	
0Fh	Графический монохромный	2	640x350	
10h	Графический цветной	16	340x350	
11h	Графический цветной	2	640x480	
12h	Графический цветной	16	640x480	
13h	Графический цветной	256	320x200	

Видеоадаптеры SVGA могут также работать в режимах, имеющих улучшенные характеристики. Для этих режимов разработан стандарт **VESA** (Video Electronics Standards Association), который определяет режимы работы с номерами 100h…11Ah [7.2].

2.3 Способы вывода информации на экран

Операционная система предоставляет несколько способов вывода информации на экран:

– с использованием **средств DOS** (группа функций ввода-вывода из диапазона 01h…0Ch прерывания **INT 21h**). Поддерживается

только текстовый монохромный режим вывода. (Этот способ использовался в лабораторных работах №1…№7 для вывода сообщений на экран);

– с использованием **средств BIOS** с помощью прерывания **INT 10h.** Позволяет реализовать все возможности видеоадаптеров в текстовом и графическом режимах. Используется в реальном режиме работы МП;

– прямое программирование видеопамяти.

2.4 Вывод информации на экран средствами BIOS

Программы BIOS находятся в ПЗУ BOIS и вызываются посредством прерываний. Причем за каждым устройством компьютера закреплено свое прерывание. Работа с видеоадаптером производится с помощью прерывания **INT 10h**. Для обращения к нужной функции видеоадаптера необходимо выполнить следующую последовательность действий:

– загрузить в регистр **AH** номер нужной функции BIOS видеоадаптера;

– загрузить остальные регистры МП в соответствии с вызываемой функцией (см. таблицу 8.2);

– выполнить прерывание **INT 10h.**

Основные функции прерывания **INT 10h**, предназначенные для работы с видеоадаптерами приведены в таблице 8.2, а их подробное описание в [7.2].

Таблица 8.2

Функции вызова прерывания **INT 10h**

Функция (AH)	Назначение	Действия
00h	Выбор режима работы (если в AL бит D7=1, то при установке режима видеопамять не очищается)	AH:= 00h AL:= № режима
02h	Установить позицию курсора	AH:=02h BH:= номер страницы видеопамяти (для графики 0) DH:= номер строки DL:= номер столбца
03h	Получить позицию курсора	AH:=03h (см. [7.2])
05h	Установить активную видеостраницу	AH:= 05h AL:= № нужной страницы

Продолжение таблицы 8.2

Функция (AH)	Назначение	Действия
06h	Инициализировать или прокрутить окно вверх	AH:= 06h AL:= число строк прокрутки CH:= номер строки верхнего левого угла CL:= номер столбца верхнего левого угла DH:= номер строки нижнего правого угла DL:= номер столбца нижнего правого угла BH:= атрибут для пустых строк
07h	Инициализировать или прокрутить окно вниз	AH:= 07h, остальное то же, что для 06h
08h	Прочитать символ и атрибут в позиции курсора	AH:= 08h BH:= номер активной страницы AL:= символ, считанный с позиции курсора AH:= атрибут символа
09h	Вывести символ и атрибут в позицию курсора	AH:= 09h AL:= выводимый символ BL:= атрибут выводимого символа (в графике нет) BH:= номер активной страницы CX:= число выводимых символов
0Ah	Вывести символ в позицию курсора	AH:= 09h BL:= атрибут цвета для графики остальное то же что и в 09h
0Bh	Установка цветовой палитры (для режимов 4, 5, 6)	AH:=0Bh, остальное см. [7.2]

Продолжение таблицы 8.2

Функция (AH)	Назначение	Действия
0Ch	Вывести пиксел	AH:=0Ch AL:= номер цвета пиксела BH:= номер страницы видеопамяти CX:= координата X пиксела DX:= координата Y пиксела
0Dh	Чтение пиксела	AH:=0Dh BH:= номер страницы видеопамяти CX:= координата X пиксела DX:= координата Y пиксела Возвращается номер цвета пиксела в AL
0Eh	Вывести символ в режиме телетайпа (символ выводится в позицию курсора, курсор сдвигается)	AH:=0Eh AL:=ASCII код символа BL:= цвет символа (в граф. режимах) (Если в регистре AL бит D7=1, то пикселы символа накладываются на содержимое экрана с использованием операции XOR)
0Fh	Определить текущий режим работы видеоадаптера	AH:=0Fh AL:= режим адаптера AH:= число символов в строке BH:= номер активной страницы
10h	Управление регистрами палитры	AH:= 10h, остальное см. [7.2]

Алгоритм создания изображения на экране в текстовом режиме работы состоит из нескольких шагов, которые реализуются функциями BIOS из таблицы 8.2:

– задание режима экрана;

– очистка экрана с заданием цвета фона и воспроизводимых символов;

– установка курсора в нужную позицию;

– вывод символа;

– вывод горизонтальной цепочки любых символов (в том числе и текстовой строки).

В начале программы следует задать режим экрана. Для этого используется **функция BIOS** с номером **0** (см. таблицу 8.2). Номер функции нужно занести в **регистр AH**, а в **регистр AL** следует занести номер режима видеоадаптера (см. таблицу 8.1). Текстовому цветному режиму с разрешением **80x25** символов, который предполагается использовать, соответствует **номер 3.**

Таким образом, режим экрана задаётся следующей последовательностью команд:

```
MOV AH, 00h  ;выбор функции задания режима экрана
MOV AL, 03h  ;режим видеоадаптера 80x25, 16 цветов
INT 10h      ;вызов функции BIOS
```

Данный набор команд вместе с заданием режима обеспечивает очистку экрана с установкой по умолчанию черного фона и белых символов. Для задания иного цвета фона и изображаемых символов используется байт-атрибут.

Таблица 8.3

Назначение битов атрибута символов

Номер бита	7	6	5	4	3	2	1	0
Атрибут	L	R	G	B	I	R	G	B
		цвет фона				цвет символа		

Отдельные биты атрибута кодируют следующие признаки:

Бит 7 (L) при установке в 1 обеспечивает эффект мигания символа.

Бит 3 (I) при установке в 1 обеспечивает повышенную яркость символа.

Биты 6, 5, 4 и 2, 1, 0 определяют цвет фона и символов соответственно. При этом **R** – красный, **G** – зеленый, **B** – синий цвет. Значения 1 обеспечивают наличие соответствующей компоненты цвета, 0 – его отсутствие. Можно заметить, что комбинация 000 соответствует черному, а 111 – белому цвету, R+G дает желтый цвет, R+B – пурпурный, G+B – голубой.

Примеры байтов – атрибутов:

00000000b (0h) – черный по черному – неотображаемый символ (для пароля),

00000111b (07h) – белый по черному нормальной яркости,

10001111b (8Fh) – ярко белый по черному с миганием;

00101110b (2Eh) – ярко желтые символы на зеленом фоне;

10001100b (8Ch) – мигающие ярко красные символы на черном фоне.

Очистка с заданием цвета фона и символов всего экрана или части строк экрана производится так называемой **прокруткой вверх или вниз**, которая реализуется с помощью **INT 10h** с **AH=06h** (вверх) или **AH=07h** (вниз), что для задания цвета равнозначно. При этом на экране создается прямоугольная область текстового окна, которая прокручивается вниз или вверх на одну или более строк.

```
MOV AH, 06h          ;Функция прокрутки вверх
MOV  AL, 00h         ;Очистка всего экрана
MOV  BH,00100100b    ;Атрибут пробела – зеленый фон,
                     ;красный символ
MOV  CL, 0           ;Верхняя левая позиция: столбец
MOV CH, 0            ; в CL (X=0)строка в CH (Y=0)
MOV  DL, 79          ;Нижняя правая позиция:  столбец в DL
                     ;(X=79),
MOV DH, 24           ;строка в DH (Y=24)
INT    10h
```

Для задания цвета фона и символов только на нескольких последовательных строках необходимо в **AL** загрузить количество строк, в **CX** – координаты начала первой строки из этой группы, а в **DX** – координаты конца последней строки.

Процедура установки курсора в нужную позицию реализуется командой **INT 10h** с **AH=02h**. В регистр **BH** заносится номер экранной страницы (обычно 00h), а в регистр **DX** – координаты курсора: в **DH** – номер строки, в **DL** – номер столбца (элемента по строке).

```
MOV AH, 02h          ;Функция перемещения курсора.
MOV BH, 00h          ;Страница 0.
MOV DH, 05h          ;Строка 5.
MOV DL,0Ch           ;Столбец 12.
INT 10h
```

Вывод отдельного символа с заданным атрибутом реализуется **INT 10h** с **AH=09h**. В регистр **AL** заносится код символа, в **BL –** байт-атрибутов символа, обычно такой же, какой был определен при очистке этого участка экрана (если он не был по умолчанию черно-белым). Если задать другой атрибут, то он будет определять цвет фона и символов только в пределах прямоугольника – знакоместа символа.

```
MOV AH, 09h          ;Функция вывода символа.
MOV AL, 2Ah          ;Символ ‘:’.
```

```
MOV BH, 00h      ;Страница 0.
MOV BL, 0Fh      ;Ярко белый по чёрному.
MOV CX,01h       ;Один символ.
INT 10h
```

Последовательность одинаковых символов в одной строке (горизонтальную цепочку) можно получить, если в предыдущем примере задать содержимое **CX** отличным от единицы.

Для получения вертикальной цепочки одинаковых символов (например, вертикальная сторона рамки) следует организовать цикл, включающий процедуры установки курсора и вывода одиночного символа с меняющейся координатой ***Y*** в каждом цикле. Поскольку регистр **CX** содержит количество выводимых символов, для организации циклов нельзя использовать команду **LOOP**, которая по умолчанию обращается к **CX** как к счетчику циклов. Для проверки условия окончания циклов необходимо использовать какие-либо другие признаки. Пример реализации этого положения можно найти в примере программы вывода графика **GRAFIC**.

Системное прерывание, вызывающее функцию DOS **INT 21h** с **номером 09 (AH=09h),** уже неоднократно использовалось в предыдущих лабораторных работах. Оно позволяет вывести на экран сразу целую символьную строку (например, текст), заданную как переменная в сегменте данных. При этом в регистр **DX** должен быть загружен адрес (смещение) этой переменной. Пусть в сегменте данных имеется переменная:

```
CITY DB ‘SAMARA’, 0Dh, 0Ah, ‘$’
;0Dh – код перевода строки,
;0Ah – код установки курсора в начало строки (возврат каретки),
;’$’ – ограничитель области вывода.
```

Вывод слова SAMARA на экран реализуется следующим фрагментом программы:

```
LEA DX, CITY
MOV AH, 09h
INT 21h
```

Цвет фона и выводимой надписи соответствует атрибуту, установленному при очистке этого участка экрана.

Прерывание **INT 16h** с **AH=00h** обеспечивает ожидание ввода символа с клавиатуры без его отображения на экране и может быть

полезно для организации некоторой паузы между фрагментами программы. Никаких других параметров не требуется.

```
MOV AH, 00h  ; пауза до
INT 16h      ; нажатия клавиши
```

Если после какого-то фрагмента программы нужна пауза, следует вывести текст «Нажмите любую клавишу», затем вставить две строки программы ожидания. Тогда после нажатия клавиши программа будет продолжена.

3 Подготовка к работе

3.1. Изучить методические указания и рекомендованную литературу.

3.2. Подготовить ответы на контрольные вопросы.

4 Задание на выполнение работы

4.1 Разработать программу **ELOCHKA** для изображения на экране прямоугольной рамки размером в 15 строк по вертикали и 40 элементов по горизонтали, расположение которой выбирается в соответствии с последней цифрой номера студенческого билета:

1 – в левой верхней части экрана;
2 – по центру в верхней части экрана;
3 – в правой верхней части экрана;
4 – по центру в левой части экрана;
5 – по центру в правой части экрана;
6 – в центральной части экрана;
7 – в левой нижней части экрана;
8 – по центру в нижней части экрана;
9 – в правой нижней части экрана.

Если номер студенческого билета заканчивается на нуль, используйте предпоследнюю цифру.

Для четных номеров рамка изображается с помощью одинарных горизонтальных и вертикальных линий, для нечетных – залитыми точками.

Коды символов: горизонтальная черточка – 5Fh,
вертикальная черточка – 7Ch,
залитая точка – 07h

Внутри рамки сформировать изображение елочки, примерно как показано слева, используя символ «звездочка» (код 2Ah).

На вершине елочки поместить мигающий яркий символ «солнышко» (код 0Fh).

Вне рамки на свободном пространстве экрана в трех строках разместить поздравительную надпись, причем каждая строка текста должна иметь другой цвет.

4.2 Проанализировать пример программы GRAFIC, приведенной ниже и реализующей вывод графика функции, последовательность значений которой определена в сегменте данных в виде массива MAS. Добавить в исходный текст недостающие комментарии.

4.3 Ассемблированием и компоновкой получить исполняемый модуль программы GRAFIC и убедиться в его работоспособности.

4.4 Разработать на основе отлаженной программы новую программу **GRAFIK_m** производящую вывод на экран графика функции, заданной в сегменте данных массивом значений **FUNC**. Число выводимых на график элементов массива следует выбрать исходя из условия: **N = 15 + n**, где **n** – последняя цифра номера зачетной книжки. (*При этом значения функции следует **программно** пронормировать путем деления на некоторый масштабный коэффициент с тем, чтобы максимальное значение не выходило за диапазон допустимых значений координаты Y на экране).*

Кроме того, в новой программе следует:

а) перенести начало осей координат в центр экрана;

б) обеспечить, чтобы значению X = 0 соответствовало значение центрального элемента массива;

в) сформировать стрелочки на концах осей координат и обозначить их символами **X** и **Y**;

г) задать другой цвет осей и графика.

```
TITLE GRAFIC
;Программа построения графика функции в текстовом режиме экрана
;Входные параметры:
;массив значений элементов графика MAS
.MODEL SMALL
.DATA
   Mas   DB  0,1,5,8,9,8,5,1,0,-1,-5,-8,-9,-5,-1
   Func DW 450, -350, 0, 250, 375, 400, 420, 360, 250, 200, 150
          DW 325, 300, 285, 200, 0,  –200, -275, -250, -150, -100
          DW  -50, 0, 50, 100, 150, 200
.STACK 256 (?)
.CODE
Start:
      mov AX, @DATA
      mov DS, AX
```

```
;--------- Задание режима экрана с очисткой  ---------------------------------
        mov AH, 0
        mov AL, 3               ;Режим 80x25, 16 цветов.
        int 10h
;-------- Построение вертикальной оси координат ---------------------------
        mov BH, 0               ;Используем страницу видеопамяти 0.
        mov DL, 5               ;Координаты начальной точки X=5,
        mov DH, 1               ; Y =1.
 met1: mov AH, 02h              ;Выбираем функцию установки курсора.
        int 10h                 ;Установка курсора.
        mov CX, 1               ;Выводим по одному символу
        mov AL, 7Ch             ;символ вертикальной черточки.
        mov BL, 00001111b       ;Атрибут: ярко белый по черному фону.
        mov AH, 09h             ;Выбираем функцию вывода символа и
                                ;атрибута в позицию курсора.
        int 10h                 ;Вывод символа.
        inc DH                  ;Переход к координате Y+1.
        cmp DH, 24              ;Сравнение с нижней позицией.
        jb met1                 ;Если ниже, повторить цикл вывода символа.
;-------------Построение горизонтальной оси координат -------------------
        mov BH, 0
        mov DL, 5               ;Координаты начальной позиции X=5,
        mov DH, 12              ;Y=12.
        mov AH, 02h
        int 10h
        mov CX, 50              ;Длина цепочки символов.
        mov AL,5Fh              ;Символ горизонтальной черточки
        mov BL, 00001111b       ;ярко-белый по черному фону.
        mov AH, 09h
        int 10h
;--------------Вывод точек графика-----------------------------------------
        lea SI, Mas             ;Загрузка адреса массива значений.
        mov DI, 15              ;Установка счетчика циклов.
        mov CX, 1               ;Вывод по одному символу.
        mov BH, 0               ;Используем страницу видеопамяти 0.
        mov DL, 5               ;Координаты первой точки X=5,
 met2: mov DH, 12               ;                          Y=12.
        sub DH, [SI]            ;Вычисление Y(i) в системе координат
                                ;графика.
        mov AH, 02h             ;Установка курсора в вычисленную
        int 10h                 ;позицию
```

```
        mov AL, 2Ah;          ;Символ “*”.
        mov BL, 00001100b;    ;Ярко красный по черному фону.
        mov AH, 09h
        int 10h               ;Вывод символа.
        add DL, 3             ;Следующая координата по X.
        inc SI                ;Индекс следующего элемента массива.
        dec DI                ;Изменение счетчика циклов.
        jnz met2              ;Повторить цикл, если ZF=0.
;----------------Завершение программы----------------------------------
        mov AX, 4C00h
        int 21h
END Start
```

5 Требования к отчёту

Отчёт должен содержать:

– титульный лист с указанием названия ВУЗа, кафедры, номера и темы лабораторной работы, а также фамилии И.О. студента, подготовившего отчёт;

– цель работы;

– вариант задания;

– листинги программы **ELOCHKA** с комментариями;

– результат работы программы **ELOCHKA**;

– листинг программы **GRAFIC_m** с комментариями;

– результат работы программы **GRAFIC_m**.

6 Контрольные вопросы

6.1 Поясните принципы вывода текстовой и графической информации на экран.

6.2 Что такое видеопамять, видеоадаптер и какова их роль в формировании изображений на экране?

6.3 Что представляет собой система координат экрана в текстовом режиме?

6.4 Назовите основные типы видеоадаптеров.

6.5 Назовите и поясните существующие способы вывода информации на экран.

6.6 Что такое системные прерывания? Какие системные прерывания используются для вывода информации на экран дисплея?

6.7 Назовите основные графические операции, реализованные с помощью BIOS?

6.8 Назовите шаги алгоритма, позволяющего построить изображение в текстовом режиме.

6.9 Чем отличаются между собой разные режимы экрана?
6.10 Какие операции необходимо произвести для вывода некоторого символа в определенном месте экрана?
6.11 Чем отличаются процедуры изображения на экране горизонтальной и вертикальной цепочки символов?
6.12 Как задается цвет экрана и цвет символов?
6.13 Какую роль выполняет байт атрибутов символа?
6.14 Каковы основные шаги алгоритма построения графика функции, значения которой хранятся в памяти (на примере программы GRAF)?
6.15 Как можно ввести паузу в программу?

7 Рекомендуемая литература

7.1 Юров, В. И. Assembler [Текст]: учеб. пособие для вузов / В. И. Юров. 2-е изд. – СПб.: Питер, 2007. с. 447…509.
7.2 Рудаков, П.И. Язык Ассемблера: уроки программирования [Текст] / П. И. Рудаков, К. Г. Финогенов. – М.: ДИАЛОГ-МИФИ, 2001. – с. 45…47, 255…280, 563…593

Лабораторная работа №9
Программирование математического сопроцессора и графических операций вывода на экран

1 Цель работы

Изучение принципов работы сопроцессора и методов его программирования средствами Ассемблера. Изучение графического режима вывода на экран и методов его программирования. Работа с процедурами и системными прерываниями.

2 Теоретический материал

2.1 Программная модель сопроцессора

Программная модель сопроцессора представляет собой набор регистров, доступных программисту (см. рис. 9.1). В модели можно выделить три группы регистров:

а) ***группа арифметических регистров***, которая состоит из 8-и арифметических регистров общего назначения **R0…R7**. Арифметические регистры представляют собой 80-и разрядные (10-и байтовые) регистры, организованные в стек. Эти регистры служат для хранения операндов и результатов вычислений. Они являются основой операционного устройства сопроцессора.

б) ***группа служебных регистров***:

– **регистр состояния сопроцессора SWR** (**S**tatus **W**ord **R**egister) отражает информацию о текущем состоянии сопроцессора и содержит поля, позволяющие определить, какой регистр является текущей вершиной стека сопроцессора, какие исключения возникли после выполнения последней команды, каковы особенности выполнения последней команды (некий аналог регистра флагов основного процессора) и т. д.;

– **управляющий регистр сопроцессора CWR** (**C**ontrol **W**ord **R**egister) управляет режимами работы сопроцессора. С помощью полей в этом регистре можно регулировать точность выполнения численных вычислений, управлять округлением, маскировать исключения;

– **регистр слова тегов**[1] **TWR** (**T**ags **W**ord **R**egister) используется для контроля за состоянием каждого из регистров **R0…R7** (команды сопроцессора используют этот регистр, например, для того, чтобы определить возможность записи значений в указанные регистры).

[1] Tag – признак

в) ***группа регистров указателей***:
- **указатель данных DPR** (**D**ata **P**oint **R**egister);
- **указатель команд IPR** (**I**nstruction **P**oint **R**egister).

Оба регистра предназначены для запоминания информации об адресе команды, вызвавшей исключительную ситуацию, и адресе ее операнда. Эти указатели используются при обработке исключительных ситуаций (но не для всех команд).

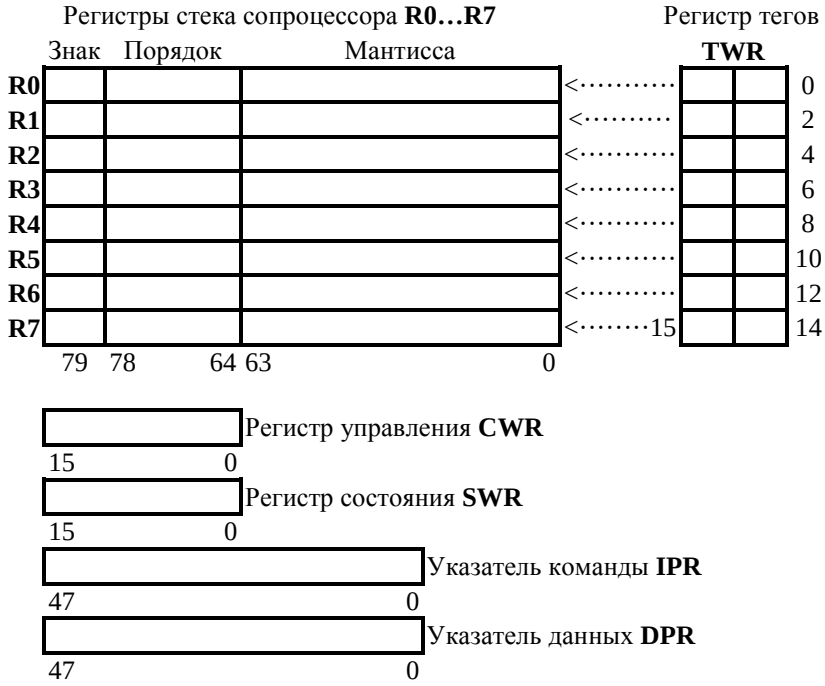

Рис. 9.1 – Программная модель сопроцессора

Рассмотрим общую логику работы сопроцессора. Регистровый стек сопроцессора организован по принципу кольца. Это означает, что среди всех регистров, составляющих стек, нет такого, который является вершиной стека. Все регистры стека с функциональной точки зрения абсолютно равноправны. Вершина стека является плавающей. Номер регистра, который является вершиной стека, хранится в 3-х разрядном поле **TOP** регистра состояний (**SWR**), т. е. поле **TOP** является указателем стека. Команды сопроцессора не используют физические номера регистров **R0…R7**, а используют их логические имена, кото-

рые обозначаются **st(0)…st(7)**. Нумерация регистров всегда начинается от вершины стека. При загрузке данных в регистры содержимое указателя стека уменьшается на 1 и показывает физический номер регистра стека в который производится загрузка. При извлечении данных из регистров стека, и их записи в память, содержимое указателя стека автоматически увеличивается на 1.

Сопроцессор может работать с данными разного формата. Особенность состоит в том, что эти данные могут являться вещественными числами. В этом случае они должны объявляться как двойные слова, т. е. соответствовать стандарту **IEEE754**. Вне зависимости от формата данные, которые передаются на обработку в сопроцессор, преобразуются в расширенный формат двойной точности – 80 битовое представление с плавающей точкой. Именно таков формат регистров стека сопроцессора. После завершения обработки результат может быть записан в память, а перед этим преобразован в необходимый формат.

2.2 Система команд сопроцессора

Система команд сопроцессора насчитывает восемь десятков команд, которые разделяют на пять групп по функциональным признакам:

- команды передачи данных;
- команды сравнения данных;
- арифметические команды;
- трансцендентные команды;
- команды управления.

Мнемоника команд сопроцессора описывается некоторыми правилами, которые отражают особенности его работы. Эти правила сводятся к перечисленным ниже позициям.

а) Все мнемонические обозначения начинаются с символа **F** (**F**loat).

б) Вторая буква мнемонического обозначения определяет тип операнда в памяти, с которым работает команда:

I – целое двоичное число;

B – целое десятичное число;

отсутствие буквы – вещественное число.

в) Последняя буква **P** в мнемоническом обозначении команды означает, что последним действием команды обязательно является извлечение операнда из стека (**POP**).

г) Последняя или предпоследняя буква **R** (**R**eversed) в мнемоническом обозначении команды означает реверсивное следование операндов при выполнении команд вычитания и деления, так как для них важен порядок следования операндов.

Группа команд передачи данных предназначена для организации обмена между регистрами стека, вершиной стека сопроцессора и ячейками оперативной памяти. С их помощью осуществляются все перемещения значений операндов в сопроцессор и из него. По этой причине для каждого из трех типов данных, с которыми может работать сопроцессор, существует своя подгруппа команд передачи данных. **Главной функцией всех команд загрузки данных в сопроцессор является преобразование данных к единому представлению в виде вещественного числа расширенного формата. Это же касается и обратной операции – сохранения в памяти данных из сопроцессора.**

Команды передачи данных можно разделить на следующие группы:

а) команды передачи данных в вещественном формате:

– **FLD источник** – загрузка вещественного числа из области памяти на вершину стека сопроцессора;

– **FST приемник** – сохранение вещественного числа из вершины стека сопроцессора в память. Сохранение числа в памяти не сопровождается выталкиванием его из стека (отсутствует символ **P**), то есть текущая вершина стека сопроцессора не меняется (поле **TOP** не меняется);

– **FSTP приемник** – сохранение вещественного числа из вершины стека сопроцессора в память с выталкиванием из стека (в мнемонике присутствует символ **P**). Команда изменяет поле **TOP**, увеличивая его на единицу. Вследствие этого вершиной стека становится следующий больший по своему физическому номеру регистр стека сопроцессора.

б) команды передачи данных в целочисленном формате:

– **FILD источник** – загрузка целого числа из памяти на вершину стека сопроцессора;

– **FIST приемник** – сохранение целого числа из вершины стека сопроцессора в память. Сохранение целого числа в памяти не сопровождается выталкиванием его из стека, то есть текущая вершина стека сопроцессора не изменяется;

– **FISTP приемник** – сохранение целого числа из вершины стека в память. Последним действием команды является выталкивание числа из стека с одновременным преобразованием его в целое значение.

в) команды передачи данных в десятичном формате:

– **FBLD источник** – загрузка десятичного числа из памяти на вершину стека сопроцессора;

– **FBSTP приемник** – сохранение десятичного числа из вершины стека сопроцессора в области памяти. Значение выталкивается из стека после преобразования его в формат десятичного числа. Заметьте, что для десятичных чисел нет команды сохранения значения в памяти без выталкивания из стека.

К группе команд передачи данных можно отнести также команду обмена вершины регистрового стека **st(0)** с любым другим регистром стека сопроцессора **st(i): FXCH st(i).**

Действие команд загрузки **FLD**, **FILD** и **FBLD** можно сравнить с командой **PUSH** основного процессора. Аналогично ей (**PUSH** уменьшает значение в регистре **SP**) команды загрузки сопроцессора перед сохранением значения в регистровом стеке сопроцессора вычитают из содержимого поля **TOP** регистра состояния **SWR** единицу. Это означает, что вершиной стека становится регистр с физическим номером на единицу меньше. При этом возможно переполнение стека. Так как стек сопроцессора состоит из ограниченного числа регистров, то в него может быть записано максимум восемь значений. Из-за кольцевой организации стека девятое записываемое значение затирает первое. Программа должна иметь возможность обработать такую ситуацию. По этой причине почти все команды, помещающие свой операнд в стек сопроцессора, после уменьшения значения поля **TOP** проверяет регистр-кандидат на новую вершину стека на предмет его занятости. Для анализа этой и подобных ситуаций используется регистр **TWR**, содержащий слово тегов. Наличие регистра тегов в архитектуре сопроцессора позволяет освободить программиста от разработки сложной процедуры распознавания содержимого регистров сопроцессора и дает самому сопроцессору возможность фиксировать определенные ситуации, например попытку чтения из пустого регистра или запись в непустой регистр. Возникновение таких ситуаций фиксируется в регистре состояния **SWR**, предназначенном для сохранения общей информации о сопроцессоре. Используя специальные команды сопроцессора, можно извлечь из него или, напротив, записать в него информацию.

Арифметические команды реализуют четыре основные арифметические операции – сложение, вычитание, умножение и деление. Имеется также несколько дополнительных команд, предназначенных для повышения эффективности использования основных арифметических команд. С точки зрения типов операндов арифметические команды сопроцессора можно разделить на команды, работающие с вещественными и целыми числами.

а) **Целочисленные арифметические команды** – предназначены для работы на тех участках вычислительных алгоритмов, где в ка-

честве исходных данных используются целые числа в памяти в формате слово и короткое слово, имеющие размерность 16 и 32 бита.

– **FIADD источник** – команда складывает значения **st(0)** и целочисленного источника. Результат сложения запоминается в регистре стека сопроцессора **st(0).**

– **FISUB источник** – команда вычитает значение целочисленного источника из **st(0)**. Результат вычитания запоминается в регистре стека сопроцессора **st(0).**

– **FIMUL источник** – команда умножает значение целочисленного источника на содержимое **st(0)**. Результат умножения запоминается в регистре стека сопроцессора **st(0).**

– **FIDIV источник** – команда делит содержимое **st(0)** на значение целочисленного источника. Результат деления запоминается в регистре стека сопроцессора **st(0).**

Для команд, реализующих арифметические действия деления и вычитания, важен порядок расположения операндов. По этой причине система команд сопроцессора содержит соответствующие реверсивные команды, повышающие удобство программирования вычислительных алгоритмов. Чтобы отличить эти команды от обычных команд деления и вычитания, их мнемокоды оканчиваются символом **R.**

– **FISUBR источник** – команда вычитает значение **st(0)** из целочисленного источника, который расположен в памяти. Результат вычитания запоминается в регистре стека сопроцессора **st(0).**

– **FIDIVR источник** – команда делит значение целочисленного источника на содержимое **st(0)**. Результат деления запоминается в регистре стека сопроцессора **st(0).**

б) **Вещественные арифметические команды**

Схема расположения операндов вещественных команд традиционна для команд сопроцессора. Один из операндов располагается в вершине стека сопроцессора – регистре **st(0)**, куда после выполнения команды записывается и результат, а второй операнд может быть расположен либо в памяти, либо в другом регистре стека сопроцессора. Допустимыми типами операндов в памяти являются все перечисленные ранее вещественные форматы за исключением расширенного.

В отличие от целочисленных арифметических команд, вещественные арифметические команды допускают большее разнообразие в сочетании местоположения операндов и самих команд для выполнения конкретного арифметического действия. Так, например, можно выделить три возможных варианта команды сложения. В дополнение к этим трем вариантам существует еще одна команда сложения, производящая дополнительное действие – удаление значения из стека.

– **FADD** – команда складывает значения в **st(0)** и **st(1)**. Результат сложения запоминается в регистре стека сопроцессора **st(0).**

– **FADD источник** – команда складывает значения **st(0)** и источника, представляющего адрес ячейки памяти. Результат сложения запоминается в регистре стека сопроцессора **st(0).**

– **FADD st(i), st** – команда складывает значение в регистре стека сопроцессора **st(i)** со значением в вершине стека **st(0)**. Результат сложения запоминается в регистре **st(i).**

– **FADDP st(i), st** – команда производит сложение вещественных операндов аналогично команде **FADD st(i), st**, однако последним действием команды является выталкивание значения из вершины стека сопроцессора **st(0)**. Результат сложения остается в регистре **st(i-l).**

Для выполнения операции вычитания также имеется большой набор команд.

– **FSUB** – команда вычитает значение в **st(1)** из значения в **st(0)**. Результат вычитания запоминается в регистре стека сопроцессора **st(0).**

– **FSUB источник** – команда вычитает значение источника из значения в **st(0)**. Источник представляет адрес ячейки памяти, содержащей допустимое вещественное число. Результат сложения запоминается в регистре стека сопроцессора **st(0).**

– **FSUB st(i), st** – команда вычитает значение в вершине стека **st(0)** из значения в регистре стека сопроцессора **st(i)**. Результат вычитания запоминается в регистре стека сопроцессора **st(i).**

– **FSUBP st(i), st** – команда вычитает вещественные операнды аналогично команде **FSUB st(i), st**. Последним действием команды является выталкивание значения из вершины стека сопроцессора **st(0).** Результат вычитания остается в регистре **st(i-l).**

Для удобства группа команд вычитания вещественных чисел дополнена командами реверсивного вычитания.

– **FSUBR st(i), st** – команда вычитает значение в вершине стека **st(0)** из значения в регистре стека сопроцессора **st(i)**. Результат вычитания запоминается в вершине стека сопроцессора – регистре **st(0).**

– **FSUBRP st(i), st** – команда производит вычитание подобно команде **FSUBR st(i), st**. Последним действием команды является выталкивание значения из вершины стека сопроцессора **st(0)**. Результат вычитания остается в регистре **st(i-l).**

Особенностью команд умножения вещественных операндов, является то, что они располагаются исключительно в стеке сопроцессора.

– **FMUL** – команда не имеет операндов. Умножает значения в **st(0)** на содержимое в **st(1)**. Результат умножения запоминается в регистре стека сопроцессора **st(0)**.

– **FMUL st(i)** – команда умножает значение в **st(0)** на содержимое регистра стека **st(i)**. Результат умножения запоминается в регистре стека сопроцессора **st(0)**.

– **FMUL st(i), st** – команда умножает значения в **st(0)** на содержимое произвольного регистра стека **st(i)**. Результат умножения запоминается в регистре стека сопроцессора **st(i)**.

– **FMULP st(i), st** – команда производит умножение подобно команде **FMUL st(i), st**. Последним действием команды является выталкивание значения из вершины стека сопроцессора **st(0)**. Результат умножения остается в регистре **st(i-l)**.

В командах, реализующих деление вещественных данных, операнды также располагаются в стеке сопроцессора.

– **FDIV** – команда (без операндов) делит содержимого регистра **st(0)** на значение регистра сопроцессора **st(1)**. Результат деления запоминается в регистре стека сопроцессора **st(0)**.

– **FDIV st(i)** – команда делит содержимое регистра **st(0)** на содержимое регистра сопроцессора **st(i)**. Результат деления запоминается в регистре стека сопроцессора **st(0)**.

– **FDIV st(i), st** – команда производит деление аналогично команде **FDIV st(i)**, но результат деления запоминается в регистре стека сопроцессора **st(i)**.

– **FDIVP st(i), st** – команда производит деление аналогично команде **FDIV st(i), st**. Последним действием команды является выталкивание значения из вершины стека сопроцессора **st(0)**. Результат деления остается в регистре **st(i-l)**.

Для реализации деления в сопроцессоре также предусмотрены две реверсивные команды, отличительным признаком которых является наличие символа **R** в качестве последнего или предпоследнего символа мнемокода:

– **FDIVRst(i), st** – команда делит содержимое регистра **st(i)** на содержимое вершины регистра сопроцессора **st(0)**. Результат деления запоминается в регистре стека сопроцессора **st(0)**.

– **FDIVRP st(i), st** – команда делит содержимое регистра **st(i)** на содержимое вершины регистра сопроцессора **st(0)**. Результат деления запоминается в регистре стека сопроцессора **st(i)**, после чего производится выталкивание содержимого **st(0)** из стека. Результат деления остается в регистре **st(i-l)**.

2.3 Вычисление полиномов

Для вычисления полиномов n – й степени вида

$$Y = a_1 X^n + a_2 X^{n-1} + ... + a_n X + a_{n+1}$$

удобно использовать формулу Горнера

$$Y = \left(...\left(\left(a_1 X + a_2\right) X + a_3\right) X + ... + a_n\right) X + a_{n+1}.$$

Если выражение, стоящее внутри скобок, обозначить через $\boldsymbol{Y_i}$, тогда значение выражения в следующих скобках $\boldsymbol{Y_{i+1}}$ можно вычислить, используя рекуррентную формулу $Y_{i+1} = Y_i X + a_{i+1}$. Значение полинома ***Y*** получается после повторения этого процесса в цикле ***n*** раз. Начальное значение $\boldsymbol{Y_1}$ должно быть равно $\mathbf{a_1}$, а цикл следует начинать с ***i*=2**.

2.4 Алгоритм вычисления полинома с вещественным аргументом

Исходными данными являются:

– коэффициенты полинома $\boldsymbol{a_1}$, $\boldsymbol{a_2}$, … $\boldsymbol{a_5}$, которые хранятся в памяти в виде одномерного массива **MAS_A**, размером WORD;

– **N** – переменная для хранения порядка полинома;

– **X** – переменная для хранения аргумента полинома;

– **Y** – переменная для хранения результата вычисления полинома размером DOUBLE WORD;

– **M** – переменная для хранения масштабного коэффициента, необходимого для построения графика;

– **Number** – переменная в которой хранится число отсчетов для построения графика;

– **Step** – переменная для хранения шага изменения аргумента X.

Регистры микропроцессора распределены следующим образом:

– регистр **SI** используется для хранения индексов элементов массива **MAS_A**;

– регистр **CX** используется как счетчик циклов, который нужно повторить **N** раз (для полинома 4-й степени число повторений равно 4).

В соответствии с формулой Горнера, основными операциями при вычислении полинома являются умножение и сложение, которые выполняются в цикле. Поскольку аргумент X является вещественным числом, т.е. может принимать не только целые, но и дробные значения, вычисления следует выполнять в сопроцессоре.

Общую идею алгоритма можно сформулировать так:

– устанавливается графический режим вывода на экран 320x200x256;

– на экран выводятся оси системы координат;

– вычисляется значение полинома для очередного отсчета аргумента X;

– вычисленное значение полинома нормируется путем деления на масштабный коэффициент;

– полученное значение округляется до целого числа и пересылается в переменную **Y**;

– значение **Y** выводится на график функции;

– после того, как выведены все 200 рассчитанных значений, происходит установка текстового режима работы экрана и выход в DOS.

В соответствии с изложенным, алгоритм вычислений можно разбить на несколько функциональных участков (см. рис. 9.2 и 9.3):

– блоки 1, 2 – подготовительные операции;

– блоки 3…5, 18 – инициализация графического режима и построение осей графика;

– блоки 6…17, 19…22 – вычисление значений полинома;

– блоки 23…25 – завершающие операции.

2.5 Отладка программ, работающих с сопроцессором

Контроль и отладка программ, с использованием команд сопроцессора производится с помощью отладчика **Turbo Debugger** (TD), который используется для отладки программ основного процессора. Поскольку программы обычно используют команды и основного процессора и сопроцессора желательно контролировать выполнение тех и других. Результаты выполнения команд основного процессора можно наблюдать в окне CPU, а для контроля выполнения команд сопроцессора следует открыть окно **Numeric processor**, выбрав соответствующий пункт в меню **View**. В этом окне выводится информация о регистрах сопроцессора и о его флагах. Положение этого окна следует отрегулировать так, чтобы оно не закрывало информацию о выполняемой программе и о CPU (см. рис. 9.4).

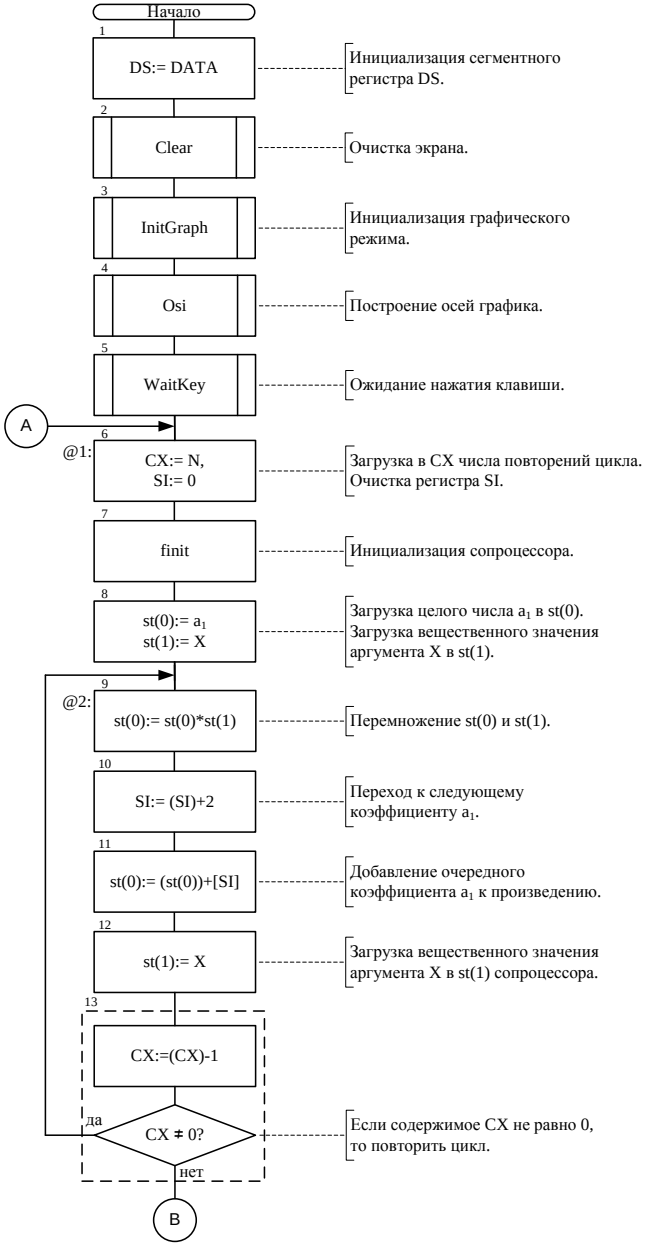

Рис. 9.2 – Алгоритм вычисления полинома

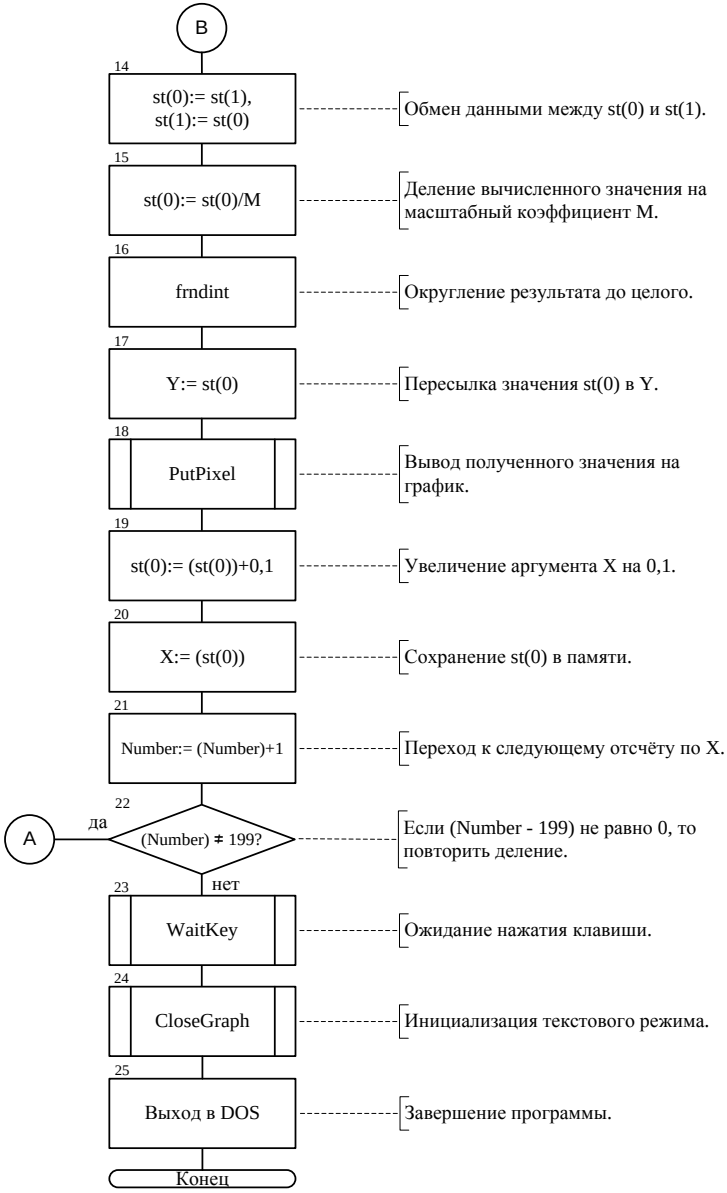

Рис. 9.3 – Продолжение алгоритма вычисления полинома

Отладку программы, как обычно, удобно проводить в пошаговом режиме с помощью клавиши **F8**, контролируя при этом состояние регистров и флагов. Отладчик позволяет пошагово возвращаться назад используя клавиши **Alt+F4**. Для того чтобы вернуться к началу программы и выполнить ее еще раз, следует использовать **Ctrl+F2.**

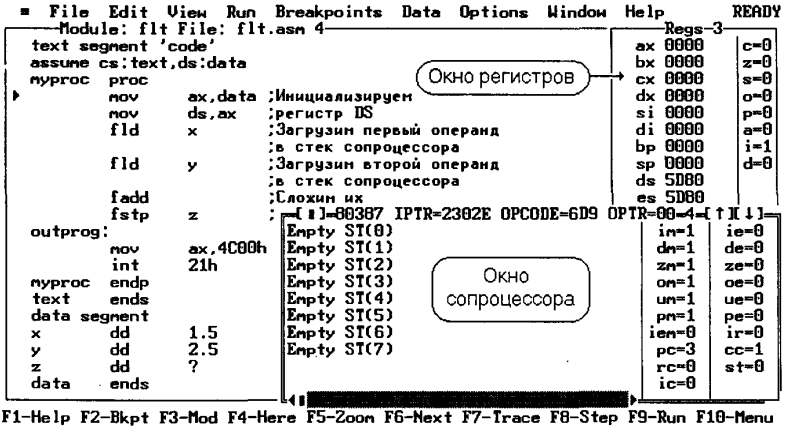

Рис. 9.4 – Окно отладчика Turbo Debugger

В процессе отладки можно, не выходя из отладчика, изменять содержимое регистров, памяти и состояние флагов. Для этого следует сделать нужное окно активным, а затем нажатием **Alt+F10**, либо правой кнопкой мыши вызвать локальное меню. Например, для изменения содержимого регистров сопроцессора, нужно в окне сопроцессора выбрать нужный регистр и вызвать локальное меню, которое включает три пункта: **Zero**, **Empty**, **Change** (см. рис. 9.5).

Если выбрать **Zero**, то в выбранный регистр **st** запишется 0. Если выбрать **Empty**, то регистр освободится и станет доступным для записи. Для того чтобы записать в этот регистр новое число следует выбрать пункт **Change**. Это приведет к появлению нового окна, в поле которого можно будет ввести требуемое значение и подтвердить его, нажав **OK**. После этого содержимое выбранного регистра изменится (см. рис. 9.5). Аналогично можно изменять содержимое памяти и регистров CPU. Для изменения флага необходимо выделить его, вызвать локальное меню, в котором будет только один пункт – **Toggle**, и подтвердить решение об изменении флага.

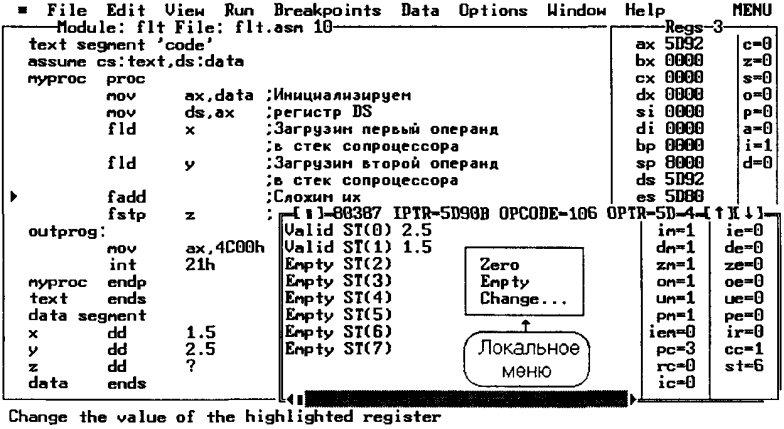

Рис. 9.5 – Локальное меню окна сопроцессора

3 Подготовка к работе

3.1. Изучить методические указания и рекомендованную литературу.

3.2. Подготовить ответы на контрольные вопросы.

4 Задание на выполнение работы

4.1 Используя, описанный в разделе 2, алгоритм вычисления полинома $Y = a_1X^n + a_2X^{n-1} + \ldots + a_nX + a_{n+1}$ разобрать исходный текст программы **POLINOM**. Программа вычисляет и строит график вычисленных значений полинома для аргумента **X**, изменяющегося в диапазоне от **–10** до **+10** с шагом **0,1**. Создать и отладить исполняемый модуль программы **POLINOM**, выполнив этапы ассемблирования и компоновки. Добавить в исходный модуль программы недостающие комментарии.

4.2 Модифицировать исходный модуль программы **POLINOM** для своего варианта задания (таблица 9.1). Создать и отладить исполняемый модуль программы **POLY_X** (**X** – номер варианта), выполнив этапы ассемблирования и компоновки.

Таблица 9.1

Исходные данные

N = 4, X = -10…+10, шаг 0,1											
№ варианта	a_1	a_2	a_3	a_4	a_5	№ варианта	a_1	a_2	a_3	a_4	a_5
1	-4	9	0	-5	0	9	5	-9	-9	9	4
2	0	-9	8	6	-9	10	3	6	6	7	6
3	2	0	9	6	6	11	-2	8	7	4	9
4	3	2	-9	2	8	12	0	-7	-9	3	-3
5	-3	9	5	1	2	13	-1	6	-4	-9	8
6	-2	8	-5	0	1	14	2	4	3	8	0
7	-3	7	6	-9	9	15	4	0	6	-8	-1
8	3	-7	6	9	-8						

```
TITLE POLINOM
;Программа вычисления и построения графика функции вида
; Y=a1X^n + a2X^n-1 + …+ anX +a^n+1
;Входные параметры:
;  коэффициенты полинома ai в массиве MAS_A
;  порядок полинома N
;  аргумент полинома X
;  масштабный коэффициент для вывода графика M
;  шаг изменения аргумента STEP
;  номер отсчёта Number для значений X
.MODEL SMALL                ;Модель памяти ближнего типа.
.STACK 256                  ;Отвести под стек 256 байт.
.486                        ;Используем расширенную систему команд.
.DATA                       ;Открыть сегмент данных.
  Mas_A DW -3, 3, -6, 9, -20  ;Коэффициенты полинома.
  N DW 4                    ;Порядок полинома равен 4.
  X DD -10                  ;Начальное значение аргумента X.
  M DW 180                  ;Масштабный коэффициент.
  Step DD 0.1               ;Шаг изменения аргумента X.
  Number DW 0               ;Номер отсчёта для значений X.
  Y DD (?)                  ;Результат вычисления полинома.
;-------------------------------------------------------------------------------
.CODE                       ;Открыть сегмент кодов.
;========== Инициализация графического режима =============
  InitGraph PROC
     pusha
```

```
    mov AH, 0                ;Установить режим экрана
    mov AL, 13h              ;320x200x256
    int 10h                  ;средствами BIOS.
    popa
    ret
  InitGraph ENDP
;============== Закрытие графического режима ==============
  CloseGraph PROC
    mov AX, 3                ;Установить текстовый
    int 10h                  ;режим 25x80 средствами BIOS.
    ret
  CloseGraph ENDP
;================ Очистка экрана ========================
  Clear PROC
    pusha
    mov CX, 64000            ;Число пикселов экрана.
    mov AX, 0A000h           ;Адрес графической видеопамяти
    mov ES, AX               ;в ES.
    mov AL, 00010100b
    xor DI, DI
    cld
    rep stosb
    popa
    ret
  Clear ENDP
;================ Ожидание нажатия клавиши ===============
  WaitKey PROC
    pusha
    mov AH, 01h
    int 21h
    popa
    ret
  WaitKey ENDP
;================ Рисование осей ========================
  Osi PROC
      pusha
      mov CX, 10             ;Начало горизонтальной
      mov DX, 5              ;оси.
      mov AL, 00000110b      ;Цвет оси желтый
  o1: mov AH, 12             ;Вывод точки.
      int 10h                ;Вызов BIOS.
```

```
      inc CX                ;Построить
      cmp CX, 300           ;300
      jne o1                ;точек.
;-------------------------------------------------------------------------------
      mov CX, 160           ;Начало вертикальной
      mov DX, 0             ;оси.
      mov AL, 00000110b     ;Цвет оси желтый.
 o2: mov AH, 12             ;Вывод точки
      int 10h
      inc DX
      cmp DX, 200
      jne o2
      popa
      ret
 Osi ENDP
;================ Вывод точки на экран ===================
 PutPixel PROC
    ;ecx, edx – координаты точки
    pusha
    mov AL,0000010b         ;Цвет пиксела.
    mov EDX, 5              ;Номер строки
    sub EDX, Y              ;вывода.
    nop
    mov CX, 60              ;Номер столбца
    add CX, Number          ;вывода.
    nop
    mov AH,12               ;Вывести пиксел
    int 10h                 ;на экран
    nop
    popa
    ret
 PutPixel ENDP
;-------------------------------------------------------------------------------
 Start:  mov AX, @Data
         mov DS, AX
         call Clear
         call InitGraph
         call Osi
         call WaitKey       ;Пауза.
 @2:     mov CX, N          ;Загрузить счетчик циклов.
         xor SI,SI
```

```
        finit                   ;Инициализировать сопроцессор.
        fild Mas_A[SI]          ;Загрузить целое a1 в st(0).
        fld X                   ;Загрузить X в st(1).
@1:     fmul                    ;Перемножть st(0):=(st(0))*(st(1)).
        inc SI                  ;Перейти к следующему
        inc SI                  ;ai.
        fiadd Mas_A[SI]         ;Добавить очередное ai к произведению.
        fld X                   ;Загрузить X в st(1).
        loop @1                 ;Перейти на метку, если CX не 0.
        fxch st(1)              ;Обменять st(0) и st(1).
        fidiv M                 ;Разделить на масштабный коэффициент.
        frndint                 ;Округлить до целого.
        fistp Y                 ;Переслать (st(0)) в Y.
        call PutPixel           ;Вывести полученное значение на график.
        fadd Step               ;Увеличить на шаг st(0):=(st(0))+0.1.
        fstp X                  ;Сохранить st(0)+0.1 в память Z.
        inc Number              ;Перейти к следующему отсчету по X.
        cmp Number,199          ;Повторить еще 199 раз.
        jnz @2
        call WaitKey
        call CloseGraph         ;Закрыть графический режим
        mov AX, 4C00h           ;и выйти
        int 21h                 ;в DOS.
END Start
```

5 Требования к отчёту

Отчёт должен содержать:

– титульный лист с указанием названия ВУЗа, кафедры, номера и темы лабораторной работы, а также фамилии И.О. студента, подготовившего отчёт;

– цель работы;

– вариант задания;

– листинги программы **POLY_X** с комментариями;

– результат работы программы **POLY_X**.

6 Контрольные вопросы

6.1 Какие регистры входят в программную модель сопроцессора?

6.2 Как организован стек сопроцессора? Физические и логические номера регистров.

6.3 Какой из регистров и каким образом используется в качестве указателя стека?

6.4 Назначение и формат регистра тегов TWR.
6.5 С данными каких форматов может работать сопроцессор?
6.6 Назначение и формат регистра состояний SWR.
6.7 Назначение управляющего регистра сопроцессора CWR.
6.8 Формат одинарной точности – короткое вещественное.
6.9 Формат двойной точности – длинное вещественное.
6.10 Расширенный формат – расширенное вещественное.
6.11 Десятичные числа – BCD-формат.
6.12 Назначение и операнды команд передачи данных сопроцессора.
6.13 Назначение и операнды арифметических команд сопроцессора.
6.14 Что такое исключения сопроцессора?
6.15 Поясните алгоритм вычисления полинома в формате вещественных чисел?
6.16 Поясните шаги процедуры построения осей графика?
6.17 Поясните шаги процедуры задания фона экрана?
6.18 Поясните шаги процедуры вывода пиксела на экран?
6.19 Какие операции позволяют включить графический режим работы?
6.20 Какие операции позволяют ввести в программу паузу до нажатия клавиши?

7 Рекомендуемая литература

7.1 Юров, В. И. Assembler [Текст]: учеб. пособие для вузов / В. И. Юров. 2-е изд. – СПб.: Питер, 2007. с. 447…509.
7.2 Рудаков, П.И. Язык Ассемблера: уроки программирования [Текст] / П. И. Рудаков, К. Г. Финогенов. – М.: ДИАЛОГ-МИФИ, 2001. – с. 45…47, 255…280, 563…593.

Лабораторная работа №10
Программирование математического сопроцессора и графических операций вывода на экран

1 Цель работы

Изучение принципов работы сопроцессора и методов его программирования средствами Ассемблера. Изучение графического режима вывода на экран и методов его программирования. Работа с процедурами и системными прерываниями.

2 Теоретический материал

2.1 Вывод изображений на экран в графическом режиме

Для вывода изображений на экран монитора используются видеоадаптеры (видеокарты), которые подключаются к системе через разъемы расширения и формируют сигналы, управляющие работой монитора. Существуют два принципиально разных режима работы видеоадаптеров – текстовый[1] и графический. И в графическом, и в текстовом режимах видеоадаптеры организованы так, что определенному адресу в видеопамяти соответствует определенное место на экране монитора. Аппаратура видеоадаптера периодически (с частотой кадров) считывает содержимое видеопамяти и отображает его на экране, формируя изображение.

Видеопамять компьютера (оперативная память для хранения изображений) физически расположена на плате видеоадаптера. В современных видеоадаптерах ее емкость может достигать сотен Мбайт. Однако логически эта память является частью адресного пространства процессора. Диапазон адресов **A0000h…AFFFFh**, размером **64К**, отводится под графический видеобуфер, а диапазон **B8000h…BFFFFh**, размером **32К**, отводится под текстовый видеобуфер видеопамяти. Обращаясь по этим адресам, можно записывать и выводить на экран графическую или текстовую информацию.

В графическом режиме каждой точке экрана соответствует своя ячейка видеопамяти, в которой храниться информация о цвете пиксела. В зависимости от выбранного видеорежима, число разрядов этой ячейки может быть различным (в зависимости от выбранного количества цветов – глубины цвета). Таким образом, систему координат экрана в графическом режиме можно представить рис. 10.1, на котором

[1] Исследование вывода в текстовом режиме производилось в лабораторной работе **№7**, где достаточно подробно излагаются теоретические основы.

показан режим 640x350 пикселов. Адресуемым элементом является пиксел, позиция которого определяется номерами столбца и строки.

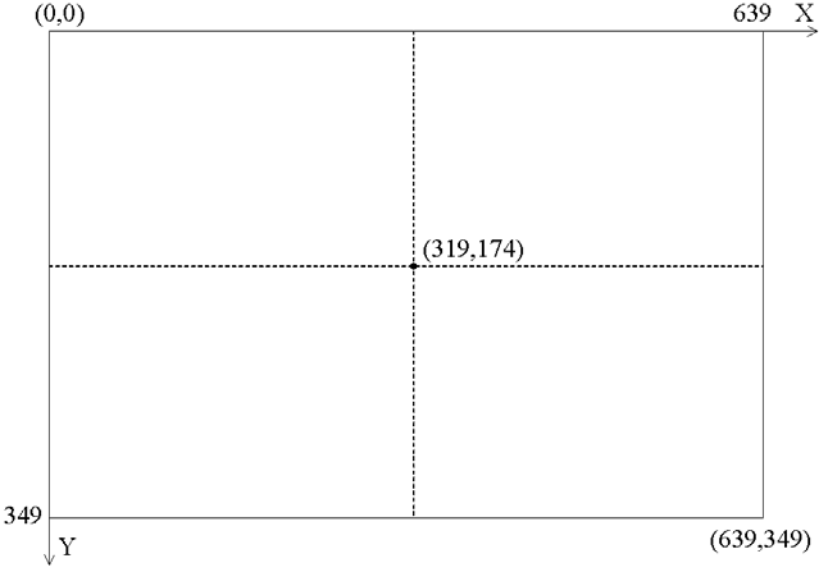

Рис. 10.1 – Система координат экрана в графическом режиме

Большинство видеоадаптеров могут работать как в текстовых, так и в графических режимах. Существует несколько стандартных режимов с **номерами 0…13h**, поддерживаемых практически всеми современными адаптерами. Они приведены в таблице 8.1 (лаб. работа №8).

Видеоадаптеры **SVGA** могут также работать в режимах имеющих улучшенные характеристики. Для этих режимов разработан стандарт **VESA** (**V**ideo **E**lectronics **S**tandards **A**ssociation), который определяет режимы работы с номерами **100h…11Ah**.

2.2 Способы вывода информации на экран

Операционная система предоставляет несколько способов вывода информации на экран:

– с использованием средств DOS (группа функций ввода-вывода из диапазона 01h…0Ch прерывания **INT 21h**). Поддерживается только текстовый монохромный режим вывода;

– с использованием средств BIOS с помощью прерывания **INT 10h**. Позволяет реализовать все возможности видеоадаптеров в текстовом и графическом режимах. Используется в реальном режиме работы МП;

– прямое программирование видеопамяти.

2.3 Вывод графической информации на экран средствами BIOS

Программы BIOS находятся в ПЗУ BOIS и вызываются посредством прерываний. Работа с видеоадаптером производится с помощью прерывания **INT 10h**. Основные функции прерывания **INT 10h**, предназначенные для работы с видеоадаптерами приведены в таблице 8.2 лабораторной работы №8.

Для обращения к нужной функции видеоадаптера необходимо выполнить следующую последовательность действий:

– загрузить в регистр **AH** номер нужной функции BIOS видеоадаптера;

– загрузить остальные регистры МП в соответствии с вызываемой функцией);

– выполнить прерывание **INT 10h**.

Графический адаптер может обеспечивать хранение и отображение нескольких страниц. По умолчанию активной (видимой) является страница 0, однако рисовать изображение и текст можно и на невидимых страницах. Для переключения страниц используется **функция 05h** прерывания **INT 10h**.

Для режимов с **16 отображаемыми цветами** (режимы **VGA**) могут использоваться разные наборы цветов (палитры), которые можно программно изменять. По умолчанию устанавливается стандартная палитра, номера цветов которой приведены в таблице 10.1.

Таблица 10.1

Коды цветов стандартной палитры **VGA**

Код (номер цвета)	Цвет	Код (номер цвета)	Цвет
00h	Чёрный	08h	Серый
01h	Синий	09h	Голубой
02h	Зелёный	0Ah	Салатовый
03h	Бирюзовый	0Bh	Светло-бирюзовый
04h	Красный	0Ch	Розовый
05h	Фиолетовый	0Dh	Светло-фиолетовый
06h	Коричневый	0Eh	Жёлтый
07h	Белый	0Fh	Ярко-белый

2.4 Построение окружности на экране

Окружность можно описать параметрическими уравнениями вида

$$x = xc + r\cos(angl);$$
$$y = yc + r\sin(angl), \qquad (10.1)$$

где *xc* и *yc* – координаты центра окружности;

r – радиус окружности;

angl – угол поворота радиус-вектора вокруг центра.

Поскольку при вычислении тригонометрических функций требуется использовать вещественные числа, такие задачи целесообразно решать с использованием математического сопроцессора. Углы в тригонометрических функциях Ассемблера должны указываться в радианах, поэтому необходимо выполнить перевод из градусов в радианы в соответствии с переводной формулой

$$angl[рад] = \frac{pi}{180} angl\ [град].$$

Исходными данными являются:

– *x360* = 180,0 – вещественная константа необходимая для перевода градусов в радианы;

– *x36* = 360 – целая константа, задающая число точек окружности;

– *forcolor* – переменная для хранения кода цвета окружности;

– *xc, yc* – целые числа, указывающие координаты центра окружности;

– *rx, ry* – целые числа, задающие радиус окружности по оси X и Y (радиусы могут быть разными из-за разного разрешения экрана в графических режимах);

– *xc_yc* – символьная переменная, выводимая на экран, которая сообщает координаты центра;

– *x, y* – целые переменные, для хранения вычисленных значений координат точки окружности;

– *angl* – переменная, задающая угол поворота радиус-вектора вокруг центра с начальным значением 1 градус;

– регистр **CX** используется как счетчик циклов при вычислении координат, которые следует повторить 360 раз.

Общую идею алгоритма можно сформулировать так:

– установить графический режим вывода на экран 320x200x16;

– заполнить экран фоном заданного цвета с помощью процедуры **Fon**;

– задавая угол поворота радиус-вектора (с начального значения 1 градус) вычислить координаты точки окружности в соответствии с уравнениями (10.1). Вычисленные значения округлить до целого и сохранить в переменных *x, y*;

– точку с вычисленными координатами *x,y* заданного цвета вывести на экран процедурой **Point**;

– цикл вычислений следует повторить еще 359 раз, пока не будут рассчитаны и построены все 360 точек окружности;

– на экран, в центр построенной окружности, с помощью процедуры **Simv**, вывести символьную строку *xc_yc*, указывающую координаты центра построенной окружности;

– завершить программу.

В соответствии с изложенным, алгоритм вычислений можно разбить на несколько функциональных участков:

– блоки 1, 2, 4, 5 – подготовительные операции;

– блок 3 – процедура заполнения экрана фоном заданного цвета;

– блоки 6…9 – вычисление коэффициента перевода градусов в радианы;

– блоки 10…19, 21, 22 – вычисление координат точек окружности;

– блок 20 – процедура вывода точки окружности на экран;

– блок 23 – процедура вывода надписи;

– блок 24 – процедура ожидания нажатия клавиши;

– блок 25 – завершающие операции.

Процедура **Point** выводит пиксел с цветом *forcolor* в позицию экрана с координатами (*xc*+*x*), (*yc*–*y*). Это выполняется с помощью функции вывода пиксела с номером 12 прерывания BIOS **INT 10h.**

Процедура **Fon** предназначена для заполнения экрана фоном заданного цвета. Такое заполнение можно выполнить, если последовательно (в цикле по столбцам и по строкам) выводить в каждую позицию графического экрана пиксел нужного цвета. Это выполняется с помощью функции вывода пиксела с номером 12 прерывания BIOS **INT 10h**.

Процедура **Simv** позволяет вывести символьную переменную, определенную в сегменте данных *xc_yc*, в нужную позицию экрана. Несмотря на использование графического режима, позиция устанавливается с помощью перемещения курсора (функция номер 2) на указанные координаты знакоместа. Использование для вывода функции с номером 0Eh (вывод символа в режиме телетайпа) позволяет автоматически смещать курсор в следующую позицию. Задание кода цвета с 1 в разряде D7 позволяет сохранить цвет фона при выводе символов.

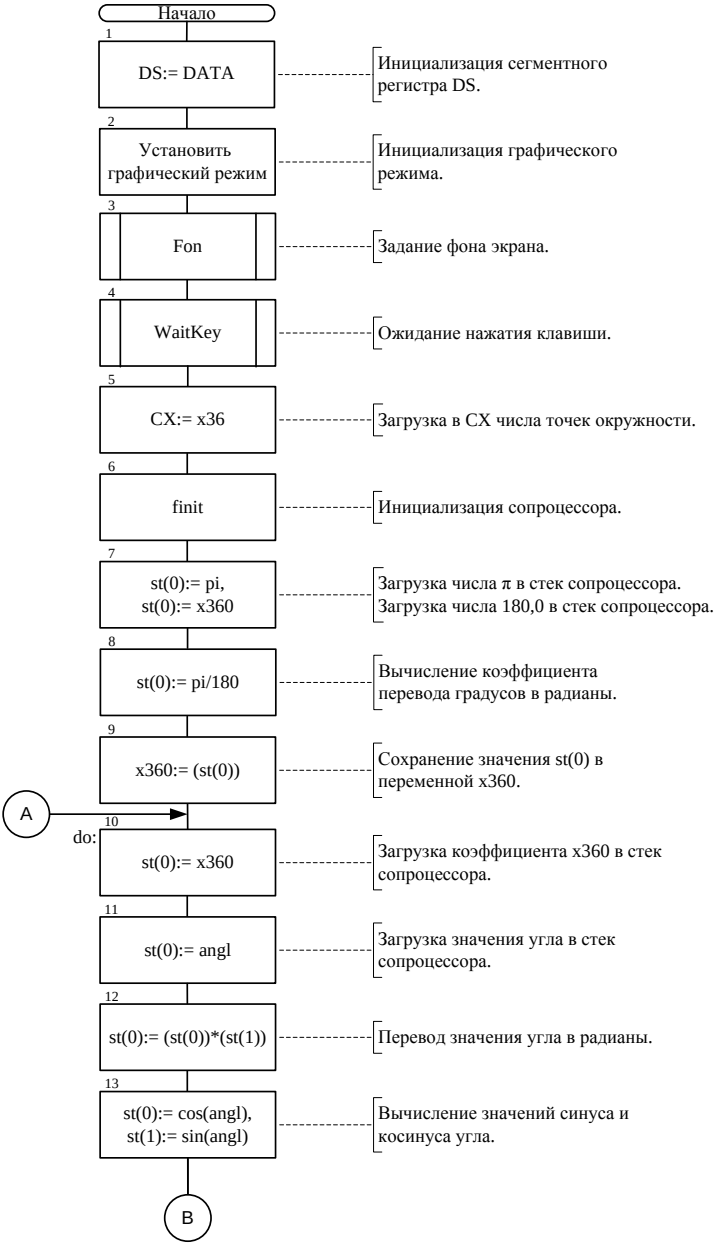

Рис. 10.2 – Алгоритм построения окружности

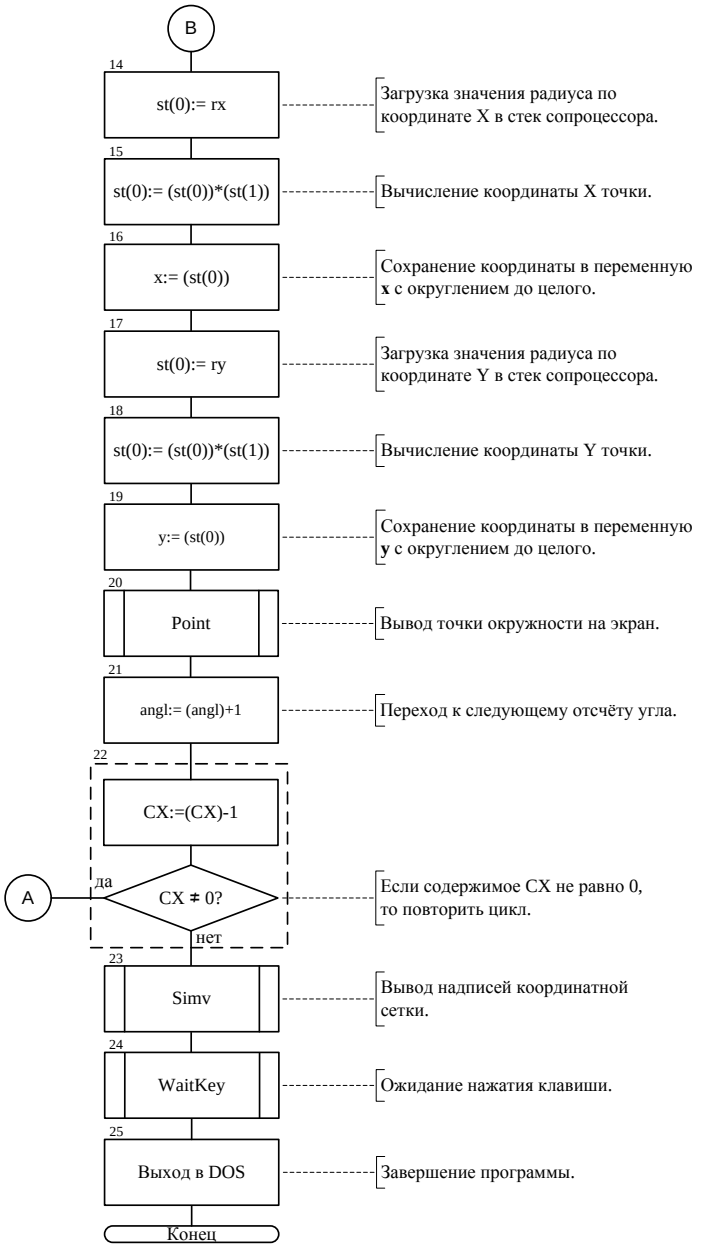

Рис. 10.3 – Продолжение алгоритма построения окружности

2.5 Команды сопроцессора для вычисления трансцендентных функций

Команды трансцендентных функций предназначены для вычисления значений тригонометрических функций, таких как синус, косинус, тангенс, арктангенс, а также значений логарифмических и показательных функций с высокой точностью. Значения аргументов в командах, вычисляющие результат тригонометрических функций, должны задаваться в радианах. Для нахождения радианной меры угла по его градусной мере необходимо число градусов умножить на pi/180, число минут – на pi/(180*60), а число секунд – на pi/(180*60*60) и найденные произведения сложить.

– **FCOS** – команда вычисляет косинус угла, находящийся в вершине стека сопроцессора – регистре **st(0)**. Команда не имеет операндов. Результат возвращается в регистр **st(0).**

– **FSIN** – команда вычисляет синус угла, находящийся в вершине стека сопроцессора – регистре **st(0)**. Команда не имеет операндов. Результат возвращается в регистр **st(0).**

– **FSINCOS** – команда вычисляет синус и косинус угла, находящиеся в вершине стека сопроцессора – регистре **st(0)**. Команда не имеет операндов. Результат возвращается в регистрах **st(0)** и **st(l)**. При этом синус помещается в **st(0)**, а косинус – в **st(1).**

– **FPTAN** – команда вычисляет частичный тангенс угла, находящийся в вершине стека сопроцессоре – регистре **st(0)**. Команда не имеет операндов. Результат возвращается в регистрах **st(0)** и **st(l).**

– **FPRATAN** – команде вычисляет частичный арктангенс угла, находящийся в вершине стека сопроцессора – регистре **st(0)**. Команда не имеет операндов. Результат возвращается в регистрах **st(0)** и **st(l).**

3 Подготовка к работе

3.1. Изучить методические указания и рекомендованную литературу.

3.2. Подготовить ответы на контрольные вопросы.

4 Задание на выполнение работы

4.1 Используя, описанный в разделе 2, алгоритм расчета и вывода на экран окружности разобрать исходный текст программы **CIRCLE.** Создать и отладить исполняемый модуль программы **CIRCLE**, выполнив этапы ассемблирования и компоновки. Добавить в исходный модуль программы недостающие комментарии.

4.2 Отредактировать исходный модуль программы **CIRCLE** для своего варианта задания (таблица 10.2). Создать и отладить исполняемый модуль программы **CIRCL_X** (**X** – номер варианта), выполнив

этапы ассемблирования и компоновки.

Таблица 10.2

Исходные данные

Для всех вариантов количество цветов 16						
№ варианта	xc	yc	rx	Цвет изображения	Цвет фона	Режим экрана
1	150	150	120	красный	фиолет.	640x350
2	100	100	80	черный	желтый	320x200
3	300	200	130	зеленый	синий	640x480
4	250	150	100	красный	белый	640x350
5	130	100	80	белый	черный	320x200
6	300	250	200	голубой	желтый	640x480
7	300	180	140	зеленый	белый	640x350
8	160	100	90	желтый	черный	320x200
9	400	280	150	черный	белый	640x480
10	500	100	100	розовый	синий	640x350
11	140	100	95	белый	голуб.	320x200
12	600	400	40	красный	серый	640x480
13	150	180	140	синий	розовый	640x350
14	90	90	90	фиолет.	белый	320x200
15	450	240	130	бирюз.	серый	640x480
16	400	100	95	черный	салат.	640x350

Примечание: Значения радиуса окружности по оси y (ry) может не совпадать со значением радиуса по оси x (rx). Его следует подобрать из условия отсутствия искажений окружности.

4.3 Добавить в программу фрагмент, который выводит точку заданного цвета в центр окружности.

```
TITLE CIRCL
;Программа вычисления координат точек окружности и вывод их на экран
;Входные параметры:
;  константа для перевода градусы в радианы x360
;  число точек на окружности x36
;  цвет пикселя forcolor
;  координаты центра окружности xc и yc
;  значения радиусов по осям x и y rx, ry
;  строка с координатами центра окружности xc_yc
;Выходные параметры:
;  координаты точки на окружности x и y
;  угол поворота радиус-вектора для точки на окружности angl
```

```
.MODEL SMALL              ;Модель памяти ближнего типа.
.STACK 256                ;Отвести под стек 256 байт.
.486                      ;Используем расширенную систему команд.
.DATA                     ;Открыть сегмент данных.
   x360 DD 180.0          ;Константа перевода градусы–радианы.
   x36 DW 360             ;Число точек на окружности.
   forcolor DB 0Ah        ;Салатовый цвет.
   xc DW 150              ;Координаты центра
   yc DW 100              ;окружности.
   rx DW 100              ;Значения радиуса по оси x.
   ry DW 80               ;Значения радиуса по оси y.
   xc_yc DB '150,100','$' ;Выводимые значения координат.
;================= Переменные ==========================
   x DW ?                 ;Координата точки окружности x.
   y DW ?                 ;Координата точки окружности y.
   Angl DW 1              ;Угол поворота радиуса.
;-------------------------------------------------------------------------------- ------
.CODE                     ;Открыть сегмент кодов.
;================ Вывод пикселя ========================
  Point PROC
;CX – координата X (столбец), DX – координата Y (строка),
;AL – цвет пиксела
     pusha
     mov CX, xc           ;Вычисляем координату
     add CX, x            ;x в регистре CX.
     mov DX, yc           ;Вычисляем координату
     sub DX, y            ;y в регистре DX.
     mov AL, forcolor
     mov BH, 0
     mov AH, 12           ;Вывести пиксел
     int 10h              ;средствами BIOS.
     popa
     ret
  Point ENDP
;============ Закрашивание экрана цветом фона ==============
  Fon PROC
;CX – координата X (столбец), DX – координата Y (строка),
;AL – цвет пиксела
        pusha
        mov CX, 0
        mov DX, 0
```

```
        mov AL, 05h             ;Цвет фона.
        mov BH, 0               ;Номер страницы.
 c_1:   mov AH, 12              ;Вывести пиксел
        int 10h                 ;средствами BIOS.
        inc CX
        cmp CX, 319
        jne c_1
        xor CX, CX
        inc DX
        cmp DX, 199
        jne c_1
        popa
        ret
 Fon ENDP
;================ Вывод символа ========================
 Simv PROC
        pusha
        mov AH, 02              ;Функция установки курсора.
        mov BH, 0               ;Номер текущей страницы.
        mov DL, 20              ;Номер столбца.
        mov DH, 12              ;Номер строки.
        int 10h                 ;Установка курсора.
        lea SI, xc_yc           ;Загрузить смещение строки в SI.
 c_2:   mov AH, 0Eh             ;Функция вывода символа.
        mov BL, 84h             ;Выбор цвета символов.
        loadsb                  ;Переслать символ из строки DS:SI в AL.
        cmp AL, '$'             ;Определить конец строки.
        je exit_pr              ;Если конец строки достигнут, выход.
        int 10h
        jmp c_2
 exit_pr: popa
        ret
 Simv ENDP
;================ Ожидание нажатия клавиши ===============
 WaitKey PROC
    pusha
    mov AH, 08h
    int 21h
    popa
    ret
 WaitKey ENDP
```

```
;=============== Главная процедура =======================
Main PROC
;Подготовка данных
      mov AX, @DATA       ;Инициализация
      mov DS, AX          ;регистра DS.
      mov AH, 0           ;Установка графического
      mov AL, 0Dh         ;режима 320x200x16
      int 10h             ;средствами BIOS.
      call Fon            ;Вызов процедуры закрашивания фона.
      call WaitKey        ;Вызов процедуры задержки.
      mov CX, x36         ;Число шагов построения окружности.
      finit               ;Инициализация сопроцессора.
      fldpi               ;Загрузка в стек числа pi.
      fld x360            ;Загрузка в стек числа 180.
      fdiv                ;pi/180, результат в ST(0).
      fstp x360           ;Сохранение в памяти коэффициента
                          ;перевода градусов в радианы.
;---------- Вычисление координат точек окружности ----------
  do: fld x360            ;Коэффициент градус->радиан в стек.
      fild  angl          ;Очередное значения угла в стек.
      fmul                ;Перевод в радианы.
      fsincos             ;sin(x) -> st(1), cos(x) -> st(0).
      fild rx             ;Загрузка радиуса по координате x.
      fmul                ;Вычисление координаты x=rx*cos(angl).
      fistp x             ;Запись ее в память в формате целого
                          ;числа с извлечением из стека.
      fild  ry            ;Загрузка радиуса по координате y.
      fmul                ;Вычисление координаты y=ry*sin(angl).
      fistp y             ;Запись ее в память в формате целого
                          ;числа с извлечением из стека.
      fwait               ;Ожидание завершения работы сопроцессора.
      call Point          ;Вывод точки на экран.
      inc Angl            ;Приращение угла.
      loop do             ;Повторить цикл, пока CX не 0.
      call Simv
      call WaitKey
      mov AX, 4C00h       ;Выход в
      int 21h             ;DOS.
Main ENDP
  END Main
```

5 Требования к отчёту

Отчёт должен содержать:

– титульный лист с указанием названия ВУЗа, кафедры, номера и темы лабораторной работы, а также фамилии И.О. студента, подготовившего отчёт;
– цель работы;
– вариант задания;
– листинги программы **CIRCL_X** с комментариями;
– результат работы программы **CIRCL_X**.

6 Контрольные вопросы

6.1 Что представляет собой система координат экрана в графическом режиме?
6.2 Какие способы вывода информации на экран существуют?
6.3 Назовите основные типы видеоадаптеров и их особенности.
6.4 Какие действия позволяют построить на экране графическое изображение?
6.5 Как происходит задание фона экрана?
6.6 Какие действия позволяют вывести пиксел в нужную точку экрана?
6.7 Какие регистры входят в программную модель сопроцессора?
6.8 Как организован стек сопроцессора? Физические и логические номера регистров.
6.9 Какой из регистров и каким образом используется в качестве указателя стека?
6.10 Назначение и формат регистра тегов TWR.
6.11 С данными каких форматов может работать сопроцессор?
6.12 Назначение и формат регистра состояний SWR.
6.13 Назначение управляющего регистра сопроцессора CWR.
6.14 Формат одинарной точности – короткое вещественное.
6.15 Формат двойной точности – длинное вещественное.
6.16 Расширенный формат – расширенное вещественное.
6.17 Назначение и операнды команд передачи данных сопроцессора.
6.18 Назначение и операнды арифметических команд сопроцессора.
6.19 Что такое исключения сопроцессора?
6.20 Поясните алгоритм работы программы **CIRCL**.

7 Рекомендуемая литература

7.1 Юров, В. И. Assembler [Текст]: учеб. пособие для вузов / В. И. Юров. 2-е изд. – СПб.: Питер, 2007. с. 447…509.
7.2 Рудаков, П.И. Язык Ассемблера: уроки программирования [Текст] / П. И. Рудаков, К. Г. Финогенов. – М.: ДИАЛОГ-МИФИ, 2001. – с. 45…47, 255…280, 563…593.

Лабораторная работа №11
Программирование математического сопроцессора

1 Цель работы

Изучение принципов работы сопроцессора и методов его программирования средствами Ассемблера. Изучение графического режима вывода на экран и методов его программирования. Знакомство с макросредствами Ассемблера.

2 Теоретический материал

2.1 Макросредства Ассемблера

Макросредства позволяют упростить исходный текст ассемблерных программ за счет использования приемов, использующихся в языках программирования более высокого уровня. Макросредства позволяют:

– выполнять или не выполнять трансляцию некоторых участков программы в зависимости от заданного условия (условная трансляция) (см. рекомендованную литературу);

– осуществлять размножение участка исходного текста программы, в том числе с модификацией каждого повторения (блоки повторения);

– включать в программу написанные отдельно фрагменты с настройкой их текста в соответствии с заданными параметрами (макрокоманды).

Объекты, создаваемые с помощью макросредств, называют **макросами**.

Макросы повторения позволяют транслятору повторить заданный блок исходного текста несколько раз. Повторяемый блок может быть описан в сегменте данных с помощью директив описания данных или в сегменте кода и состоять из команд микропроцессора. Например, следующий макрос повторения, включенный в сегмент данных программы, позволяет сформировать массив, состоящий из заглавных символов английского алфавита:

```
sym = ‘A’           ;Начальное значение элемента массива.
symbols:            ;Имя массива для ссылок на него
      rept 26       ;Повторять 26 раз.
      db sym        ;Повторяемая директива.
      sym = sym+1   ;Сформировать код следующего символа.
   endm             ;конец макроса
```

Как видно, **макрос повторения** начинается с директивы Ассемблера **rept** (повторение – repetition) и заканчивается директивой **ENDM** (конец макроса). Более подробно о макросах повторения можно прочитать в рекомендованной литературе.

Макрокоманды чаще всего используются для замены повторяющихся участков исходного текста с одинаковой структурой.

Макрокоманда (макроопределение) должно начинаться заголовком с именем макроопределения и директивой **macro**, за которой может следовать список формальных параметров (в простейшем случае параметров может и не быть). Вслед за заголовком располагается текст макроопределения. Заканчивается макроопределение директивой endm.

Например, в программе требуется неоднократно сохранять в стеке содержимое трех регистров, но в каждом конкретном случае имена регистров и их порядок отличаются. Эти действия можно оформить в виде макроопределения:

```
push3 MACRO A, B, C   ;начало макроса с именем push3
      push A          ;первая команда макроса
      push B          ;вторая команда макроса
      push C          ;третья команда макроса
ENDM                  ;конец макроса
```

Обращение к макросу производится заданием в тексте программы его имени и списка формальных аргументов. Например, строка

```
push3 CX, BX, AX; вызов макроса push3
```

приведет к генерации строк исходного текста программы:

```
push CX
push BX
push AX
```

Если макроопределение разработано специально для конкретной программы, то его можно целиком включить в текст программы (обычно в начало). Однако из полезных макроопределений можно создать библиотеку макрокоманд и хранить ее в виде файла. Макрокоманды записываются в этот файл по тем же правилам, что и в ассемблерную программу. Для того, чтобы транслятору были доступны макрокоманды, включенные в библиотеку, его следует подсоединить к исходному тексту программы директивой

include <имя файла>.

В этом случае все макрокоманды, входящие в файл с указанным именем, будут доступны из любого места программы.

2.2 Алгоритм выполняемой лабораторной работы

Архимедова спираль – плоская кривая, траектория точки *M* (рис. 11.1), которая равномерно движется вдоль луча *OV* с началом в *O*, в то время как сам луч *OV* равномерно вращается вокруг *O*. Другими словами, расстояние $\rho = OM$ пропорционально углу поворота φ луча *OV*. Повороту луча *OV* на один и тот же угол соответствует одно и то же приращение ρ.

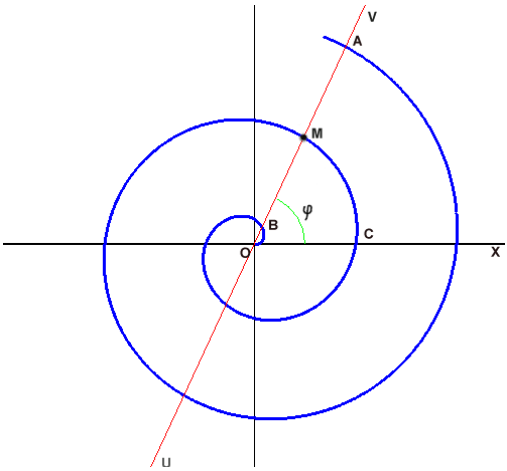

Рис. 11.1 – Спираль Архимеда

Уравнение Архимедовой спирали в полярной системе координат записывается как $\rho = k\varphi$, где *k* – смещение точки *M* по лучу, при его повороте на угол φ равный одному радиану.

Повороту прямой на 2π соответствует смещение $a = |BM| = |MA| = 2k\pi$. Число a называется шагом спирали. Уравнение Архимедовой спирали можно переписать так:

$$\rho = \frac{a}{2\pi}\varphi. \qquad (11.1)$$

При $\varphi = 0$ значение $\rho = 0$ и точка спирали *M* находится в центре *O*, при $\varphi = 2\pi$, точка спирали *M* находится в точке *C* на расстоянии *a* от центра. Далее по мере увеличения угла φ она будет уда-

ляться от центра на величину шага *a* с каждым витком. При вращении луча против часовой стрелки получается правая спираль (см. рис. 11.2), при вращении по часовой стрелке – левая спираль.

Обе ветви спирали (правая и левая) описываются одним уравнением (11.1). Положительным значениям соответствует правая спираль, отрицательным — левая спираль. Если точка *M* будет двигаться по прямой *UV* из отрицательных значений через центр вращения *O* и далее в положительные значения, вдоль прямой *UV*, то точка *M* опишет обе ветви спирали.

Луч *OV*, проведённый из начальной точки *O*, пересекает спираль бесконечное число раз – точки *B, M, A* и так далее. Расстояния между точками *B* и *M, M* и *A* равны шагу спирали. При раскручивании спирали, расстояние от точки *O* до точки *M* стремится к бесконечности, при этом шаг спирали остаётся постоянным (конечным), то есть, чем дальше от центра, тем ближе витки спирали, по форме, приближаются к окружности.

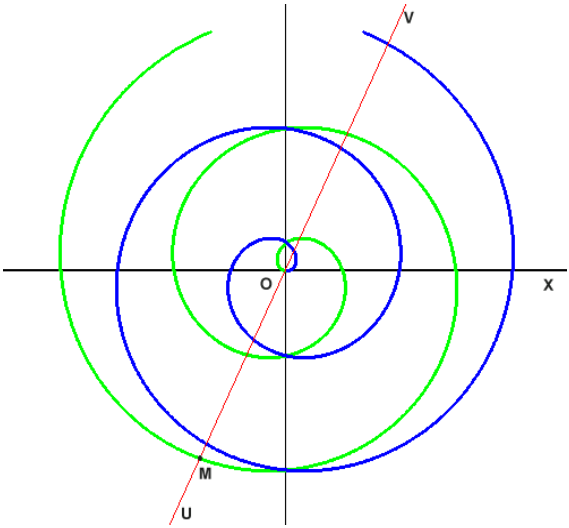

Рис. 11.2 – Правая и левая спирали Архимеда

В системе координат X,Y спираль с центром в начале координат можно описать уравнениями, подобными уравнениям окружности с изменяющимся радиусом:

$$x = \frac{a}{2\pi}\varphi * \cos(\varphi) = k\varphi * \cos(\varphi);$$
$$y = \frac{a}{2\pi}\varphi * \sin(\varphi) = k\varphi * \sin(\varphi). \quad , \qquad (11.2)$$

где параметр k будет определять шаг спирали, а угол φ поворот луча вокруг центра.

Поскольку при вычислении тригонометрических функций требуется использовать вещественные числа, такие задачи целесообразно решать с использованием математического сопроцессора.

Для получения достаточной точности изображения спирали выберем шаг изменения угла поворота φ равным 0,001рад. Такой угол будет соответствовать углу $\varphi[град] = \frac{\varphi[рад]}{pi}180$. Подставив значение в радианах получим 0,057 град.

Будем изменять угол φ от 0 до некоторого конечного значения с шагом 0,001 рад, для каждого значения угла рассчитывать координаты точки спирали *x, y* и выводить их на экран. При $\varphi = 0$ точка спирали находится в центре экрана О, при $\varphi = 2\pi$ точка находится на расстоянии С от центра и далее по мере увеличения угла она будет удаляться от центра на величину шага спирали с каждым ее витком.

Исходными данными являются:

```
fi dd 0.0              ;начальное значение переменной угла
                       ;(вещественное)
delta dd 0.001         ;шаг изменения угла в радианах
                       ;(вещественное)
xdiv2 dw 320           ;координаты центра экрана по X (целое)
ydiv2 dw 240           ;координаты центра экрана по Y (целое)
K dd 5.0               ;коэффициент, определяющий шаг спирали
                       ;(вещественное)
xr dw 0                ;координата выводимой точки по X (целое)
yr dw 0                ;координата выводимой точки по Y (целое)
forcolor DB 0Ah        ;салатовый цвет (цвет спирали)
Variant DB 'XXXXXX$' ;поясняющая надпись
```

Регистр **CX** используется как счетчик циклов при вычислении координат, которые следует повторить $\frac{2\pi}{delta} * (число\ витков)$ раз.

Основные шаги алгоритма, отражающие его общую идею можно сформулировать так:

– установить графический режим вывода на экран;

– заполнить экран фоном заданного цвета с помощью процедуры **FON**;

– вывести на экран символьную строку Variant, используя макрос **OutCharG**;

– вывести на экран координатные оси, используя макросы **AxleX** и **AxleY**;

– задавая угол поворота луча спирали (с начального значения 0 радиан) вычислить координаты точки окружности в соответствии с уравнениями (11.2). Вычисленные значения округлить до целого и сохранить в переменных **xr**, **yr**;

– точку с вычисленными координатами **xr**, **yr** заданного цвета вывести на экран процедурой **POINT**; цикл вычислений следует повторить тех пор, пока не будут рассчитаны и построены все витки спирали;

– перейти в текстовый режим и завершить программу.

В соответствии с изложенным, алгоритм вычислений, который показан на рис. 11.3…11.5, можно разбить на несколько функциональных участков:

– блоки 1, 2, 3 – подготовительные операции;

– блок 4 – процедура заполнения экрана фоном заданного цвета;

– блоки 5 и 6– вывод на экран символьной строки и осей координат;

– блоки 7…14– вычисление координаты X точек спирали;

– блоки 15…21– вычисление координаты Y точек спирали;

– блок 22 – процедура вывода точки спирали на экран;

– блоки 23…25 – изменение переменной цикла (угла fi) на шаг приращения и проверка условия выхода из цикла;

– блок 26…28 – завершающие операции.

Процедура **POINT** выводит пиксел с цветом **forcolor** в позицию экрана с координатами *(xdiv2+xr), (ydiv2–yr).* Это выполняется с помощью функции вывода пиксела с **номером 12** (**0Ch**) прерывания BIOS **INT 10h**.

Процедура **FON** предназначена для заполнения экрана фоном заданного цвета. Такое заполнение можно выполнить, если последовательно (в цикле по столбцам и по строкам) выводить в каждую позицию графического экрана пиксел нужного цвета. Это также выполняется с помощью функции вывода пиксела с **номером 12** прерывания BIOS **INT 10h**.

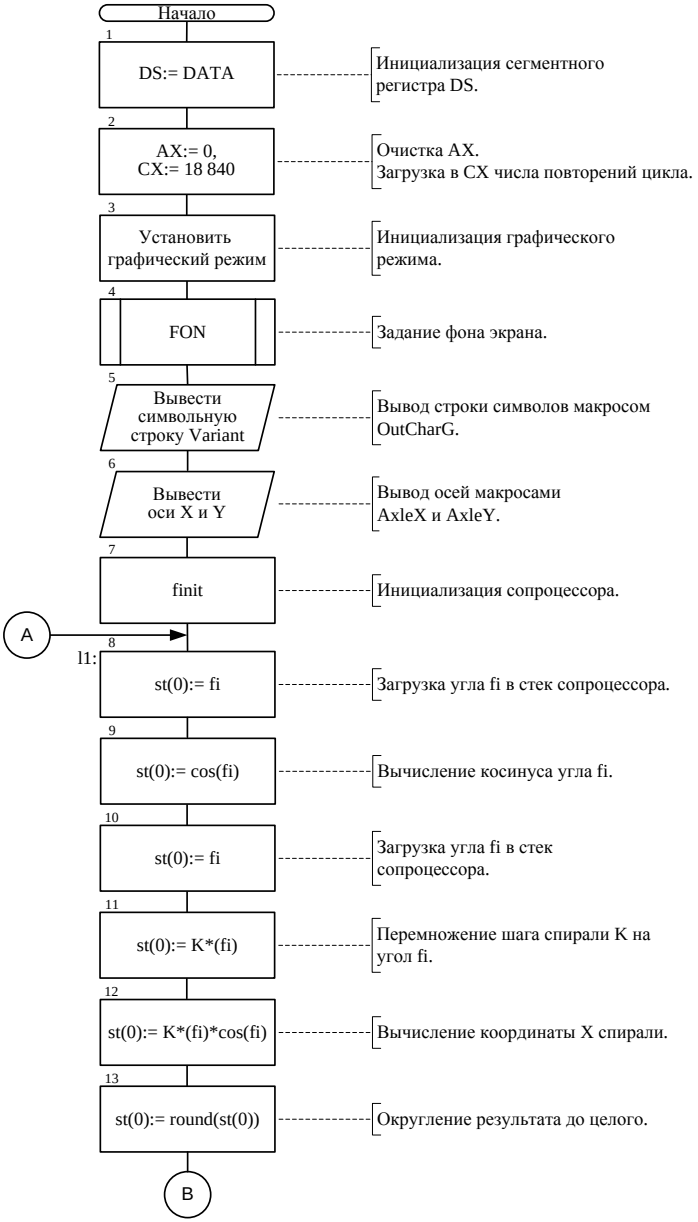

Рис. 11.3 – Алгоритм программы

Рис. 11.4 – Продолжение алгоритма программы

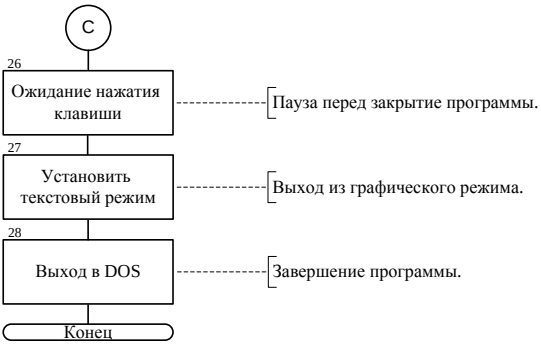

Рис. 11.5 – Продолжение алгоритма программы

3 Подготовка к работе

3.1. Изучить методические указания и рекомендованную литературу.
3.2. Подготовить ответы на контрольные вопросы.

4 Задание на выполнение работы

4.1 Используя, описанный в разделе 2, алгоритм расчета и вывода на экран спирали разобрать исходный текст программы SPIRAL. Создать и отладить исполняемый модуль программы SPIRAL.EXE, выполнив этапы ассемблирования и компоновки. Добавить в исходный модуль программы недостающие комментарии.

4.2 Программа использует для вывода символов и построения осей системы координат **макросы**, которые располагаются в отдельном файле с именем **pixels.inc** текст которого приводится ниже. Этот текстовый файл требуется создать и сохранить в вашем рабочем каталоге.

4.3 Отредактировать исходный модуль программы **SPIRAL** для своего варианта задания (таблица 11.1).

4.4 Создать и отладить исполняемый модуль программы **SPIRALXX.EXE** (**XX – номер варианта**), выполнив этапы ассемблирования и компоновки.

Таблица 11.1

Исходные данные

Вариант	Направление витков спирали	Число витков	Цвет изобр/фон	Режим экрана	Вариант	Направление витков спирали	Число витков	Цвет изобр/фон	Режим экрана
1	левое	4	Красн/фиол	640x350	9	левое	7	Черн/бел	640x480
2	правое	3	Черн/желт	320x200	10	правое	4	Розов/син	640x350
3	левое	7	Зелен/син	640x480	11	левое	3	Белый/гол	320x200
4	правое	5	Красн/бел	640x350	12	правое	8	Красн/сер	640x480
5	левое	3	Белый/черн	320x200	13	левое	5	Синий/роз	640x350
6	правое	6	Голуб/желт	640x480	14	правое	4	Фиол/бел	320x200
7	левое	4	Зелен/бел	640x350	15	левое	6	Бирюз/сер	640x480
8	правое	2	Желт/черн	320x200	16	правое	5	Черн/сал	640x350

```
TITLE SPIRAL
;Программа построения спирали Архимеда

include pixels.inc ;подключить макросы вывода точки, осей и символа

.model small

.stack 100h

.data
    fi dd 0.0                       ;Начальное значение переменной угла.
    delta dd 0.001                  ;Шаг изменения угла.
    xdiv2 dw 320                    ;Координаты центра экрана по X.
    ydiv2 dw 240                    ;Координаты центра экрана по Y.
    K dd 5.0                        ;Коэффициент шага спирали.
    xr dw 0                         ;Координата выводимой точки по X.
    yr dw 0                         ;Координата выводимой точки по Y.
    forcolor DB 0Ah                 ;Цвет спирали (салатовый).
    Variant db 'Spiral_XX_3_vitka', '$' ;Поясняющая надпись.

.code
.486                            ;Используем расширенную систему команд
;---------------Вывод пиксела-----------------------------------------------
;CX-координата X (столбец), DX-координата Y (строка),
;AL-цвет пиксела
POINT PROC
        pusha
        mov  CX,xr              ;Вычисляем координату x
        add  CX, xdiv2          ;в регистре CX.
        mov  DX, ydiv2          ;Вычисляем координату y
        sub  DX, yr             ;в регистре DX.
        mov  AL, forcolor       ;Задать цвет спирали.
```

```
        mov  BH, 0
        mov  AH,12              ;Вывести пиксел
        int 10h                 ;средствами BIOS.
        popa
        ret
POINT ENDP
;----------Закрашивание экрана цветом фона--------------------
;CX-координата X (столбец), DX-координата Y (строка),
;AL-цвет пиксела
FON PROC
        pusha
        mov CX, 0
        mov DX, 0
        mov AL, 05h             ;Цвет фона.
        mov  BH, 0              ;Номер страницы.
c_1:    mov AH, 12              ;Вывести пиксел
        int 10h                 ;средствами BIOS.
        inc CX
        cmp CX, 639
        jne c_1
        xor CX, CX
        inc DX
        cmp DX, 479
        jne c_1
        popa
        ret
FON ENDP
;==========Основная программа====================
start:
        mov ax,@DATA
        mov ds,ax
        xor ax,ax
        mov CX, 18840           ;Количество итераций цикла
                                ;(определяет число витков).
;-------------------------------------------------------------------------------
        mov ah,0h               ;Инициализация графического
        mov al,12h              ;режима 640x480.
        int 10h
        call FON                ;Вызов процедуры закрашивания фона.
;-------Вывод строки Variant-----------------------------------------------
        pusha
        mov cx,17
        mov bx,0
 l3:
        mov al,Variant[bx]
        inc bx
        OutCharG bl, 02h, 03h, al  ;Вызов макроса.
        loop l3
        popa
;---------рисуем оси-----------------------------------------------------------
        AxleX                   ;Вызов макроса.
        AxleY                   ;Вызов макроса.
```

```
;------Вычисляем формулу x=round(fi*K*cos(fi))----------------------
        finit                   ;Инициализация сопроцессора.
 l1:
        fld fi                  ;Загрузить угол fi в стек FPU.
        fcos                    ;Вычислить cos(fi).
        fld fi                  ;Загрузить угол fi в стек FPU.
        fmul K                  ;ST(0):=K*(ST(0))
        fmul                    ;ST()):=(ST(0))*(ST(1))
        frndint                 ;ST(0):=round(ST(0))
        fistp word ptr xr       ;Заносим X в переменную для вывода
                                ;на экран
;------Вычисляем формулу y=round(fi*K*sin(fi))----------------------
        fld fi
        fsin
        fld fi
        fmul K
        fmul
        frndint
        fistp word ptr yr       ;Заносим Y в переменную для вывода
                                ;на экран.
        call POINT
 ;---------Вычисляем новое значение угла fi----------------------------
        fld delta
        fld fi
        fadd
        fstp fi
        loop l1                 ;Повторить цикл пока (CX) не равно 0.
;-------------------------------------------------------------------------------
        mov ah,1h               ;Ожидание нажатия клавиши.
        int 21h
;-------------------------------------------------------------------------------
        mov ah,0h               ;Перевод  в Text Mode.
        mov al,03h
        int 10h
;-------------------------------------------------------------------------------
 exit:
        mov ax,4C00h            ;Стандартный выход.
        int 21h
  END  start

;Исходный текст файла макросов pixels.inc
;-------------------------------------------------------------------------
;Макрос вывода символа в графическом режиме
;(char - ASCII код символа)
OutCharG macro x, y, color, char
        pusha
        mov ah,02h
        mov bh,0h
        mov dh,y
        mov dl,x
        int 10h
        mov ah,09h
```

```
            mov al,char
            mov bh,0h
            mov bl,color
            mov cx,01h
            int 10h
            popa
endm
;-------------------------------------------------------------------------------
;Макрос вывода пиксела на экран с коорд. x,y и цветом color
PutPixel macro x,y,color
            pusha
            mov ah,0ch
            mov al,color
            mov bh,0h
            mov cx,x
            mov dx,y
            int 10h
            popa
endm
;-------------------------------------------------------------------------------
;Макрос вывода горизонтальной линии в середине экрана
AxleX macro
            local iter
            pusha
            OutCharG 4eh,0fh,03h,78h ;X
            mov cx,640
  iter:
             PutPixel cx,240,4h
            loop iter
            PutPixel 637,241,4h         ;стрелка
            PutPixel 637,239,4h
            PutPixel 636,241,4h
            PutPixel 636,239,4h
            PutPixel 635,241,4h
            PutPixel 635,239,4h
            PutPixel 634,241,4h
            PutPixel 634,239,4h
            PutPixel 633,241,4h
            PutPixel 633,239,4h
            PutPixel 632,242,4h
            PutPixel 632,238,4h
            PutPixel 633,242,4h
            PutPixel 633,238,4h
            PutPixel 632,241,4h
            PutPixel 632,239,4h
            PutPixel 634,242,4h
            PutPixel 634,238,4h
            popa
endm
;-------------------------------------------------------------
;Макрос вывода вертикальной линии в середине экрана
AxleY macro
```

```
        local iters
        pusha
        mov cx,480
 iters:
        mov dx,cx
        PutPixel 320,dx,4h
        dec cx
        cmp cx,19
        jge iters
        PutPixel 319,22,4h          ;Стрелка.
        PutPixel 321,22,4h
        PutPixel 319,23,4h
        PutPixel 321,23,4h
        PutPixel 319,24,4h
        PutPixel 321,24,4h
        PutPixel 318,25,4h
        PutPixel 322,25,4h
        PutPixel 318,26,4h
        PutPixel 322,26,4h
        PutPixel 318,27,4h
        PutPixel 322,27,4h
        PutPixel 319,26,4h
        PutPixel 321,26,4h
        PutPixel 319,27,4h
        PutPixel 321,27,4h
        PutPixel 319,25,4h
        PutPixel 321,25,4h
        OutCharG 29h,01h,03h,79h ;Y
        popa
endm
```

5 Требования к отчёту

Отчёт должен содержать:

– титульный лист с указанием названия ВУЗа, кафедры, номера и темы лабораторной работы, а также фамилии И.О. студента, подготовившего отчёт;

– цель работы;

– вариант задания;

– листинги программы **SPIRALXX** с комментариями;

– результат работы программы **SPIRALXX**.

6 Контрольные вопросы

6.1 Что представляет собой система координат экрана в графическом режиме?

6.2 Какие способы вывода информации на экран существуют?

6.3 Последовательность каких действий позволяют построить на экране графическое изображение?

6.4 Как происходит задание фона экрана?

6.5 Какие действия позволяют вывести пиксел в нужную точку экрана?
6.6 Как организован стек сопроцессора? Физические и логические номера регистров.
6.7 Какой из регистров и каким образом используется в качестве указателя стека?
6.8 С данными каких форматов может работать сопроцессор?
6.9 Формат одинарной точности – короткое вещественное.
6.10 Формат двойной точности – длинное вещественное.
6.11 Расширенный формат – расширенное вещественное.
6.12 Назначение и операнды команд передачи данных сопроцессора.
6.13 Назначение и операнды арифметических команд сопроцессора.
6.14 Что такое исключения сопроцессора?
6.15 Поясните алгоритм работы программы SPIRAL.
6.16 Какими выражениями можно описать построение спирали в декартовой системе координат?
6.17 Поясните назначение макросредств Ассемблера.
6.18 Опишите правила создания макроса повторения в сегменте данных программы.
6.19 Опишите правила создания макрокоманд, которые можно использовать ассемблерными программами.
6.20 Где размещаются и как вызываются макросы?

7 Рекомендуемая литература

7.1 Assembler. Учебник для вузов. 2-е изд. [Текст] /В. И. Юров – СПб.: Питер, 2004. , с.293…323, с.447…509.
7.2 Рудаков, П. И. К. Г. Финогенов. Язык Ассемблера: уроки программирования [Текст] / П. И. Рудаков, К. Г. Финогенов:. – М.: ДИАЛОГ-МИФИ, 2001. с.86…89, 255…280, 563…593.
7.3 Финогенов, К. Г. Использование языка Ассемблера: учеб. пособие для вузов [Текст] / К. Г. Финогенов. – М.: Горячая линия-Телеком, 2004. – с. 96…106.

Лабораторная работа №12
Программный генератор случайной последовательности чисел

1 Цель работы

Практическое овладение навыками программирования на Ассемблере. Исследование линейного конгруэнтного метода генерации случайной последовательности. Работа с процедурами и системными прерываниями.

2 Теоретический материал

2.1 Алгоритм генерации случайной последовательности

Линейный конгруэнтный метод – один из алгоритмов генерации псевдослучайных чисел. Применяется в простых случаях и не обладает криптографической стойкостью. Входит в стандартные библиотеки различных компиляторов. Этот алгоритм заключается в итеративном применении целочисленного деления и получения остатка в соответствии с выражением:

$$X_{n+1} = (aX_n + c) \bmod m, \qquad (12.1)$$

где n – номер итерации;

a, c и m – константы;

X_0 – начальное значение (энтропия);

mod – операция вычисления остатка от деления ($(aX_n + c)/m$).

Получаемая последовательность зависит от выбора стартового числа $\boldsymbol{X_o}$ и при разных его значениях получаются различные последовательности случайных чисел. В то же время, многие свойства последовательности $\boldsymbol{X_n}$ определяются выбором коэффициентов в формуле и не зависят от выбора стартового числа. Ясно, что последовательность чисел, генерируемая таким алгоритмом, периодична с периодом, не превышающим $\boldsymbol{m}$. Статистические свойства получаемой последовательности случайных чисел полностью определяются выбором констант $\boldsymbol{a}$ и $\boldsymbol{c}$ при заданной разрядности. Поэтому выбор указанных коэффициентов следует производить с учетом рекомендаций, указанных в таблицах задания на выполнение работы. Начальное значение последовательности $\boldsymbol{X_o}$ должно быть случайной величиной, поэтому его получают из данных системного таймера компьютера.

Обобщенный алгоритм работы программы **RAND** показан на рис. 12.1. Как видно, он состоит из 8 функциональных блоков.

В блоке 1 – выполняется инициализация сегментного регистра.

В блоке 2 – происходит подготовка экрана к диалогу с пользователем. Она включает установку текстового режима ввода и очистку экрана.

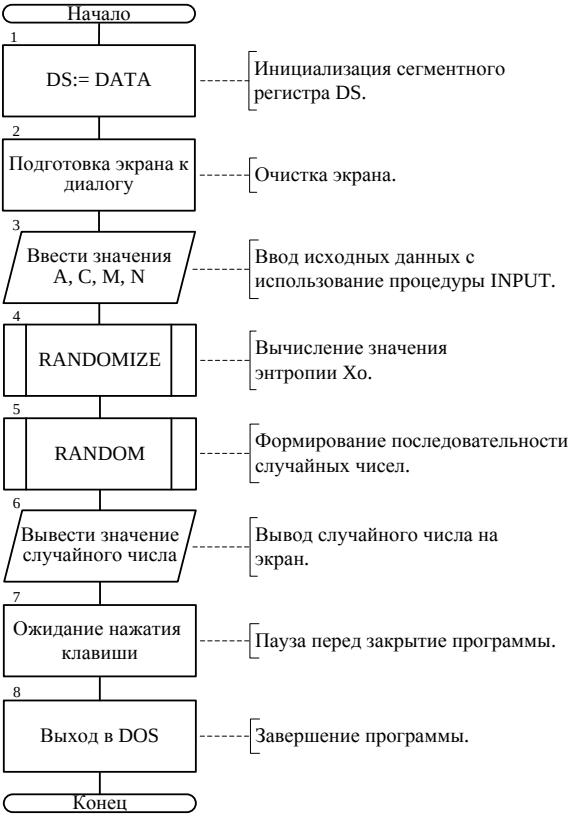

Рис. 12.1 – Алгоритм формирования случайной последовательности

В блоке 3 – выполняется интерактивный ввод исходных данных, необходимых для создания случайной последовательности. При этом на экране формируется запрос на ввод параметров выражения (12.1), а после ввода параметра происходит проверка на отсутствие ошибок ввода и его преобразование (параметра) в двоичный вид. Для этого используется процедура INPUT.

В блоке 4 – происходит формирование начального значения случайной последовательности $\mathbf{X_o}$ (энтропии) с помощью процедуры RANDOMIZE.

В блоке 5 – выполняется программная генерация элементов слу-

чайной последовательности в соответствии с выражением (12.1) с помощью процедуры RANDOM.

В блоках 6 и 7 – созданный в памяти массив случайных чисел преобразуется в символьный вид и выводится на экран. Для этого используется процедура OUTPUT.

В блоке 8 – происходит корректное завершение программы.

Как следует из описания алгоритма, он, в основном, использует процедуры, которые позволяют упростить процесс отладки программы. Рассмотрим структуры процедур более подробно.

Процедура «Randomize» – используется для инициализации генератора случайной последовательности, т. е. для присвоения начального значения переменной $\mathbf{X_0}$ в которой хранится начальное случайное число. Это начальное значение получаем с помощью системного прерывания DOS **INT 21h** с номером **2Ch**, в результате выполнения которого в регистры микропроцессора помещаются значения:
в **CH** – часы (от 0 до 23);
в **CL** – минуты (от 0 до 59);
в **DH** – секунды (от 0 до 59);
в **DL** – сотые доли секунды (от 0 до 99).
Полученные данные сохраняются в регистр **EDX**, при этом в его старшую часть записывается значения секунд и долей секунд, а в младшую часть значения часов и минут. При этом происходит дополнительное кодирование полученных данных. Выходным параметром процедуры является значение переменной $\mathbf{X_0}$ в которой сохраняется **32 битное случайное число.**

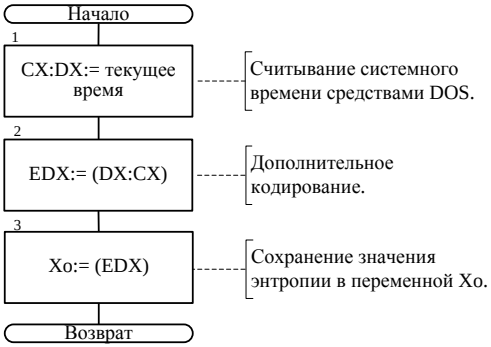

Рис. 12.2 – Процедура RANDOMIZE

В **Процедура Random** (рис. 12.3) выполняется арифметический расчет выражения по формуле (12.1), т.е. происходит расчет чисел

случайной последовательности из которых образуется массив **mas_1**. Входными параметрами процедуры являются значения энтропии $\mathbf{X_o}$, параметров уравнения (12.1) **A**, **C**, **M** и заданной длины последовательности **N**.

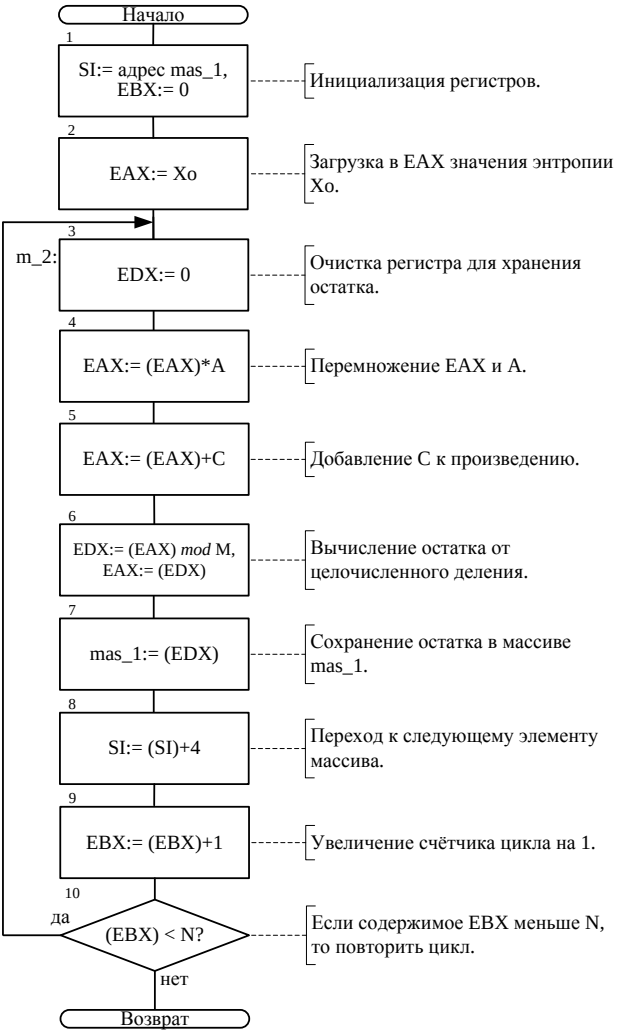

Рис. 12.3 – Процедура RANDOM

Выходным параметром процедуры является массив значений случайных чисел **mas_1**, которые формируются из целочисленных ос-

татков деления на **M**. Размер случайных чисел – двойное слово. Для формирования массива используется косвенная индексная адресация с помощью регистра **SI**. Регистр **EBX** используется в качестве счетчика циклов, который следует повторять **N** раз.

Процедура OUTPUT производит вывод на экран символьных значений сформированного массива случайных чисел. Входными параметрами процедуры являются массив случайных чисел, хранящийся в переменной **mas_1** и число случайных чисел, хранящееся в переменной **N.** Выходными параметрами процедуры являются символьные значения случайных чисел, которые формируются с помощью еще одной процедуры **PREOBR** и сохраняются в символьной переменной **Y_ASCII**. Символьные значения **Y_ASCII** выводятся на экран. Регистры: **EBX** – счетчик циклов, **SI** – указатель индексов элементов массива.

Процедура PREOBR предназначена для преобразования чисел в символьный вид. Данная процедура аналогична процедуре, использованной в лабораторной работе №6, за исключением использования 32-битных регистров.

3 Подготовка к работе

3.1. Изучить методические указания и рекомендованную литературу.
3.2. Подготовить ответы на контрольные вопросы.

4 Задание на выполнение работы

4.1 Используя описанный алгоритм работы программы разобрать исходный текст программы **RAND**, приведенный ниже. Создать и отладить исполняемый модуль программы **RANDXX.EXE**, используя исходные данные Вашего варианта (табл. 12.1).
Дополнить текст программы недостающими комментариями. Отладить программу используя отладчик.
4.2 Упростить программу, добавив в нее процедуру вывода на экран строковых переменных.
4.3 Сгенерировать с помощью созданной программы последовательность из 100 элементов, используя при генерации параметры в соответствии с таблицей 12.1.

При генерации указать диапазон генерируемых чисел в соответствии с таблицей 12.2.

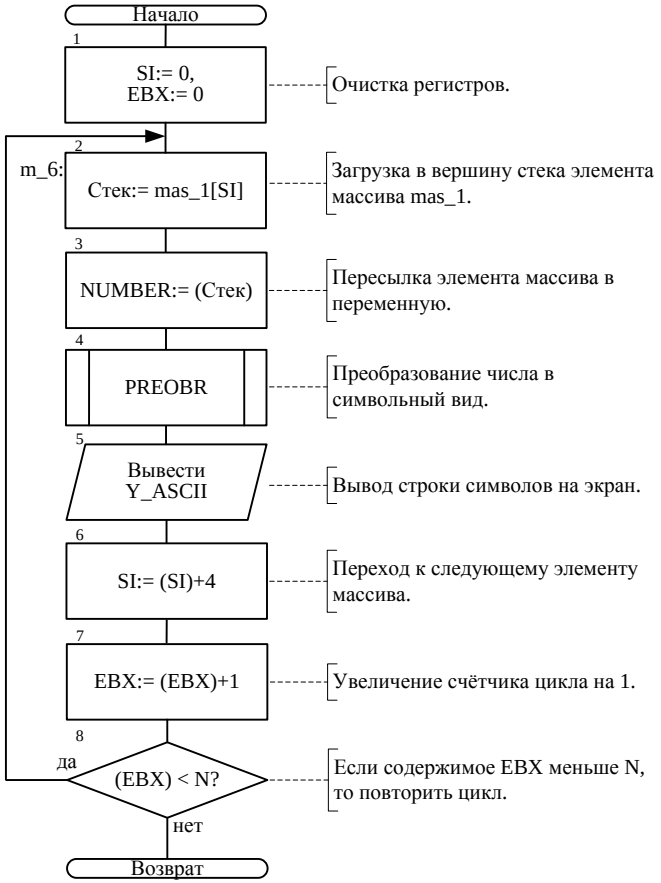

Рис. 12.4 – Процедура OUTPUT

Таблица 12.1

Исходные данные

Вариант	m	a	c
1/ 9	2^{32}	1664525	1013904223
2/ 10	2^{32}	22695477	1
3/ 11	2^{32}	69069	5
4/ 12	2^{32}	1103515245	12345
5/ 13	2^{32}	134775813	1
6/ 14	2^{32}	214013	12345
7/ 15	2^{32}	1103515245	12345
8	$2^{31} - 1$	16807	0

Таблица 12.2

Варианты заданий

Вариант	N	Вариант	N
1	0	9	1000
2	524	10	732
3	65536	11	1048576
4	512	12	1024
5	30	13	800
6	90	14	600
7	890	15	1000
8	700		

```
TITLE RAND
;Генератор формирования последовательности из N случайных
;чисел по конгруэнтному методу

.MODEL  SMALL
.486
.STACK 100h
.DATA
  Ask_A    db 'X(n+1) = (A*X(n)+ C) mod M', 13, 10, 'input A ?', 13, 10, '$'
  Ask_C    db 13, 10, 'input C ?', 13, 10, '$'
  Ask_M    db 13, 10, 'input M ? [1...255]', 13, 10, '$'
  Ask_N    db 13, 10, 'input length N ?', 13, 10,' $'
  Err_msg  db 'Error input'          ;Сообщение об ошибке ввода.
  Crlf       db 0Dh, 0Ah, '$'        ;Перевод строки, возврат каретки.
  Msg      db 13, 10, 'Press any key', 13, 10, '$'
  Blength  db   ?                    ;Длина буфера после считывания.
  A dd 0                             ;Переменная для параметра A.
  C dd 0                             ;Переменная для параметра C.
  M dd 0                             ;Переменная для параметра M.
  N dd ?                             ;Переменная для длины последовательности N.
  Xo dd 0                            ;Переменная для энтропии Xo.
  mas_1 dd 255 dup (0)               ;Переменная для массива случайных чисел.
  Y_ASCII  db 12 DUP (?)             ;Переменная для хранения ASCII
                                     ;кодов случайных чисел.
  Sign     db (?)                    ;Переменная для хранения знака числа.
  Number   db (?)                    ;Переменная для хранения случайного числа.
  Del      dd 10                     ;Делитель для процедуры Preobr.
  Buffer   db 10                     ;Буфер для вводимых чисел в конце
                                     ;сегмента данных.

.CODE
;=====================================================
Input PROC
;Процедура осуществляет ввод цифровых данных с клавиатуры и
;помещает их в буфер.
            mov DX, offset Buffer      ;Считать строку
```

```
        mov AH, 0Ah             ;символов
        int 21h         ;в буфер.
        mov DX, offset Crlf
        mov AH, 9               ;Перевод
        int 21h         ;строки.
        mov SI, offset Buffer+1 ;Адрес ячейки с количеством цифр
                                ;в буфере.
        xor ECX, ECX
        xor EAX, EAX
        xor EBX, EBX
        xor DI,  DI
        mov CX, 10              ;Записать множитель для преобразования.
        mov BL, [SI]            ;Записать количество цифр в буфере
        mov Blength, BL         ;в переменную Blength.
;----------------------- Обрабатываем введенные цифры -----------------------
  m_1:  inc SI                  ;Устанавливаем указатель на первую цифру.
        mov BL, [SI]            ;Взять первый символ.
        sub  BL, '0'            ;Преобразовать первый символ в цифру.
        jb error
        cmp BL, 9
        ja error
        mul ECX                 ;Умножить текущий результат на 10.
        add EAX, EBX            ;Добавить к нему новую цифру.
        dec Blength             ;Повторять цикл пока
        jnz m_1                 ;Blength не равна 0.
        ret
  error: mov DX, offset Err_msg
        mov AH, 9
        int 21h
        jmp exit                ;Завершить программу.
Input ENDP
;================================================
Randomize PROC
;Процедура инициализации генератора случайной последовательности.
;Начальное значение энтропии (Xo) берется из системных часов и
;сохраняется в переменную Xo.
        mov AH, 2ch     ;Получить системное время
        int 21h  ;средствами DOS:
                        ;CH – часы (0...23)
                        ;CL – минуты (0...59)
                        ;DH – секунды (0...59)
                        ;DL – сотые доли сек (0...99).
        shl EDX, 16     ;Сдвинуть мл.часть EDX на 16 бит влево.
        mov DX, CX      ;Записать в мл. часть EDX часы и минуты.
        mov Xo, EDX     ;Записать результат в переменную Xo.
        ret             ;Возврат.
Randomize ENDP
;================================================
Random PROC
;Процедура расчета значений случайной последовательности.
;Очередное значение случайной последовательности хранится в
;регистре EDX.
```

```
;Из значений формируется массива mas_1 с заданным числом элементов N.
;SI-указатель индексов элементов массива;
;BX-счетчик числа элементов массива.
        lea SI, mas_1       ;Записать адрес массива случайных чисел.
        xor EBX, EBX        ;Очистить счетчик элементов массива.
        mov EAX, Xo         ;Поместить в EAX значение Xo.
 m_2:   xor EDX, EDX        ;Очистить EDX.
        mul A               ;Умножить на A (Xn*A).
        add EAX, C          ;(Xn*A)+C
        div dword ptr M     ;((Xn*A)+C)mod M
        mov EAX, EDX        ;Записать остаток (случ. число) в EAX.
        mov [SI], EDX       ;Записать сл. число в массив.
        add SI,4            ;Перейти к следующему элементу массива.
        inc EBX             ;Сосчитать элемент массива.
        cmp EBX, N          ;Сравнить число элементов с заданным N.
        jb m_2              ;Перейти к началу цикла, если число < N.
        ret                 ;Возврат.
Random ENDP
;=================================================
Preobr PROC
;Процедура преобразования двоичнокодированного десятичного
;числа в символьный ASCII-код.
;Входные параметры: исходное число в переменной  Number.
;Выходные параметры: символьное представление числа Number в
;десятичной системе.
        mov EAX, Number     ;Поместить в AX исходное число.
        mov SIGN, ' '       ;Пробел (знак +) в переменную знак.
        cmp EAX, 0          ;Сравнить число с нулем.
        jns M_3             ;Если больше или равно 0, перейти на
                            ;метку m_3,
        mov Sign, '-'       ;иначе знак "-" в переменную знака.
        neg EAX             ;Преобразовать в прямой код.
 ;-------------------------------------------------------------------------
 m_3:   xor  CX,  CX
 m_4:   xor  EDX, EDX       ;Очистить регистр остатка деления.
        div  dword ptr Del  ;Выполнить деление на Del.
        push EDX            ;Сохранить остаток в стеке.
        inc  CX;
        cmp EAX, 0          ;Если (AX) не равно 0, то
        jne  m_4            ;повторить деление.
 ;-------------------------------------------------------------------------
        xor SI, SI          ;Очистить SI.
        mov AL, SIGN        ;Загрузить в AL знак числа.
        mov Y_ASCII[SI], AL ;Переслать знак в Y_ASCII.
        inc SI
 m_5:   pop EAX;            ;Извлечь содержимое стека в AX.
        add AL, 30h         ;Вычислить ASCII-код для цифры.
        mov Y_ASCII[SI] ,AL ;Переслать ASCII-код  в Y_ASCII.
        inc SI              ;Если содержимое CX не 0, то
        loop m_5            ;повторить цикл.
        mov Y_ASCII[SI], '$' ;Символ конца строки в Y_ASCII.
        ret                 ;Завершение процедуры.
```

```
Preobr ENDP
;=================================================
Output PROC
;Процедура вывода символьных значений массива случайных чисел
;на экран
        xor EBX, EBX    ;очистить счетчик числа случайных чисел
        xor SI, SI
  m_6:  push Mas_1[SI]          ;поместить элемент массива в стек
        pop Number              ;извлечь из стека в переменную Number
        pusha                   ;сохранить все регистры в стек
        call Preobr             ;Вызвать процедуру преобразования
                                ;в символы.
        popa                    ;Восстановить регистры из стека.
        mov DX, offset Y_ASCII ;Вывести очередное
        mov AH, 09h             ;значение
        int 21h                 ;на экран.
        add SI, 4               ;Перейти к следующему элементу массива.
        inc EBX                 ;Повторять цикл
        cmp EBX, N              ;пока число случ. чисел
        jb  m_6                 ;меньше заданного N.
        ret
  Output ENDP
;=================================================
Start:  mov AX, @DATA
        mov DS, AX
  ;-------------------- Подготовка экрана ----------------------------------
        mov AX, 0600h           ;Очистка
        mov BH, 07              ;экрана
        mov CX, 0000            ;прокруткой
        mov DX, 184Fh           ;вверх.
        int 10h
  ;-------------------- Ввод исходных данных -------------------------------
        mov DX,offset Ask_A     ;Вывести приглашение
        mov AH, 9               ;ко вводу
        int 21h                 ;параметра A.
        nop
        call Input
        mov A, EAX              ;Сохранить значение в A.
        mov DX, offset Ask_C    ;Вывести приглашение
        mov AH, 9               ;ко вводу
        int 21h                 ;параметра C.
        call Input
        mov C, EAX              ;Сохранить значение в C.
        mov DX, offset Ask_M    ;Вывести приглашение
        mov AH, 9               ;ко вводу
        int 21h                 ;параметра M.
        call Input
        mov M, EAX              ;Сохранить значение в M.
        mov DX, offset Ask_N    ;Вывести приглашение
        mov AH, 9               ;ко вводу длины
        int 21h                 ;случайной последовательности N.
        call Input
```

```
        mov N, EAX              ;Сохранить значение в N.
;----------------- Обработка исходных данных -------------------------------
        call Randomize          ;Вычислить энтропию Xo.
        call Random             ;Вычислить случ. последовательность.
        mov DX, offset Crlf
        mov AH, 9               ;Перевод
        int 21h                 ;строки.
        call Output             ;Вывести случайное числа на экран.
;----------------- Завершение программы ------------------------------------
        mov AH, 9               ;Вывести сообщение
        mov DX, offset Msg      ;о нажатии любой
        int 21h                 ;клавиши.
        mov AH, 00h             ;Пауза.
        int 16h
exit:   mov AX, 4C00h           ;Завершение
        int 21h                 ;программы.
END Start
```

5 Требования к отчёту

Отчёт должен содержать:

– титульный лист с указанием названия ВУЗа, кафедры, номера и темы лабораторной работы, а также фамилии И.О. студента, подготовившего отчёт;

– цель работы;

– вариант задания;

– листинги программы **RANDXX** с комментариями;

– результат работы программы **RANDXX**.

6 Контрольные вопросы

6.1 Команды арифметических операций. Формат, операнды и флаги?

6.2 С какой целью используются подпрограммы и процедуры?

6.3 Как выглядит типовая структура для организации процедуры?

6.4 По какому алгоритму вычисляются случайные числа последовательности в линейном конгруэнтном методе?

6.5 Что означает энтропия в выражении для расчета случайной числовой последовательности?

6.6 Как задается энтропия в исследуемом алгоритме?

6.7 Какие значения можно использовать для задания параметров генерации случайных чисел последовательности?

6.8 Как вывести число, хранящееся в регистре процессора на экран?

6.9 Какие регистры процессора вы знаете? Сколько их и какого они размера?

6.10 Как ввести число с экрана в какую-нибудь переменную?

6.11 Чем отличается случайная последовательность от псевдослучайной?

6.12 Почему вместо **idiv** используется команда **shr**, что она означает и как выполняется?
6.13 Чем отличаются команды **idiv** и **fidiv**?
6.14 Каким образом происходит генерация случайного числа в линейном конгруэнтном методе?
6.15 Для чего необходимо применять преобразование числа в символ для отображения его на экране?
6.16 На чем основан алгоритм преобразования чисел в символы?
6.17 Как можно ввести в программу паузу до нажатия клавиши?

7 Рекомендуемая литература

7.1 Assembler. Учебник для вузов. 2-е изд. [Текст] /В. И. Юров – СПб.: Питер, 2004. , с.293…323, с.447…509.
7.2 Рудаков, П. И. К. Г. Финогенов. Язык Ассемблера: уроки программирования [Текст] / П. И. Рудаков, К. Г. Финогенов:. – М.: ДИАЛОГ-МИФИ, 2001. с. 255…280, 563…593.

Краткая система команд микропроцессора i80X86

Команды передачи данных

```
MOV   dst, src    ; dst:= (src)
XCHG  op1, op2    ; op1:= (op2) ; op2:= (op1)
LEA   reg, mem    ; reg:= [mem]
PUSH  src         ; SP:= (SP)–2 ; [(SS):(SP)]:= (src)
POP   dst         ; dst:= [(SS):(SP)]; SP:= (SP)+2
```

Команды арифметических операций

```
ADD   dst, src    ; dst:= (dst)+(src)
ADC   dst, src    ; dst:= (dst)+(src)+CF
SUB   dst, src    ; dst:= (dst) – (src)
SBB   dst, src    ; dst:= (dst) – (src) –CF
MUL   src         ; AX:= (AL)* (src), DX:AX:= (AX)* (src),
                  ; умножение байтов или слов без знака
IMUL  src         ; умножение для чисел со знаками
DIV   src         ; целочисленное деление беззнаковых чисел
                  ;AL := quot ((AX)/(src)); частное при
                  ;делении на байт.
                  AH := rem ((AX)/(src)); остаток при
                  ;делении на байт.
                  AX := quot ((DX:AX)/(src)); частное
                  ;при делении на слово
                  DX := rem ((DX:AX)/(src)); остаток
                  ;при делении на слово.
IDIV  src         ; целочисленное деление чисел со знаками
CDW               ; преобразование байта в AL в слово в AX
                  DB —> DW
CWD               ; преобразование слова в AX в двойное слово в
                  DX:AX    DW —> DD
CMP   op1, op2    ; сравнение операндов (op1) –(op2)
INC   op          ; op:= (op) + 1
DEC   op          ; op:= (op) – 1
NEG   op          ; op:=  – (op)
```

Команды логических операций

```
OR    dst, src    ; dst := (dst) v (src)
XOR   dst, src    ; dst := (dst)⊕(src)
NOT   op          ; инверсия op
AND   dst, src    ; dst := (dst)^(src)
TEST  dst, src    ; флаги:=  (dst)^(src)
```

Команды сдвигов

```
SHL     op,N      ; логический влево
SAL     op,N      ; арифметический влево
SHR     op,N      ; логический вправо
SAR     op,N      ; арифметический вправо
ROL     op, N     ; циклический влево
ROR     op, N     ; циклический вправо
RCL     op, N     ; циклический влево через перенос
RCR     op, N     ; циклический вправо через перенос
```

Примечание: N=1 или содержимому регистра CL

Команды передачи управления

```
JMP     op        ; безусловный переход, op – метка или
                  ; адрес в регистре или ячейка памяти
```

Команды условных переходов по флагам

J*cond* метка ; условный переход к метке по флагу (признаку):
cond = C (CF=1), NC (CF=0), S (SF=1), NS (SF=0), Z (ZF=1), NZ (ZF=0), O (OF=1), NO (OF=0), P (PF=1), NP (PF=0).

Команды условных переходов по результатам операции сравнения

```
JE      метка     ; op1= op2  для любых чисел
JNE     метка     ; op ≠ op2  для любых чисел
```

Для чисел без знака		*Для чисел со знаком*		
JB / JNAE	метка;	JL / JNGE	метка	; при op1 < op2
JBE / JNA	метка;	JLE / JNG	метка	; при op1 <= op2
JA / JNBE	метка;	JG / JNLE	метка	; при op1 > op2
JAE / JNB	метка;	JGE / JNL	метка	; при op1 >= op2

Команды организации циклов

```
LOOP      метка          ; CX:= (CX)–1, переход к метке при
                         ; (CX)≠ 0
LOOPZ / LOOPE    метка   ; CX:= (CX)–1, переход к метке при
                         ; (CX)≠ 0 и ZF=1
LOOPNZ / LOOPNE  метка   ; CX:= (CX)–1, переход к метке при
                         ; CX≠ 0 и ZF=0
JCXZ      метка          ; переход к метке при (CX)=0
```

Команды управления флагами

```
CLD     ; DF <— 0          CLC     ; CF <— 0
STD     ; DF <— 1          STC     ; CF <— 1
CLI     ; IF <— 0          CMC     ; инверсия  CF
STI     ; IF <— 1
```

```
mov
cld
PrintLoop:
mov
inc
mov
int
cmp
jne
xor
```