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                        Номоконова Александра  Олеговича

(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 схема основного алгоритма программы
  + Руководство пользователя и описание интерфейса
  + Исходный код программы в приложении
* Заключение
* Список литературы

Объем отчета

Содержание.

[**Введение.** 4](#_Toc115128381)

[**Основная часть**. 5](#_Toc115128382)

[**Условие задачи:** 5](#_Toc115128383)

[**Теоретические сведения.** 6](#_Toc115128384)

[**Описание основных процедур**. 12](#_Toc115128385)

[**Блок-схема основного алгоритма программы**. 14](#_Toc115128386)

[**Руководство пользователя и описание интерфейса**. 17](#_Toc115128387)

[**Исходный код программы.** 18](#_Toc115128388)

[**Заключение.** 35](#_Toc115128389)

[**Литература.** 35](#_Toc115128390)

**Введение.**

Освоение ассемблера способствует пониманию внутренней работы компьютерных программ на уровне машинного кода и позволяет создавать собственные компиляторы, оптимизаторы, среды исполнения, виртуальные машины, антивирусы для защиты на самом низком уровне и т.п. Несомненная польза владения ассемблером помогает программисту на протяжении всей его деятельности.

**Основная часть**.

**Условие задачи:**

Даны целые числа a0, …, an; b1, …, bm; k. Если в последовательностиa1,…,anнет ни одного члена со значением k, то первый по порядку член этой последовательности, не меньший всех остальных членов, заменить на значение k. По такому же правилу преобразить последовательность b1, …, bmприменительно к значению 10.

**Теоретические сведения.**

**Процедура (подпрограмма)** — это основная функциональная единица декомпозиции (разделения на несколько частей) некоторой задачи. Процедура представляет собой группу команд для решения конкретной подзадачи и обладает средствами получения управления из точки вызова задачи более высокого приоритета и возврата управления в эту точку. 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:

PROCИмяПроцедуры   
; тело процедуры

ret  
ENDP

call **ИмяПроцедуры**— вызов процедуры (подпрограммы).

Внутри процедуры обращаться к параметрам можно как к простым переменным — с помощью квадратных скобок! При вызове процедуры параметры должны помещаться в стек, начиная с последнего. Если запустить программу в отладчике, то можно увидеть сгенерированный макросами код. Макросы создали нужный пролог и эпилог процедуры. Команда [RET](https://fasmworld.ru/spravochnik-komand/ret/) дополнительно выталкивает 6 байт из стека

**Циклом** называется повторяющееся выполнение последовательности команд.

Для организации цикла предназначена команда **LOOP**. У этой команды один операнд — имя метки, на которую осуществляется переход. В качестве счётчика цикла используется регистр **CX**. Команда LOOP выполняет декремент **CX**, а затем проверяет его значение. Если содержимое CX не равно нулю, то осуществляется переход на метку, иначе управление переходит к следующей после LOOP команде.

Содержимое CX интерпретируется командой как положительное число. В CX нужно помещать число, равное требуемому количеству повторений цикла. Метка должна находиться в диапазоне -127…+128 байт от команды LOOP (иначе будет ошибка).

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. Например:

m1: mov ax,4C00h

int 21h

Теперь вместо имени m1 компилятор везде будет подставлять адрес команды mov ax,4C00h. 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.

**Логические команды.**

**Команда AND.** Команда AND выполняет логическое умножение двух операндов - ol и о2. Результат сохраняется в операнде ol. Операнды могут быть 8-, 16- или 32-битными регистрами, адресами памяти или непосредственными значениями.

AND ol, o2

Следующий пример вычисляет логическое И логической единицы и логического нуля (1 AND 0).

mov al, 1

mov bl, 0

and al, bl ; AL = 0b00000001 and 0b00000000 = 0b00000000

**Команда OR**. Команда OR выполняет логическое сложение двух операндов - ol и о2. Результат сохраняется в операнде ol. Типы операндов такие же, как у команды AND.

OR ol, o2

Рассмотрим простой пример установки наименее значимого бита (первый справа) переменной mask в 1.

orbyte [mask], 1;mask = 0bxxxxxxx1

**Команда XOR.**Вычисляет так называемое «исключающее ИЛИ» операндов ol и о2. Результат сохраняется в о1. Типы операндов такие же, как у предыдущих инструкций. Формат команды:

XOR ol, о2

Исключающее ИЛИ обратимо: выражение ((х хог у) хог у) снова возвратит х.

mov al, 0x55; AL = 0x55

хог al, 0xAA;

AL = AL xorOxAAxoral, 0xAA ;возвращаем в AL исходное значение - 0x55

**Команда NOT**. Используется для инверсии отдельных битов единственного операнда, который может быть регистром или памятью. Соответственно команда может быть записана в трех различных форматах:

NOT r/m8

NOT r/ml6

NOT r/m32

Следующий пример демонстрирует различие между операциями NOT и NEG:

mov al, 0b00000010 ;AL = 2

mov bl, al ;BL = 2

notal ;после этой операции мы получим 11111101b = OxFD (-3)

negbl ;a после этой операции результат будет другим 11111110 = OxFE (-2)

**Команда SUB.** Инструкция SUB тоже довольно проста для понимания. Если [**инструкция ADD**](http://av-assembler.ru/instructions/add.php) выполняет сложение, то команда SUB процессоров семейства i80x86 используется для вычитания. Синтаксис команды SUB такой:

SUB РАЗНОСТЬ, ЧИСЛО

С помощью этой команды можно из РАЗНОСТИ вычесть ЧИСЛО. Результат помещается в РАЗНОСТЬ.

**Команда XCHG** в Ассемблере выполняет перестановку операндов (то есть меняет местами значения операндов). Синтаксис:

XCHG ОПЕРАНД1, ОПЕРАНД2

После выполнения команды значение из ОПЕРАНД1 будет помещено в ОПЕРАНД2, а из ОПЕРАНД2 - в ОПЕРАНД1.

**Умножение.**Для умножения чисел без знака предназначена команда **MUL**. У этой команды только один операнд — второй множитель, который должен находиться в регистре или в памяти. Местоположение первого множителя и результата задаётся неявно и зависит от размера операнда

**Стек.**Стеком называется структура данных, организованная по принципу **LIFO** («Last In — First Out» или «последним пришёл — первым ушёл»). Стек является неотъемлемой частью архитектуры процессора и поддерживается на аппаратном уровне: в процессоре есть специальные регистры (SS, BP, SP) и команды для работы со стеком.

Стек используется для сохранения содержимого регистров, используемых программой, перед вызовом подпрограммы, которая, в свою очередь, будет использовать регистры процессора “в своих личных целях”.

Стек располагается в оперативной памяти в сегменте стека, и поэтому адресуется относительно сегментного регистра SS. Шириной стека называется размер элементов, которые можно помещать в него или извлекать. Для стека существуют две основные операции:

* PUSH – добавление элемента на вершину стека;
* POP – извлечение элемента с вершины стека.

**Регистр флагов** – это очень важный [регистр](http://av-assembler.ru/asm/afd/asm-cpu-registers.htm) процессора, который используется при выполнении большинства команд. Регистр флагов носит название **EFLAGS**. Это 32-разрядный регистр. Однако старшие 16 разрядов используются при работе в защищённом режиме, и пока мы их рассматривать не будем. К младшим 16 разрядам этого регистра можно обращаться как к отдельному регистру с именем FLAGS. Именно этот регистр мы и рассмотрим в этом разделе.

Каждый бит в регистре **FLAGS** является флагом. **Флаг** – это один или несколько битов памяти, которые могут принимать двоичные значения (или комбинации значений) и характеризуют состояние какого-либо объекта. Обычно флаг может принимать одно из двух логических значений. Поскольку в нашем случае речь идёт о бите, то каждый флаг в регистре может принимать либо значение 0, либо значение 1. Флаги устанавливаются в 1 при определённых условиях, или установка флага в 1 изменяет поведение процессора.

[**Макросы PROC и ENDP**](https://fasmworld.ru/uchebnyj-kurs/029-makrosy-proc-i-endp/).

Для начала, нам потребуется заголовочный файл с макросами. Стандартный пакет FASM для Windows, к сожалению, не включает в себя макросы для 16-битных процедур. Однако такие макросы можно найти на официальном форуме FASM. Это переделанная версия файла PROC32.INC с точно таким же синтаксисом.

Заголовочный файл необходимо будет включить в программу с помощью директивы include:

|  |  |
| --- | --- |
| 1 | include'proc16.inc' |

После proc указывается имя процедуры. Далее через запятую список параметров. Между именем процедуры и списком параметров запятую ставить не обязательно (можно просто поставить пробел). Для возврата из процедуры следует использовать команду [RET](https://fasmworld.ru/spravochnik-komand/ret/) без операндов. Завершается процедура макросом endp. Например, объявим процедуру с тремя параметрами:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 8-битный операнд, который задает номер нужного прерывания.  INT o1  Возврат из обработчика прерывания осуществляется с помощью команды  IRET, которая восстанавливает исходные значения (E)IP, CS и флагов из стека. Формат команды. **Описание и инициализация массива в программе** Специальных средств описания массивов в программах ассемблера, конечно, 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 4 байта:  masdd 1,2,3,4,5 |  1. Используя оператор повторения **dup**. К примеру:  |  | | --- | | ;массив из 5 нулевых элементов.  ;Размер каждого элемента 2 байта:  masdw 5 dup (0) |   Такой способ определения используется для резервирования памяти с целью размещения и инициализации элементов массива.   1. Используя директивы **label** и **rept**. Пара этих директив может облегчить описание больших массивов в памяти и повысить наглядность такого описания. Директива **rept** относится к макросредствам языка ассемблера и вызывает повторение указанное число раз строк, заключенных между директивой и строкой endm. К примеру, определим массив байт в области памяти, обозначенной идентификатором *mas\_b*. В данном случае директива **label** определяет символическое имя *mas\_b*, аналогично тому, как это делают директивы резервирования и инициализации памяти. Достоинство директивы **label** в том, что она не резервирует память, а лишь определяет характеристики объекта. В данном случае объект — это ячейка памяти. Используя несколько директив **label**, записанных одна за другой, можно присвоить одной и той же области памяти разные имена и разный тип, что и сделано в следующем фрагменте:  |  | | --- | | ...  n=0  ...  mas\_b label byte  mas\_w label word  rept 4  dw 0f1f0h  endm |   В результате в памяти будет создана последовательность из четырех слов **f1f0**. Эту последовательность можно трактовать как массив байт или слов в зависимости от того, какое имя области мы будем использовать в программе — *mas\_b* или *mas\_w*.   1. Использование цикла для инициализации значениями области памяти, которую можно будет впоследствии трактовать как массив.  **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:  **база + (индекс\*размер элемента)**   * ***ндексная адресация со смещением*** — режим адресации, при котором эффективный адрес формируется из двух компонентов:   + **постоянного (базового)** — указанием прямого адреса массива в виде имени идентификатора, обозначающего начало массива;   + **переменного (индексного)** — указанием имени индексного регистра.   + К примеру:  |  | | --- | | masdw 0,1,2,3,4,5  ...  mov si,4  ;поместить 3-й элемент массива mas в регистр ax:  movax,mas[si] |  * ***базовая индексная адресация со смещением*** — режим адресации, при котором эффективный адрес формируется максимум из трех компонентов:   + **постоянного** (необязательный компонент), в качестве которой может выступать прямой адрес массива в виде имени идентификатора, обозначающего начало массива, или непосредственное значение;   + **переменного (базового)** — указанием имени базового регистра;   + **переменного (индексного)** — указанием имени индексного регистра. |
|  |

**Описание основных процедур**.

; Процедура ввода строки c консоли

; вход: AL - максимальная длина (с символом CR) (1-254)

; выход: AL - длина введённой строки (не считая символа CR)

; DX - адрес строки, заканчивающейся символом CR(0Dh)

proc input\_str uses cx

mov cx,ax ; Сохранение AX в CX

mov ah,0Ah ; Функция DOS 0Ah - ввод строки в буфер

mov [buffer],al ; Запись максимальной длины в первый байт буфера

movbyte[buffer+1],0 ; Обнуление второго байта (фактической длины)

mov dx,buffer ; DX = aдресбуфера

int 21h ; Обращение к функции DOS

moval,[buffer+1] ; AL = длина введённой строки

add dx,2 ; DX = адрес строки

movah,ch ; Восстановление AH

ret

endp

; Процедура преобразования десятичной строки в слово без знака

; вход: AL - длина строки

; DX - адрес строки, заканчивающейся символом CR(0Dh)

; выход: AX - слово (в случае ошибки AX = 0)

; CF = 1 - ошибка

proc str\_to\_udec\_word uses cx dx bx si di

movsi,dx ; SI = адрес строки

mov di,10 ; DI = множитель 10 (основание системы счисления)

movzxcx,al ; CX = счётчик цикла = длина строки

jcxzstudw\_error ; Если длина = 0, возвращаем ошибку

xorax,ax ; AX = 0

xorbx,bx ; BX = 0

studw\_lp:

movbl,[si] ; Загрузка в BL очередного символа строки

incsi

cmp bl,'0' ; Если код символа меньше кода "0"

jlstudw\_error ; возвращаем ошибку

cmp bl,'9' ; если код символа больше кода "9"

jgstudw\_error ; возвращаем ошибку

sub bl,'0' ; Преобразование символа-цифры в число

muldi ; AX = AX \* 10

jcstudw\_error ; Если результат больше 16 бит - ошибка

addax,bx ; Прибавляем цифру

jcstudw\_error ; Если переполнение - ошибка

loopstudw\_lp ; Команда цикла

jmpstudw\_exit ; Успешное завершение (здесь всегда CF = 0)

studw\_error:

xorax,ax ; AX = 0

stc ; CF = 1 (Возвращаем ошибку)

studw\_exit: ; завершение функции

ret

endp

; Процедура вывода строки на консоль

; DI - адрес строки

proc print\_str uses ax

mov ah,9 ; Функция DOS 09h - вывод строки

xchg dx,di ; Обмен значениями DX и DI

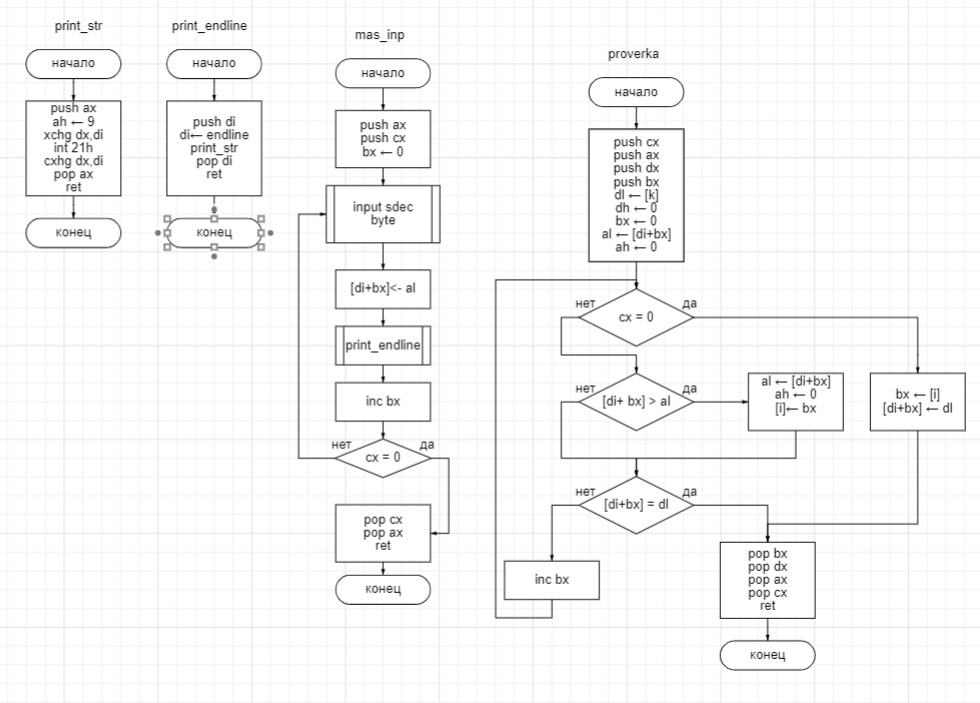
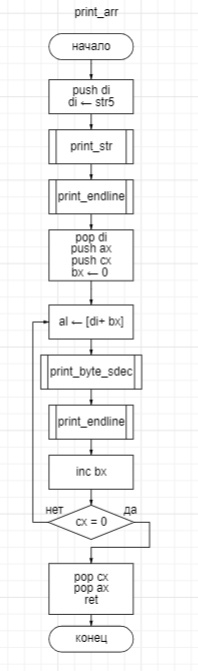
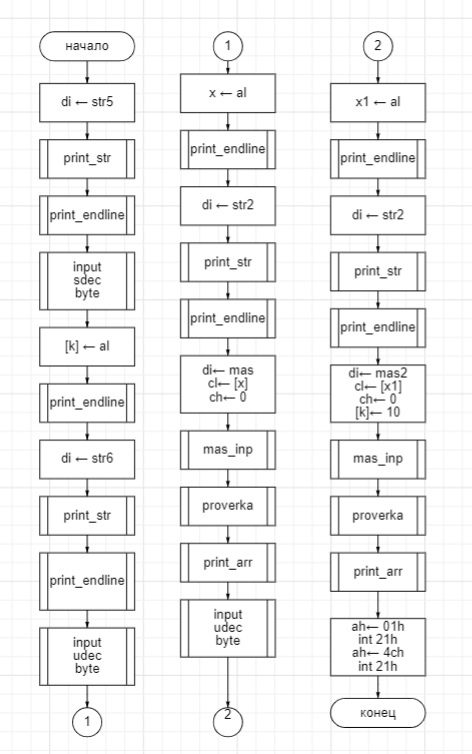
int 21h ; Обращение к функции DOS

xchg dx,di ; Обмен значениями DX и DI

ret

endp

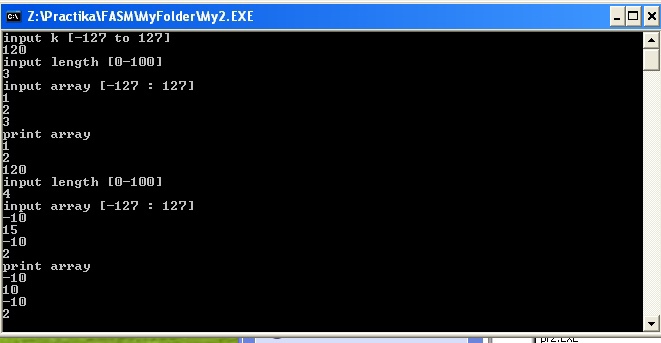
**Блок-схема основного алгоритма программы**.



**Руководство пользователя и описание интерфейса**.

Интерфейс программы представлен в виде консоли.

Руководство:

Сначала вводим число, с которым будем сравнивать первый массив, затем заполняем длину массива от 0 до 100, а также заполняем массив числами от -127 до 127. Выводим первый измененный массив. Со вторым массивом делаем то же самое, но сравниваем с 10.

**Исходный код программы.**

include 'proc16.inc'

format MZ ;Определяем тип выходного файла (ехе)

org 100h ;Переходим на адрес 100h сегмента кода программы

;Вывод строки

movdi, str5 ; помещаем в регистр строку "input n [-127 to127]$"

callprint\_str ; вызываем функцую вывода строки

callprint\_endline ; вызываем функую перехода на следующую строку

;Вводим число, с которым будем сравнивать элементы массивов

callinput\_udec\_byte ; вызов функции ввода беззнакового числа с консоли

mov [k], al

call print\_endline

movdi, str6 ; помешаем в регистр строку "inputlength [0-100]$"

call print\_str

call print\_endline

;Ввод длины первого массива

callinput\_udec\_byte

mov [x], al

call print\_endline

;Вывод строки

movdi, str2 ; помещаем в регистр строку "inputarray $"

call print\_str

call print\_endline

;передача массива в процедуру

movdi, mas ; di = адрес первого эл. массива

movcl, [x] ; помещаем размер массива в регистр cl

movch, 0

callmas\_inp ; заполнение массива

callproverka ; проверка условия и редактирование массива

callprint\_arr ; вывод массива на экран

;Вывод строки

movdi, str6 ; помешаем в регистр строку "inputlength [0-100]$"

call print\_str

call print\_endline

;Ввод длины второго массива

callinput\_udec\_byte

mov [x1], al

call print\_endline

;Вывод строки

movdi, str2 ; помещаем в регистр строку "inputarray $"

call print\_str

call print\_endline

;Передача второго массива в процедуру и его размера

mov di, mas2

mov cl, [x1]

movch, 0

callmas\_inp ; заполнение массива

mov [k],10 ; сравиниваемое число для 2 массива

callproverka ; проверка условия и редактирование массива

callprint\_arr ; вывод массива на экран

;Ввод символа с консоли, для поддержания программы в состоянии ожидания

movah, 01h

int 21h

;Завершение работы

mov ah, 4ch

int 21h

;----------------------------Данные-----------------

str2 db "input array [-127 to 127] $"

str4 db "print array $"

str5 db "input k [-127 to 127]$"

str6 db "input length [0-100]$"

x db 0

x1 db 0

k db 0

i dw 0

masdb 100 dup(?) ;объявление массива длиной 100 элементов размером в 1 байт неопределенного значения

mas2 db 100 dup(?)

endline db 13,10,'$'

buffer rb 7

;----------------------------Данные-----------------

str2 db "input array [-127 to 127] $"

str4 db "print array $"

str5 db "input k [-127 to 127]$"

str6 db "input length [0-100]$"

x db 0

x1 db 0

k db 0

i dw 0

masdb 100 dup(?) ;объявление массива длиной 100 элементов размером в 1 байт неопределенного значения

mas2 db 100 dup(?)

endline db 13,10,'$'

bufferrb 7

; ------------------- Процедуры ------------------------

; заполнение массива

; вход: di - адрес первого эллемента массива

; выход: заполненный массив. Доступ к массиву по его адресу (имени) либо через di

proc mas\_inp uses ax di cx

push ax

pushcx ;сохраняем размер массива в стеке

mov bx,0 ;счетчик цикла

c1:

callinput\_sdec\_byte

mov [di+bx],al ;заполнение элемента массива

callprint\_endline

incbx

loop c1 ;выполняется пока cx не станет 0

popcx ;извлекаем размер массива из стека

popax

ret ;вохврат из процедуры

endp ;конец процедуры

; изменение массива

; вход: di - адрес первого эллемента массива

; выход: изменённый массив. Доступ к массиву по его адресу (имени) либо через di

proc proverka uses ax cx di dx

push cx ; сохраняем значения в стеке

push ax

push dx

push bx

movdl,[k] ;помещаем сравниваемое число в регистр

movdh, 0

movbx, 0 ;счетчик

moval,[di+bx] ;al - максимальный элемент массива

movah, 0

if1:

cmp [di+bx], al ;проверка условия

jgsetmax ;еслиmas[si] >ax, переход на метку setmax

if2:

cmp [di+bx], dl ;сравнивание с n

jeexit ;еслиmas[si] = n, тогда выходим из цикла

incbx ;увеличение индекса на 1

loop if1 ;eslimas[si] != n, начать цикл снова, пока не пройдет по всем элементам массива

movbx,[i] ;индекс максимального элемента массива

mov [di+bx], dl ;если в массиве не было n, заменить максимальный элемент на n

jmpexit ;выход из цикла

setmax: ;Новый максимальный элемент массива

moval, [di+bx] ;сохраняет новый максимальный элемент

movah, 0

mov [i], bx ;запоминаем индекс максимального элемента si- 16 bit

jmp if2 ;продолжить цикл, переход ко 2 условию

exit:

popbx ;извлекаем значения из массива

pop dx

pop ax

pop cx

ret

endp

; вывод массива

; вход: di - адрес первого эллемента массива

; выход: ничего не изменилось. Доступ к массиву по его адресу (имени) либо через di

proc print\_arr uses ax cx di

pushdi ;сохраняем значение в стеке

movdi, str4 ;помещаем в регистр строку "printarray $"

call print\_str

call print\_endline

pop di

pushax ; сохраняем значения

push cx

mov bx,0 ;счетчик

pr1:

moval, [di+bx] ;помещаем значение массива в региср

callprint\_byte\_sdec ;вызов функции вывода беззнакового числа

call print\_endline

inc bx ;увеличиваеминдекс

loop pr1 ;проход по всему массиву

popcx ;извлекаем значения

popax

ret

endp

/\*---input---\*/

/\*---------------------------------------------------------------------------------------\*/

; Процедура ввода строки c консоли

; вход: AL - максимальная длина (с символом CR) (1-254)

; выход: AL - длина введённой строки (не считая символа CR)

; DX - адрес строки, заканчивающейся символом CR(0Dh)

proc input\_str uses cx

mov cx,ax ; Сохранение AX в CX

mov ah,0Ah ; Функция DOS 0Ah - ввод строки в буфер

mov [buffer],al ; Запись максимальной длины в первый байт буфера

movbyte[buffer+1],0 ; Обнуление второго байта (фактической длины)

mov dx,buffer ; DX = aдресбуфера

int 21h ; Обращение к функции DOS

moval,[buffer+1] ; AL = длина введённой строки

add dx,2 ; DX = адрес строки

movah,ch ; Восстановление AH

ret

endp

; Процедура преобразования десятичной строки в слово без знака

; вход: AL - длина строки

; DX - адрес строки, заканчивающейся символом CR(0Dh)

; выход: AX - слово (в случае ошибки AX = 0)

; CF = 1 - ошибка

proc str\_to\_udec\_word uses cx dx bx si di

movsi,dx ; SI = адрес строки

mov di,10 ; DI = множитель 10 (основание системы счисления)

movzxcx,al ; CX = счётчик цикла = длина строки

jcxzstudw\_error ; Если длина = 0, возвращаем ошибку

xorax,ax ; AX = 0

xorbx,bx ; BX = 0

studw\_lp:

movbl,[si] ; Загрузка в BL очередного символа строки

incsi

cmp bl,'0' ; Если код символа меньше кода "0"

jlstudw\_error ; возвращаем ошибку

cmp bl,'9' ; если код символа больше кода "9"

jgstudw\_error ; возвращаем ошибку

sub bl,'0' ; Преобразование символа-цифры в число

muldi ; AX = AX \* 10

jcstudw\_error ; Если результат больше 16 бит - ошибка

addax,bx ; Прибавляем цифру

jcstudw\_error ; Если переполнение - ошибка

loopstudw\_lp ; Команда цикла

jmpstudw\_exit ; Успешное завершение (здесь всегда CF = 0)

studw\_error:

xorax,ax ; AX = 0

stc ; CF = 1 (Возвращаем ошибку)

studw\_exit: ; завершение функции

ret

endp

; Процедура преобразования десятичной строки в байт без знака

; вход: AL - длина строки

; DX - адрес строки, заканчивающейся символом CR(0Dh)

; выход: AL - байт (в случае ошибки AL = 0)

; CF = 1 - ошибка

proc str\_to\_udec\_byte uses dx

pushax

stdcallstr\_to\_udec\_word ; Преобразование строки в слово (без знака)

jcstudb\_exit ; Если ошибка, то возвращаем ошибку

testah,ah ; Проверка старшего байта AX

jzstudb\_exit ; Если 0, то выход из процедуры (здесь всегда CF = 0)

xoral,al ; Обнуление AL

stc ; CF = 1 (Возвращаем ошибку)

studb\_exit:

popdx

movah,dh ; Восстановление только старшей части AX

ret

endp

; Процедура ввода слова с консоли в десятичном виде (без знака)

; выход: AX - слово (в случае ошибки AX = 0)

; CF = 1 - ошибка

proc input\_udec\_word uses dx

mov al,6 ; Ввод максимум 5 символов (65535) + конец строки

stdcallinput\_str ; Вызов процедуры ввода строки

stdcallstr\_to\_udec\_word ; Преобразование строки в слово (без знака)

ret

endp

; Процедура ввода байта с консоли в десятичном виде (без знака)

; выход: AL - байт (в случае ошибки AL = 0)

; CF = 1 - ошибка

proc input\_udec\_byte uses dx

mov al,5 ; Ввод максимум 4 символов (255) + конец строки

stdcallinput\_str ; Вызов процедуры ввода строки

stdcallstr\_to\_udec\_byte ; Преобразование строки в байт (без знака)

ret

endp

; Процедура преобразования десятичной строки в слово со знаком

; вход: AL - длина строки

; DX - адрес строки, заканчивающейся символом CR(0Dh)

; выход: AX - слово (в случае ошибки AX = 0)

; CF = 1 - ошибка

proc str\_to\_sdec\_word uses bx dx

testal,al ; Проверка длины строки

jzstsdw\_error ; Если равно 0, возвращаем ошибку

movbx,dx ; BX = адрес строки

movbl,[bx] ; BL = первый символ строки

cmpbl,'-' ; Сравнение первого символа с "-"

jnestsdw\_no\_sign ; Если не равно, то преобразуем как число без знака

inc dx

dec al

stsdw\_no\_sign:

stdcallstr\_to\_udec\_word ; Преобразуем строку в слово без знака

jcstsdw\_exit ; Если ошибка, то возвращаем ошибку

cmpbl,'-' ; Снова проверяем знак

jnestsdw\_plus ; Если первый символ не '-', то число положительное

cmp ax,32768 ; Модуль отрицательного числа должен быть не больше 32768

jastsdw\_error ; Если больше (без знака), возвращаем ошибку

negax ; Инвертируем число

jmpstsdw\_ok ; Переход к нормальному завершению процедуры

stsdw\_plus:

cmp ax,32767 ; Положительное число должно быть не больше 32767

jastsdw\_error ; Если больше (без знака), возвращаем ошибку

stsdw\_ok:

clc ; CF = 0

jmpstsdw\_exit ; Переход к выходу из процедуры

stsdw\_error:

xorax,ax ; Обнуление AX

stc ; CF = 1 (Возвращаем ошибку

stsdw\_exit:

ret

endp

; Процедура преобразования десятичной строки в байт со знаком

; вход: AL - длина строки

; DX - адрес строки, заканчивающейся символом CR(0Dh)

; выход: AL - байт (в случае ошибки AL = 0)

; CF = 1 - ошибка

proc str\_to\_sdec\_byte uses dx

pushax

stdcallstr\_to\_sdec\_word ; Преобразование строки в слово (со знаком)

jcstsdb\_exit ; Если ошибка, то возвращаем ошибку

cmp ax,127 ; Сравнение результата с 127

jgstsdb\_error ; Если больше - ошибка

cmpax,-128 ; Сравнение результата с -128

jlstsdb\_error ; Если меньше - ошибка

clc ; CF = 0

jmpstsdb\_exit ; Переход к выходу из процедуры

stsdb\_error:

xoral,al ; Обнуление

stc ; CF = 1 (Возвращаемошибку)

stsdb\_exit:

popdx

movah,dh ; Восстановление только старшей части AX

ret

endp

; Процедура ввода слова с консоли в десятичном виде (со знаком)

; выход: AX - слово (в случае ошибки AX = 0)

; CF = 1 - ошибка

proc input\_sdec\_word uses dx

mov al,7 ; Ввод максимум 7 символов (-32768) + конец строки

stdcallinput\_str ; Вызов процедуры ввода строки

stdcallstr\_to\_sdec\_word ; Преобразование строки в слово (со знаком)

ret

endp

; Процедура ввода байта с консоли в десятичном виде (со знаком)

; выход: AL - байт (в случае ошибки AL = 0)

; CF = 1 - ошибка

proc input\_sdec\_byte uses dx

mov al,5 ; Ввод максимум 4 символов (-128) + конец строки

stdcallinput\_str ; Вызов процедуры ввода строки

stdcallstr\_to\_sdec\_byte ; Преобразование строки в байт (со знаком)

ret

endp

/\*---output---\*/

/\*---------------------------------------------------------------------------------------\*/

; Процедура вывода байта на консоль в десятичном виде (без знака)

; AL - байт

proc print\_byte\_udec uses di

mov di,buffer ; DI = адресбуфера

pushdi ; Сохранение DI в стеке

stdcallbyte\_to\_udec\_str ; Преобразование байта в AL в строку

movbyte[di],'$' ; Добавление символа конца строки

popdi ; DI = адрес начала строки

stdcallprint\_str ; Вывод строки на консоль

ret

endp

; Процедура вывода слова на консоль в десятичном виде (без знака)

; AX - слово

proc print\_word\_udec uses di

mov di,buffer ; DI = адресбуфера

pushdi ; Сохранение DI в стеке

stdcallword\_to\_udec\_str ; Преобразование слова в AX в строку

movbyte[di],'$' ; Добавление символа конца строки

popdi ; DI = адрес начала строки

stdcallprint\_str ; Вывод строки на консоль

ret

endp

; Процедура вывода байта на консоль в десятичном виде (со знаком)

; AL - байт

proc print\_byte\_sdec uses di

mov di,buffer ; DI = адресбуфера

pushdi ; Сохранение DI в стеке

stdcallbyte\_to\_sdec\_str ; Преобразование байта в AL в строку

movbyte[di],'$' ; Добавление символа конца строки

popdi ; DI = адрес начала строки

stdcallprint\_str ; Вывод строки на консоль

ret

endp

; Процедура вывода слова на консоль в десятичном виде (со знаком)

; AX - слово

proc print\_word\_sdec uses di

mov di,buffer ; DI = адресбуфера

pushdi ; Сохранение DI в стеке

stdcallword\_to\_sdec\_str ; Преобразование слова в AX в строку

movbyte[di],'$' ; Добавление символа конца строки

popdi ; DI = адрес начала строки

stdcallprint\_str ; Вывод строки на консоль

ret

endp

; Процедура вывода строки на консоль

; DI - адресстроки

proc print\_str uses ax

mov ah,9 ; Функция DOS 09h - вывод строки

xchgdx,di ; Обмен значениями DX и DI

int 21h ; Обращение к функции DOS

xchgdx,di ; Обмен значениями DX и DI

ret

endp

; Процедура вывода конца строки (CR+LF)

proc print\_endline uses di

mov di,endline ; DI = адресстрокиссимволамиCR,LF

stdcallprint\_str ; Вывод строки на консоль

ret

endp

; Процедура преобразования байта в строку в десятичном виде (без знака)

; AL - байт.

; DI - буфер для строки (3 символа). Значение регистра не сохраняется.

procbyte\_to\_udec\_strusesax

xorah,ah ; Преобразование байта в слово (без знака)

stdcallword\_to\_udec\_str ; Вызов процедуры для слова без знака

ret

endp

; Процедура преобразования слова в строку в десятичном виде (без знака)

; AX - слово

; DI - буфер для строки (5 символов). Значение регистра не сохраняется.

procword\_to\_udec\_strusesaxcxdxbx

xorcx,cx ; Обнуление CX

mov bx,10 ; В BX делитель (10 для десятичной системы)

wtuds\_lp1: ; Цикл получения остатков от деления

xordx,dx ; Обнуление старшей части двойного слова

divbx ; Деление AX=(DX:AX)/BX,остаток в DX

add dl,'0' ; Преобразование остатка в код символа

pushdx ; Сохранение в стеке

inccx ; Увеличение счетчика символов

testax,ax ; Проверка AX

jnz wtuds\_lp1 ; Переход к началу цикла,если частное не 0.

wtuds\_lp2: ; Цикл извлечения символов из стека

popdx ; Восстановление символа из стека

mov [di],dl ; Сохранение символа в буфере

incdi ; Инкремент адреса буфера

loop wtuds\_lp2 ; Команда цикла

ret

endp

; Процедура преобразования байта в строку в десятичном виде (со знаком)

; AL - байт.

; DI - буфер для строки (4 символа). Значение регистра не сохраняется.

procbyte\_to\_sdec\_strusesax

movsxax,al ; Преобразование байта в слово (со знаком)

stdcallword\_to\_sdec\_str ; Вызов процедуры для слова со знаком

ret

endp

; Процедура преобразования слова в строку в десятичном виде (со знаком)

; AX - слово

; DI - буфер для строки (6 символов). Значение регистра не сохраняется.

proc word\_to\_sdec\_str uses ax

testax,ax ; Проверка знака AX

jnswtsds\_no\_sign ; Если >= 0,преобразуем как беззнаковое

movbyte[di],'-' ; Добавление знака в начало строки

incdi ; Инкремент DI

negax ; Изменение знака значения AX

wtsds\_no\_sign:

stdcallword\_to\_udec\_str ; Преобразование беззнакового значения

ret

endp

**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.**

[**https://works.doklad.ru/view/APAuPlrI48Q.html**](https://works.doklad.ru/view/APAuPlrI48Q.html)

[**https://pro-prof.com/forums/topic/asm\_i8086\_19**](https://pro-prof.com/forums/topic/asm_i8086_19)

[**https://prog-cpp.ru/asm-proc/**](https://prog-cpp.ru/asm-proc/)

[**https://www.i-assembler.ru/25/Text/Data.htm**](https://www.i-assembler.ru/25/Text/Data.htm)