МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯРОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Забайкальский государственный университет»

(ФГБОУ ВО «ЗабГУ»)

Энергетический факультет

Кафедра информатики, вычислительной техники и прикладной математики

**ОТЧЕТ**

по учебной практике

(технологическая (проектно-технологическая))

в                                            ФГБОУ ВО «ЗабГУ»

(полное наименование организации)

обучающегося               Банковского Александра Сергеевича

(фамилия, имя, отчество)

Курс 2 Группа ИВТ-20-1

Направление подготовки 09.03.01 Информатика и вычислительная техника

(шифр, наименование)

Направленность ОП Программное обеспечение вычислительной техники и автоматизированных систем

Руководитель практики от университета\_\_\_\_\_\_\_\_\_\_ старший преподаватель Палкин Г. А.

(Ученая степень, должность, Ф.И.О.)

Руководитель практики от предприятия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(должность, Ф.И.О.) подпись,  печать

г. Чита2022

Реферат

Данный отчет содержит:

* Введение
* Основная часть
  + Условия задачи
  + Описание теоретических сведений в выбранной области (математические формулы, при необходимости)
  + Описание основных процедур
  + Блок-схемы основных алгоритмов программы
  + Руководство пользователя и описание интерфейса
  + Исходный код программы в приложении
* Заключение
* Список литературы

Объем отчета

Содержание

[Введение 4](#_Toc115358363)

[Основная часть 5](#_Toc115358364)

[Теоретические сведения 6](#_Toc115358365)

[Описание основных процедур 9](#_Toc115358366)

[Процедура ввода строки c консоли 9](#_Toc115358367)

[Процедура @InputEnd 9](#_Toc115358368)

[Процедура проверки корректности введённой строки 9](#_Toc115358369)

[Процедура *@Error* 11](#_Toc115358370)

[Процедура преобразования цифр в слово 12](#_Toc115358371)

[Процедура сохранения числа в массив данных 13](#_Toc115358372)

[Процедура расчёта периметра по заданным координатам 14](#_Toc115358373)

[Процедура преобразования числа в строку и её вывод в консоль с использованием стека 15](#_Toc115358374)

[Блок-схема основного алгоритма программы 16](#_Toc115358375)

[Руководство пользователя и описание интерфейса 24](#_Toc115358376)

[Исходный код программы 26](#_Toc115358377)

[Заключение 33](#_Toc115358378)

[Литература 34](#_Toc115358379)

# Введение

Ассемблер – низкоуровневый язык программирования, позволяющий осуществить почти любые задумки программиста. Ассемблер в связи с ходом научно-технического прогресса, а именно появления более продвинутых высокоуровневых языков программирования, стал почти не востребованным на общем трудовом рынке в сфере IT. Но это совсем не означает, что ассемблер бесполезен на сегодняшний день. Напротив, освоение даже основ языка Ассемблер позволит программисту получше ознакомиться с мироустройством компьютера. При изучении Ассемблера придётся научиться работать с регистровой памятью; поподробнее соприкоснуться с разноразмерностью типов данных; дополнительно ознакомиться с модулем FPU, позволяющим работать с числами с плавающей точкой и т.д.  
 В некоторых аспектах любому хорошему программисту следует углублённо ознакомиться с темами косвенно или прямо касающихся Ассемблера: системы исчисления, строение чисел с плавающей точкой, машинный код, стек и другие. Они придадут большего понимания о работе компьютера, что положительно скажется на общем навыке любого программиста.

# Основная часть

**Условия задачи**:

Даны целые числа x1, y1, x2, y2, ..., x10, y10. Найти периметр десятиугольника, вершины которого имеют соответственно координаты (x1, y1), (x2, y2), ..., (x10, y10).

# Теоретические сведения

**Объявление переменных, констант, их инициализация в FASMW.**

Для того, чтобы объявить или инициализировать переменную, необходимо использовать псевдокоманду.

**Псевдокоманды.**

Операторы языка ассемблера делятся на команды и псевдокоманды (директивы).

*Команды ассемблера* — это символические имена машинных команд, обработка их компилятором приводит к генерации машинного кода. Псевдокоманды же управляют работой самого компилятора. В машинный код они не преобразуются, а просто выполняют какие-либо действия в процессе компиляции (аналогично меткам). На одной и той же аппаратной архитектуре могут работать различные ассемблеры: их команды обязательно будут одинаковыми, но псевдокоманды могут быть разными.

**Псевдокоманды DB, DW и DD.**

Чаще всего используется псевдокоманда DB (define byte), позволяющая определить числовые данные и строки, каждый элемент которых будет имеет размерность в 1 байт. Рассмотрим несколько примеров:

x1 DB 55 x1 – однобайтовая переменная – число 55

helpstr DB ‘S’, ‘O’, ‘S’ helpstr – строка, состоящая из однобайтовых символов.

*Примечание:* при подобной инициализации переменных, будь то x1 или helpstr, x1 фактически является адресом к области памяти; Обращаясь к x1, программист обращается к области памяти по адресу x1. Для того, чтобы обратиться к области памяти по определённому адресу, необходимо «заточить» сам адрес в квадратные скобки:

INC [x1] , где х1, к примеру – DS: 0014, а [x1] – значение,

хранящееся по данному адресу

В случае с helpstr, helpstr – адрес лишь на начало последовательности символов (на 1-ый символ в строке, которые идут друг за другом, занимая по 1-ому байту данных. Таким же образом можно инициализировать или объявлять массивы из чисел. Потребуется лишь не записывать элементы в кавычках или строфах)

HS DB 'Hello!', 13, 10, '$'

Символ доллара в кавычках означает конец строки, а символ доллара без кавычек – адрес текущей команды.

Для определения порции данных размера, кратного слову, служит директива DW (define word); Двойного слова – DD (Define double word) и т.д.

Также можно лишь объявлять переменные, не задавая им конкретных значений. Для этого необходимо при инициализации переменной в значение занести вопросительный знак - ?

Пример:

Primer1 DW ?

**Инициализация константы.**

Для определения константы нужно лишь указать имя константы и приравнять ей какое-либо значение. В таком случае не нужно будет обращаться к ней в квадратных скобках, ибо при обращении к имени константы, программист получит заданное значение этой константы.

Пример:

AlmostPI = 3

*Примечание:* Имя меток, переменных и констант должно быть уникальным и не совпадать с названиями команд, директивами и тому подобному Ассемблера.

**Процедура (подпрограмма)** — это основная функциональная единица декомпозиции (разделения на несколько частей) некоторой задачи. Процедура представляет собой группу команд для решения конкретной. В простейшем случае программа может состоять из одной процедуры. Процедуру можно определить и как правильным образом оформленную совокупность команд, которая, будучи однократно описана, при необходимости может быть вызвана в любом месте программы.

Синтаксис описания процедуры:

***@ИмяПроцедуры***:

… Тело процедуры …

**RET**

**CALL *@ИмяПроцедуры***— вызов процедуры (подпрограммы).

При вызове процедуры адрес команды переходит к вызванной подпрограмме, а чтобы после вернуться обратно, в стек записывает адрес команды равный адресу команды, идущему сразу же после вызова процедуры. Команда RET считывает адрес команды из стека и перенаправляет выполнение команду по адресу.

Незнание данного факта может привести к тому, что неопытный программист не очистит стек после его использования и команда RET не сможет помочь вернуться туда, куда планировалось.

**Метка** представляет собой символическое имя, вместо которого компилятор подставляет адрес. Имена переменных, объявленных с помощью директив объявления данных, тоже являются метками. Но с ними компилятор дополнительно связывает размер переменной. Метка объявляется очень просто: достаточно в начале строки написать имя и поставить двоеточие. И ещё лучше перед именем метки поставить «@» для лучшей читаемости кода. Например:

@m1:

mov ax,4C00h

int 21h

Имя метки может состоять из латинских букв, цифр и символов подчёркивания, но должно начинаться с буквы. Имя метки должно быть уникальным. В качестве имени метки нельзя использовать директивы и ключевые слова компилятора, названия команд и регистров.

|  |  |  |
| --- | --- | --- |
| **Команды INT и IRET: работа с прерываниями.**  Прерыванием называется такое событие, когда процессор приостанавливает нормальное выполнение программы и начинает выполнять другую программу (подпрограмму), предназначенную для обработки прерывания. Закончив обработку прерывания, он возвращается к выполнению приостановленной программы.  Все прерывания делятся на две группы: программные и аппаратные. Программные прерывания порождаются по команде INT. Команде INT нужно передать всего один операнд - номер нужного прерывания.  INT 21H – Одно из самых частых программных прерываний  Возврат из обработчика прерывания осуществляется с помощью команды **Массивы в программе.** Специальных средств описания массивов в программах ассемблера нет. При необходимости использовать массив в программе его нужно моделировать одним из следующих способов:   1. Перечислением элементов массива в поле операндов одной из директив описания данных. При перечислении элементы разделяются запятыми. К примеру:  |  | | --- | | ; массив из 5 элементов. Размер каждого элемента 2 байта:  Arr DW 1,2,3,4,5 |  1. Используя оператор повторения **DUP**. К примеру:  |  | | --- | | ; массив из 5-ти элементов - нулей.  ; Размер каждого элемента - 1 байт:  Arr DB 5 DUP (?) |   Перед **DUP** должно стоять значение – кол-во копий, а в скобках само значение этих копий. На примере объявлен массив из 5-ти неинициализированных эл-тов. Если посмотреть на Arr в памяти, то это будут 5 байтов нулевых значений. **Доступ к элементам массива** В общем случае для получения адреса элемента в массиве необходимо начальный (базовый) адрес массива сложить с произведением индекса (номер элемента минус единица) этого элемента на размер элемента массива:  **Адрес 1-ого эл-та + (индекс n-ого эл-та \* размер элемента).** |
|  |

# Описание основных процедур

## Процедура ввода строки c консоли

***Вход***: **STRING** – строка-буфер, состоящая из однобайтовых символов, с внесённой в 1-ый байт числом, являющимся максимальной длиной вводимой строки (включая символ переноса строки). Строка должна состоять из количества символов, рассчитываемого по формуле: 2 + максимальное кол-во вводимых символов строки + 1.

***Выход***: Введённая строка, которая начинается с 3-его байта в строке-буфере **STRING** и заканчивается символом ‘$’

***Замечание***: Окончанием вызываемой процедуры ввода строки является подпрограмма по метке **@InputEnd**, где увеличивается счётчик, содержащий номер вводимой строки (От 0 до **N**, где **N** – константа, которая равна количеству точек в многоугольнике), и сравнивается с **N**: при равенстве выполняется команда **RET**; при отсутствии равенства – возврат к **@InputString**.

***Код:***

**@InputString**:

MOV DX, STRING ; В регистре DX должен находиться

; адрес на начало строки-буфера

MOV AH, 0AH ; Функция **DOS 0AH** - ввод строки в

; буфер через консоль

INT 21H ; Программное прерывание для

; обращения к функции **DOS**

***Переменные***:

**STRING** DB 5, ?, 5 DUP('$')

## Процедура @InputEnd

***Код***:

**@InputEnd**:

INC WORD SI

CMP SI, N

JNE @InputString

RET

***Переменные:***

***N*** = 10

## Процедура проверки корректности введённой строки

***Вход***: **STRING** – строка-буфер, уже изменённая с помощью функции **DOS 0AH**.

***Функция***: В случае некорректно введённой строки программа досрочно завершается. А также автоматический перевод символа цифры в цифру.

***Код***:

**@CheckInputedString**:

; Inputed string is in STRING + 2

; DI – Counter

MOV DI, 0 ; В данной подпрограмме SI – регистр,

PUSH SI ; хранящий число, обозначающее знак

MOV SI, 1 ; введённого числа

**@CheckFS**:

; Check first symbol

CMP [STRING + 2], 2DH

JNE @IsEmpty

**@IsSign**:

INC DI

MOV SI, -1

**@IsEmpty**:

CMP [STRING + 2 + DI], 0DH

JE @Error

CMP [STRING + 2 + DI], 24H

JE @Error

**@IsNumber**:

; Check the digits

CMP [STRING + 2 + DI], '0'

JE @Correct

CMP [STRING + 2 + DI], '1'

JE @Correct

CMP [STRING + 2 + DI], '2'

JE @Correct

CMP [STRING + 2 + DI], '3'

JE @Correct

CMP [STRING + 2 + DI], '4'

JE @Correct

CMP [STRING + 2 + DI], '5'

JE @Correct

CMP [STRING + 2 + DI], '6'

JE @Correct

CMP [STRING + 2 + DI], '7'

JE @Correct

CMP [STRING + 2 + DI], '8'

JE @Correct

CMP [STRING + 2 + DI], '9'

JE @Correct

; Check if it was a last symbol - CR

CMP [STRING + 2 + DI], 0DH

JE @ReplaceCR

JNE @Error

**@Correct**:

; StrToInt: '0' - 30h = H [ASCII]

PUSH BX

MOV BL, [STRING + 2 + DI]

SUB BL, 30H

MOV [STRING + 2 + DI], BL

POP BX

; Going to next symbol

INC DI

JMP @IsNumber

**@ReplaceCR**:

MOV [STRING + 2 + DI], 24H

## Процедура *@Error*

***Функция***: Прекратить дальнейшее выполнение программы, оповестив об ошибке со стороны пользователя, а именно ввода строки, не соответствующей условиям задачи – не число, ничего или не целое число.

***Код***:

**@Error**:

MOV DX, error

MOV AH, 09H

INT 21H

JMP @Close

**@Close**:

; Read Enter before program closing

MOV DX, el

MOV AH, 0AH

INT 21H

; Closing console

MOV AH, 4CH

INT 21H

***Переменные***:

**error** DB 0AH, 'ERROR: Was inputed not a integer/number or nothing!', 0AH, 'Program ended ahead of schedule', 0AH, 'Press Enter to exit...', '$'

**el** DB 1, 2 DUP(?)

## Процедура преобразования цифр в слово

***Вход***: **STRING** – строка-буфер, уже изменённая с помощью функции **DOS 0AH** и с помощью процедуры **@CheckInputedString** (Для корректности перевода и перевода символов цифр в цифры). Регистр **SI** должен содержать значение отрицательной единицы, если было введено отрицательное число, и положительной единицы, если было введено положительное число.

***Выход***: Итоговое число, находящее в регистре **CX**.

***Функция***: Преобразование цифр из введённой строки с помощью функции **DOS 0AH** в число, которое можно использовать в расчётах в программе.

***Код***:

**@DigitsToNumber**:

PUSH BX

PUSH AX

; Constant multiplier

MOV BX, 10

CMP SI, -1

JE @Sign

; 10^n

MOV AX, 1

MOV BP, 3

JMP @StartDTN

**@Sign**:

MOV AX, -1

MOV BP, 4

**@StartDTN**:

; Here will be result number

MOV CX, 0

MOV DX, 0

; Get a length

MOV DL, [STRING + 1]

; Add 2, because first 2 bytes - max L and current L - not a ; string

ADD DL, 2

MOV DI, DX

MOV DX, 0

**@L1:**

MOV DX, WORD [STRING + DI - 1]

MOV DH, 0

MOV [TEMP], AX

IMUL DX, WORD [TEMP]

; DX = Digit \* 10^n

; (n = from STRING length to BP)

; Result = DX + Result

ADD WORD CX, WORD DX

CMP DI, BP

JE @Save

DEC DI

MUL BX

JMP @L1

***Переменные:***

**TEMP** DW ?

## Процедура сохранения числа в массив данных

***Вход***: Регистр **CX** должен содержать необходимое к сохранению число. Переменная **IsY** необходимадля выбора пути сохранения числа: при **IsY** равной нулю сохранение происходит в массив координат по **X**; при **IsY** равной единице – массив координат по **Y**. **IsY** должна назначаться перед началом сохранения числа, а лучше перед началом выполнения функции **DOS 0AH**. Массивы **X** и **Y** должны иметь размерность **N** + 1; По индексу **N** + 1 в массивах **X** и **Y** должны находиться координата по X и Y первой точки соответственно. Размерность элемента в массиве **X** и **Y** должна быть равна двум байтам.В регистре **SI** должен находиться номер вводимой строки по счёту.

***Выход***: В массивах **X** и **Y** будут храниться значения введённых координат.

***Код***:

**@Save**:

MOV [TEMP], CX

MOV AX, [TEMP]

MOV BX, SI

IMUL BX, 2

CMP [IsY], 0

JE @SaveX

**@SaveY**:

MOV [Y + BX], AX

JMP @InputEnd

**@SaveX**:

MOV [X + BX], AX

***Переменные***:

**N** = 10

**TEMP** DW ?

**X** DW N + 1 DUP(?)

**Y** DW N + 1 DUP(?)

## Процедура расчёта периметра по заданным координатам

***Вход***: Регистры **SI** и **DI** должны содержать адреса начала массивов координат **X** и **Y** соответственно.

***Выход***: Итоговый результат – периметр многоугольника будет располагаться в регистре **AX**.

***Функция***: Расчёт длины отрезка по координатам двух точек и дальнейшее суммирование длин всех отрезков в регистре **AX**.

***Код***:

MOV CL, 0

MOV AX, WORD 0

**@FindLength**:

FILD WORD [SI]

ADD SI, 2

FILD WORD [SI]

FSUB ST0, ST1

FLD ST0

FMUL ST0, ST1

FILD WORD [DI]

ADD DI, 2

FILD WORD[DI]

FSUB ST0, ST1

FLD ST0

FMUL ST0, ST1

FADD ST0, ST3

FSQRT

FISTP [R]

ADD AX, WORD [R]

FISTP [R]

FISTP [R]

FISTP [R]

FISTP [R]

FISTP [R]

INC CL

CMP CL, N

JNE @Calculating

***Переменные***:

**R** DW ?

## Процедура преобразования числа в строку и её вывод в консоль с использованием стека

***Вход***: В регистре **AX** должно находиться число, которое нужно преобразовать. В регистре **BL** должен находиться постоянный делитель равный 10. В вершине стека **SS** должен находиться символ конца строки – ‘$’.

***Функция***: Разбор числа на цифры, преобразование цифр с символы цифры, сохранение символов цифр в стек (Используя **PUSH**) для дальнейшего последовательного вывода сохранённых символов в консоли (Используя **POP** и команду **DOS** **02H**).

***Код***:

MOV DX, WORD 0

PUSH WORD '$'

**@IntToStr**:

DIV BL

; Now digit in AH/DX; Number without digit in AL/AX

MOV DH, AH

ADD DH, 30H

MOV DL, DH

MOV DH, 0

PUSH DX

MOV AH, 0

CMP AL, 0

JNE @IntToStr

MOV AH, 02H

**@ShowResult**:

POP DX

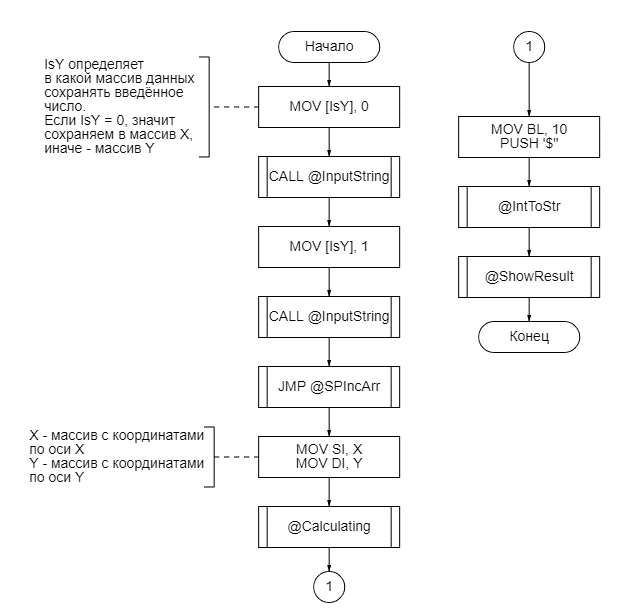
CMP DX, WORD '$'

JE @Close

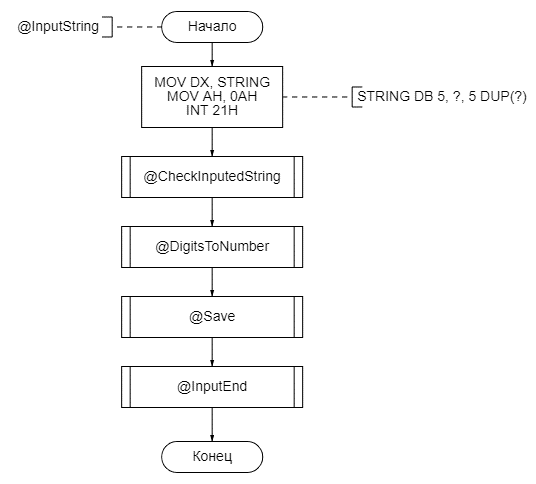
INT 21H

JMP @ShowResult

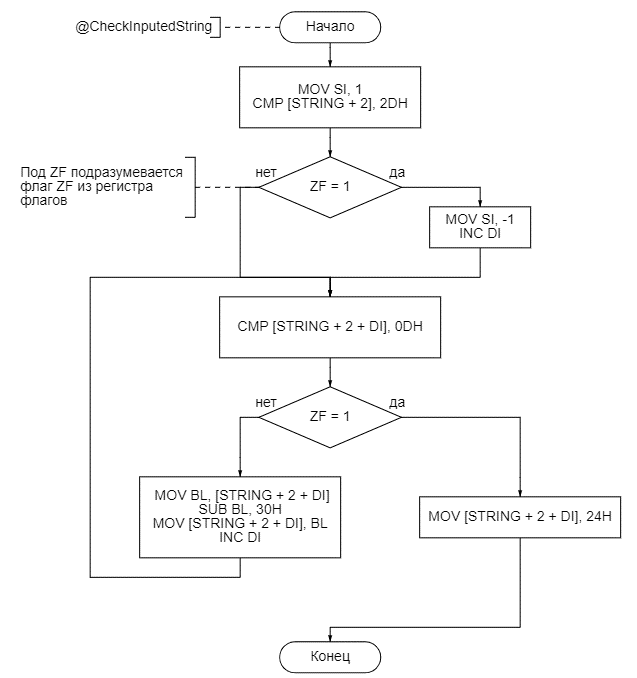
# **Блок-схема основного алгоритма программы**



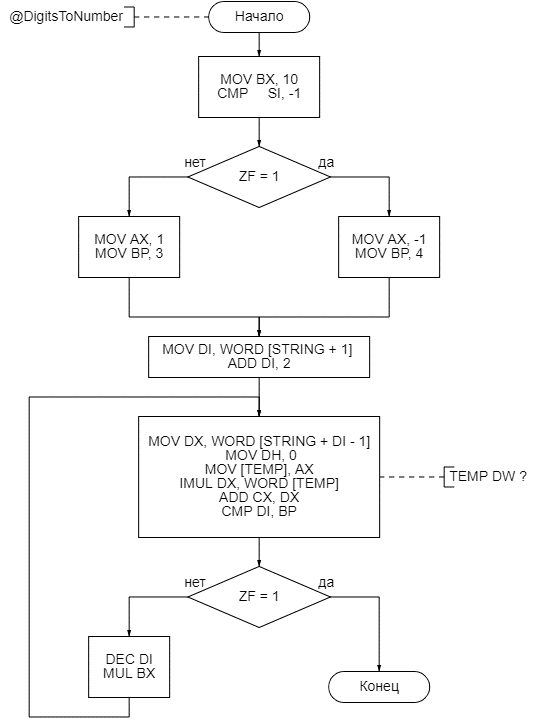
*Рисунок 1.1. Блок-схема основного алгоритма программы*



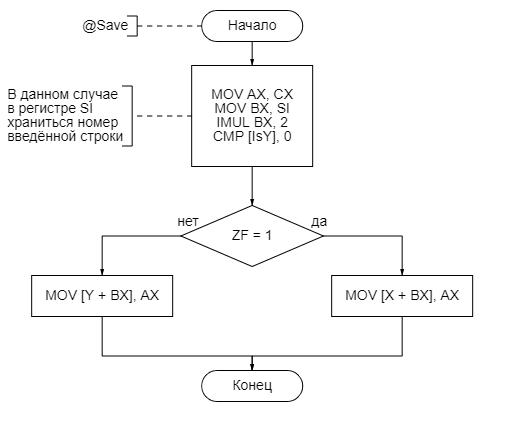
*Рисунок 1.2. Блок-схема процедуры @InputString*

**

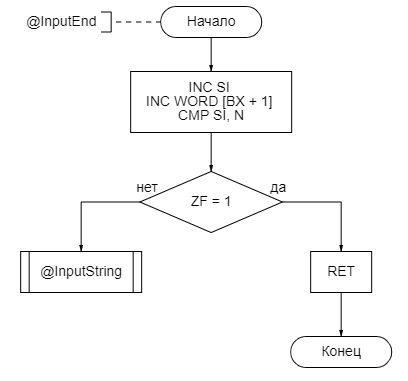
*Рисунок 1.3. Блок-схема процедуры @CheckInputedString*

**

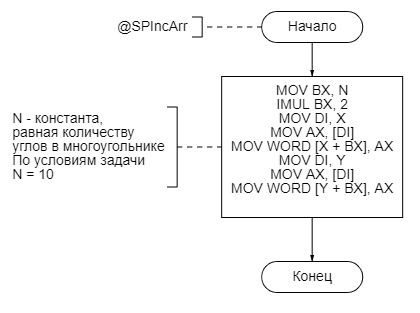
*Рисунок 1.4. Блок-схема процедуры @DigitsToNumber*



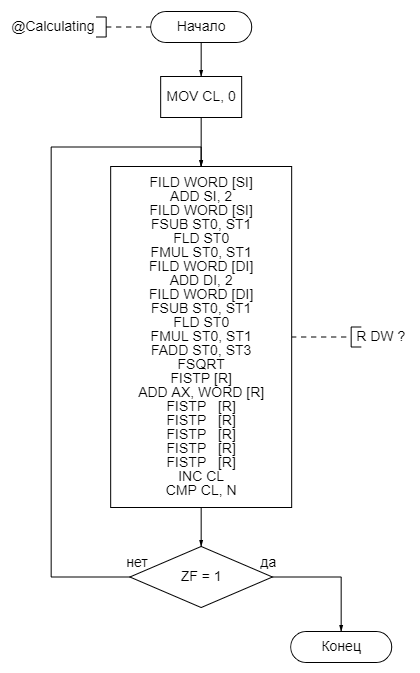
*Рисунок 1.5. Блок-схема процедуры @Save*

**

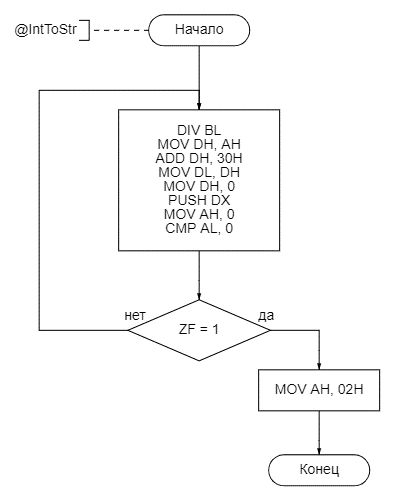
*Рисунок 1.6. Блок-схема процедуры @InputEnd*

**

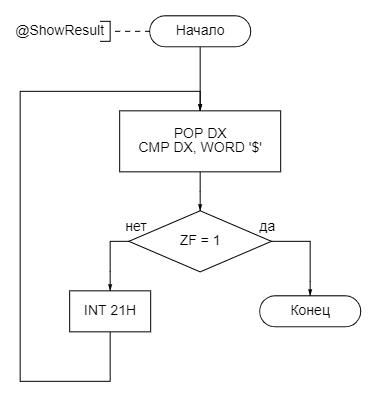
*Рисунок 1.7. Блок-схема процедуры @SPIncArr*



*Рисунок 1.8. Блок-схема процедуры @Calculating*



*Рисунок 1.9. Блок-схема процедуры @IntToStr*



*Рисунок 1.10. Блок-схема процедуры @ShowResult*

# **Руководство пользователя и описание интерфейса**

Интерфейс программы представлен в виде консоли.

**Руководство**:

Первым делом необходимо ввести последовательность координат 10 точек по оси **Х**, используя клавишу Enter. Затем необходимо совершить ввод последовательности координат тех же 10 точек, но уже по оси **Y**, всё ещё используя клавишу Enter.

В случае некорректности введённых данных, программа уведомит об ошибке ввода числа и досрочно завершит работу.

В случае корректности введённых данных, в консоли будет произведён вывод результата – периметра десятиугольника, по введённым координатам.

Для того, чтобы выйти из консольной программы необходимо нажать на Enter после завершения работы программы.

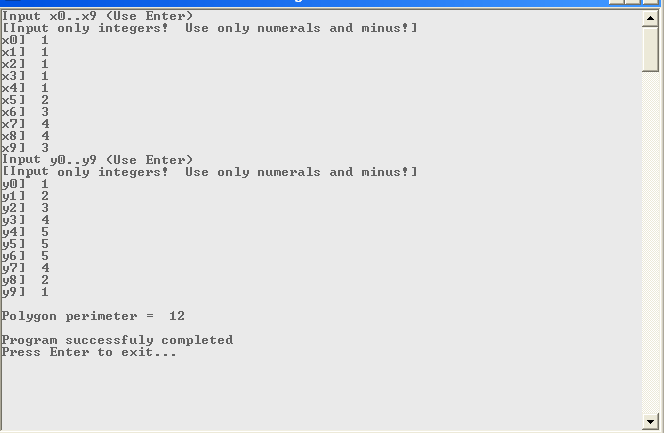


Рисунок 2.1. Пример работы программы.

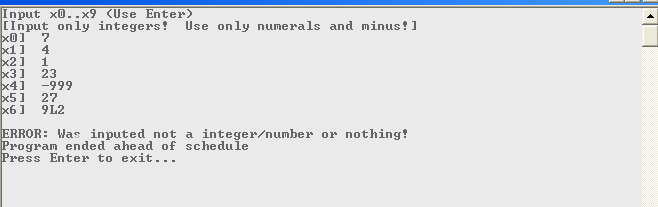


Рисунок 2.2. Пример досрочного завершения работы программы.

# **Исходный код программы**

FORMAT MZ

ORG 100H

;-----------------------------------------------

; START OF PROGRAM

;-----------------------------------------------

@Input:

; X coordinates

MOV SI, 0

MOV [IsY], 0

MOV DX, nachs1

MOV AH, 09H

INT 21H

MOV BX, inputS

CALL @InputString

; Y coordinates

MOV SI, 0

MOV [IsY], 1

MOV DX, nachs2

MOV AH, 09H

INT 21H

MOV BX, inputS

INC BYTE [BX]

MOV [BX + 1], BYTE 30H

CALL @InputString

JMP @SPIncArr

;-----------------------------------------------

; INPUT STRING (PART 1)

;-----------------------------------------------

; STRING is buffer string for string input (With 0AH)

; N - numbers of angles

; SI, DI - counters

; SI - Counter (Count from 0 to N [10 angles])

@InputString:

MOV DX, BX

MOV AH, 09H

INT 21H

; Input string

MOV DX, STRING

MOV AH, 0AH

INT 21H

MOV DX, nl

MOV AH, 09H

INT 21H

;-----------------------------------------------

; STRING CHECK & STR TO INT

;-----------------------------------------------

@CheckInputedString:

; Inputed string is in STRING + 2

; DI - Counter

MOV DI, 0

PUSH SI

MOV SI, 1

@CheckFS:

; Check first symbol

CMP [STRING + 2], 2DH

JNE @IsEmpty

@IsSign:

INC DI

MOV SI, -1

@IsEmpty:

CMP [STRING + 2 + DI], 0DH

JE @Error

CMP [STRING + 2 + DI], 24H

JE @Error

@IsNumber:

; Check the digits

CMP [STRING + 2 + DI], '0'

JE @Correct

CMP [STRING + 2 + DI], '1'

JE @Correct

CMP [STRING + 2 + DI], '2'

JE @Correct

CMP [STRING + 2 + DI], '3'

JE @Correct

CMP [STRING + 2 + DI], '4'

JE @Correct

CMP [STRING + 2 + DI], '5'

JE @Correct

CMP [STRING + 2 + DI], '6'

JE @Correct

CMP [STRING + 2 + DI], '7'

JE @Correct

CMP [STRING + 2 + DI], '8'

JE @Correct

CMP [STRING + 2 + DI], '9'

JE @Correct

; Check if it was a last symbol - CR

CMP [STRING + 2 + DI], 0DH

JE @ReplaceCR

JNE @Error

@Correct:

;-----------------------------------------------

; StrToInt: '0' - 30h = H [ASCII]

PUSH BX

MOV BL, [STRING + 2 + DI]

SUB BL, 30H

MOV [STRING + 2 + DI], BL

POP BX

;-----------------------------------------------

; Going to next symbol

INC DI

JMP @IsNumber

@ReplaceCR:

MOV [STRING + 2 + DI], 24H

;-----------------------------------------------

; DIGITS TO NUMBER

;-----------------------------------------------

@DigitsToNumber:

PUSH BX

PUSH AX

; Constant multiplier

MOV BX, 10

CMP SI, -1

JE @Sign

; 10^n

MOV AX, 1

MOV BP, 3

JMP @StartDTN

@Sign:

MOV AX, -1

MOV BP, 4

@StartDTN:

; Here will be result number

MOV CX, 0

MOV DX, 0

; Get a length

MOV DL, [STRING + 1]

; Add 2, because first 2 bytes - max L and current L

ADD DL, 2

MOV DI, DX

MOV DX, 0

@L1:

MOV DX, WORD [STRING + DI - 1]

MOV DH, 0

MOV [TEMP], AX

IMUL DX, WORD [TEMP]

; DX = Digit \* 10^n

; (n = from STRING lenght to BP)

; Result = DX + Result

ADD WORD CX, WORD DX

CMP DI, BP

JE @Save

DEC DI

MUL BX

JMP @L1

@Save:

MOV [TEMP], CX

POP AX

POP BX

POP SI

PUSH BX

MOV AX, [TEMP]

MOV BX, SI

IMUL BX, 2

CMP [IsY], 0

JE @SaveX

@SaveY:

; X[n]/Y[n] = 2 bytes. X[0] start in 1-st byte

; X[1] start in 3-rd byte

MOV [Y + BX], AX

JMP @InputEnd

@SaveX:

MOV [X + BX], AX

;-----------------------------------------------

; INPUT STRING (PART 2)

;-----------------------------------------------

@InputEnd:

POP BX

; Inc counters

INC WORD SI

INC WORD [BX + 1]

CMP SI, N

JNE @InputString

RET

;-----------------------------------------------

; ADD FIRST POINT COORDINATE TO ARRAYS END

;-----------------------------------------------

@SPIncArr:

MOV BX, N

IMUL BX, 2

MOV DI, X

MOV AX, [DI]

MOV WORD [X + BX], AX

MOV DI, Y

MOV AX, [DI]

MOV WORD [Y + BX], AX

; Show result string, because next we need to show result

MOV DX, results

MOV AH, 09H

INT 21H

;-----------------------------------------------

; CALCULATING (USING FPU)

;-----------------------------------------------

; L = SQRT((x2 - x1)^2 + (y2 - y1)^2)

MOV SI, X

MOV DI, Y

; CL is counter

MOV CL, 0

MOV AX, WORD 0

; Coordinates have to be in 2 arrays:

; 1) Array with x1..x10 Address of array beginning have to be in SI

; 2) Array with y1..y10 Address of array beginning have to be in DI

; Result will be in AX

@Calculating:

FILD WORD [SI]

ADD SI, 2

FILD WORD [SI]

FSUB ST0, ST1

FLD ST0

FMUL ST0, ST1

FILD WORD [DI]

ADD DI, 2

FILD WORD[DI]

FSUB ST0, ST1

FLD ST0

FMUL ST0, ST1

FADD ST0, ST3

FSQRT

FISTP [R]

ADD AX, WORD [R]

FISTP [R]

FISTP [R]

FISTP [R]

FISTP [R]

FISTP [R]

INC CL

CMP CL, N

JNE @Calculating

;-----------------------------------------------

; INT TO STR

;-----------------------------------------------

MOV BX, WORD 0

MOV BL, 10

MOV DX, WORD 0

PUSH WORD '$'

; Resulting string will be in stack

; And here will be console out from stack

@IntToStr:

; Result is in AX

DIV BL

; Now digit in AH/DX; Number without digit in AL/AX

MOV DH, AH

ADD DH, 30H

MOV DL, DH

MOV DH, 0

PUSH DX

MOV AH, 0

CMP AL, 0

JNE @IntToStr

MOV AH, 02H

@ShowResult:

POP DX

CMP DX, WORD '$'

JE @HappyClose

INT 21H

JMP @ShowResult

;-----------------------------------------------

; ERROR: WAS INPUTED NOT A NUMBER

;-----------------------------------------------

@Error:

MOV DX, error

MOV AH, 09H

INT 21H

JMP @Close

;-----------------------------------------------

; END OF PROGRAM

;-----------------------------------------------

@HappyClose:

MOV DX, ends

MOV AH, 09H

INT 21H

@Close:

; Read a Enter before program closing

MOV DX, el

MOV AH, 0AH

INT 21H

; Closing console

MOV AH, 4CH

INT 21H

;-----------------------------------------------

; VARIABLES

;-----------------------------------------------

; Number of angles

N = 10

; Inputed string; Max length = [1-st byte] - 1

STRING DB 5, ?, 5 DUP('$')

TEMP DW ?

X DW N + 1 DUP(?)

Y DW N + 1 DUP(?)

; R - Use for saving result

; (Length of line segment)

R DW ?

IsY DB ?

; Array for tests; P = 52

;ArrX DB 1, -1, -5, -5, -2, 1, 5, 7, 2, 2

;ArrY DB 1, 3, 1, -4, -2, -7, -4, 6, 10, 3

; Array 2 for tests; P = 12

;ArrX2 DB 1, 1, 1, 1, 1, 2, 3, 4, 4, 3

;ArrY2 DB 1, 2, 3, 4, 5, 5, 5, 4, 2, 1

;-----------------------------------------------

; STRINGS

;-----------------------------------------------

nachs1 DB 'Input x0..x', N + 47, ' (Use Enter)', 0AH, '[Input only integers! Use only numerals and minus!]', 0AH, '$'

nachs2 DB 'Input y0..y', N + 47, ' (Use Enter)', 0AH, '[Input only integers! Use only numerals and minus!]', 0AH, '$'

inputS DB 'x0] ', '$'

ends DB 0AH, 0AH, 'Program successfuly completed', 0AH, 'Press Enter to exit...' , '$'

error DB 0AH, 'ERROR: Was inputed not a integer/number or nothing!', 0AH, 'Program ended ahead of schedule', 0AH, 'Press Enter to exit...' , '$'

results DB 0AH, 'Polygon perimeter = ', '$'

nl DB 0AH, '$'

el DB 1, 2 DUP(?)

# **Заключение**

Во время выполнения задания были использованы множество команд, директив, обращений к регистрам ассемблера. Реализованы были циклы, условия разветвления, сравнения, осуществлялся переход по флагам. Был создан динамический массив, элементы которого изменялись соответственно условию задачи. Вызывалась функция DOS – INT 21H, и через неё осуществлялось управление вводом и выводом в консоль.

# Литература

* <http://av-assembler.ru/assembler.htm>
* <http://it.kgsu.ru/Assembler/>
* <https://asmforfun.blogspot.com/2009/05/blog-post_04.html>
* <http://flatassembler.narod.ru/fasm.htm#1-2-3>
* <http://natalia.appmat.ru/c&c++/assembler.html>