

Министерство науки и высшего образования Российской Федерации

Калужский филиал
федерального государственного бюджетного образовательного
учреждения высшего образования
**«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»**
(КФ МГТУ им. Н.Э. Баумана)

К. А. АМЕЛИЧЕВА

ОБРАБОТКА ДВУХМЕРНЫХ МАССИВОВ ЦЕЛЫХ ЧИСЕЛ

Методические указания к выполнению домашней работы
по курсу «Машинно-зависимые языки программирования»

Калуга - 2019

УДК 004.4 424

ББК 32.973.3

Методические указания к выполнению домашней работы по курсу «Машинно-зависимые языки программирования». — М.: Издательство МГТУ им. Н.Э. Баумана, 2019. — 47 с.

Методические указания к выполнению домашней работы, в рамках самостоятельной работы студентов, по дисциплине «Машинно-зависимые языки программирования» содержат описание выполнения домашнего задания: теоретическую часть, практическую часть, индивидуальные варианты и методические рекомендации к их выполнению.

Предназначены для студентов 2-го курса бакалавриата КФ МГТУ им. Н.Э. Баумана, обучающихся по направлению подготовки 09.03.04 «Программная инженерия».

ОГЛАВЛЕНИЕ

ОГЛАВЛЕНИЕ	3
ВВЕДЕНИЕ	4
ЦЕЛЬ И ЗАДАЧИ РАБОТЫ, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕЕ ВЫПОЛНЕНИЯ.....	5
ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	6
ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ.....	36
ВАРИАНТЫ ЗАДАНИЙ.....	37
ФОРМА ОТЧЕТА ПО ДОМАШНЕЙ РАБОТЕ.....	40
КОНТРОЛЬНЫЕ ВОПРОСЫ:.....	40
ЛИТЕРАТУРА	41

ВВЕДЕНИЕ

В Ассемблере только программист с помощью составленного им алгоритма обработки определяет, как нужно трактовать последовательность байтов, составляющих массив. Так как массивом считается непрерывный участок памяти, в котором друг за другом размещены значения фиксированного размера., то одну и ту же область памяти можно трактовать одновременно и как одномерный, и как двухмерный массив. Все зависит только от алгоритма обработки этих данных в конкретной программе. Архитектура процессора предоставляет довольно удобные программно-аппаратные средства для работы с массивами. К ним относятся базовые и индексные регистры, позволяющие реализовать несколько режимов адресации данных.

Цель настоящих методических указаний - облегчить самостоятельную работу студентов при разработке программного кода на языке Ассемблер. Они содержат указания к выполнению домашней работы, а так же разобранные примеры основных способов обработки массивов, снабженные необходимыми комментариями, что позволит студенту самостоятельно реализовать индивидуальный вариант задания.

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕЕ ВЫПОЛНЕНИЯ

Целью выполнения домашней работы является практическое овладение навыками разработки программного кода на языке Ассемблер. Обработка массивов.

Основными задачами выполнения домашней работы являются изучение основных приемов обработки массивов: ввод-вывод, доступ к элементам массива, транспонирование, выполнение типовых операции.

Результатами работы являются:

Подготовленный отчет, содержащий:

- цели и задачи;
- блок-схемы разработанных макросов обработки массивов;
- разработанный программный код;
- результаты работы программы.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

МАССИВЫ В АССЕМБЛЕРЕ

Дадим формальное определение:

Массив – структурированный тип данных, состоящий из некоторого числа элементов одного типа. Массив характеризуется типом элементов, числом элементов и способом их нумерации. Число элементов называется размером, или длиной, массива; способ нумерации описывается одномерным целочисленным массивом, называемым формой массива.

Массив — это последовательность элементов, доступ к которым осуществляется при помощи целочисленного индекса. Индексы всегда следуют по порядку, и поэтому очевидным является использование “циклов” для работы с массивами.

Массивы, доступ к элементам которых осуществляется при помощи одного индекса, называются одномерными массивами или векторами.

Для выполнения домашней работы необходимо разобраться в возможностях и особенностях обработки массивов в программах на языке Ассемблер и ответить на следующие вопросы:

- [Как описать массив](#) в программе?
- [Как инициализировать массив](#), то есть, как задать начальные значения его элементов?
- [Как организовать доступ к элементам массива](#)?
- Как организовать [выполнение типовых операций с массивами](#)?

Описание и инициализация массива в программе

Специальных средств описания массивов в программах ассемблера нет. Чтобы использовать массив в программе, его нужно смоделировать.

Можно перечислить элементы массива в поле операндов одной из директив описания данных. При перечислении элементы разделяются запятыми.

Например,

MAS db 1,0,9,8,0,7,8,0,2,0 ; массив из 10 элементов

MAS1 dd 1, 2, 3, 4, 5 ; массив из 4 элементов, размер каждого
; элемента 4 байта.

Можно использовать оператор повторения DUP.

Например,

MAS3 db 4096 dup (0) ; буфер размером 4096, все 0

MAS 4 db 20 dup (?) ; вопросительный знак указывает
;Ассемблеру, что вы резервируете
;ячейку памяти, но не инициализируете
ее

MAS5 db 2, 5, 8, 7 ;объявление матрицы констант
db 1, 8, 6, 0 ;в виде таблицы
db 2, 2, 4, 3
db 3, 5, 7, 9

MAS5 db 4 dup (4 dup (??)) ;инициализация таблицы из 4 строк и
;столбцов

Доступ к элементам массива

При работе с массивами необходимо четко представлять себе, что все элементы массива располагаются в памяти компьютера последовательно. Только программист с, помощью составленного им алгоритма обработки определяет, как нужно трактовать последовательность байтов, составляющих массив. Так, одну и ту же область памяти можно трактовать одновременно и как одномерный, и как двухмерный массив. Все зависит только от алгоритма обработки этих данных в конкретной программе.

Так же следует знать, что Ассемблер не подозревает о существовании индексов элементов массива, а так же об их численных смысловых значениях. При программировании на ассемблере индексы массивов — это обычные адреса, но с ними работают особым образом.

Пример:

Имеется массив X из 30 элементов-слов: X DW 30 DUP (?):

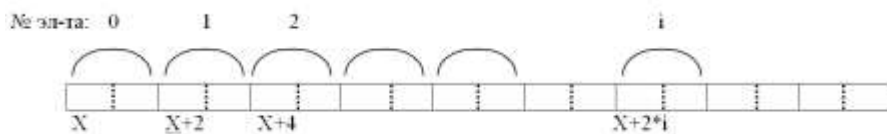


Рис. 1. Индексирование элементов массива

Элементы массива лучше нумеровать, начиная с 0, тогда:

адрес i-го элемента массива X равен $X+2*i$.

Здесь X – символьный начальный адрес массива (постоянная величина), $2*i$ изменяется при перемещении по элементам.

Архитектура процессора предоставляет довольно удобные программно-аппаратные средства для работы с массивами. К ним относятся базовые и индексные регистры, позволяющие реализовать несколько режимов адресации данных. Используя данные режимы адресации, можно организовать эффективную работу с массивами в памяти.

Индексная адресация со смещением — режим адресации, при котором эффективный адрес формируется из двух компонентов:

- **постоянный** (базовый) компонент формируется указанием прямого адреса массива в виде **имени идентификатора**, обозначающего **начало массива**;
- **переменный** (индексный) компонент формируется указанием имени **индексного регистра**, чаще всего SI, DI.

Пример

; Сегмент данных

MAS6 dw 0,1,2,3,4,5

; Сегмент команд

mov SI,4

mov AX, **MAS6[SI]** ;поместить 3-й элемент массива
;MAS6 в регистр AX

Этот способ адресации обычно применяется для доступа к элементам статического массива, начинающегося с адреса MAS6 адрес базы которого не меняется во время выполнения программы.

Базовая индексная адресация со смещением — режим адресации, при котором эффективный адрес формируется максимум из трех компонентов:

- в качестве постоянного (необязательного) компонента может выступать **прямой адрес массива в виде имени идентификатора**, обозначающего начало массива, или непосредственного значения;
- **переменный** (базовый) компонент формируется указанием имени базового регистра BX или BP;
- **переменный** (индексный) компонент формируется указанием имени индексного регистра SI или DI.

Пример

; Сегмент данных

MAS7 db 256 dup(0)

; Сегмент команд

```
mov BX, offset MAS7 ;BX –базовый адрес массива
mov SI,0             ; начальное значение индекса
mov AX,0             ;первый элемент массива
mov CX,256           ;число шагов в цикле
fill: mov [BX][SI],AX ;определим число в массив
inc AX               ;инкремент элемента массива
inc SI               ;смещение в массиве к
                    ;следующему байту
loop fill            ;на метку fill (CX раз)
```

Команда помещает в ячейку памяти, адрес которой **равен сумме значения MAS7, базы BX и индекса SI**, содержимое регистра AX.

Этот способ адресации обычно применяют для доступа к структурированным массивам данных.

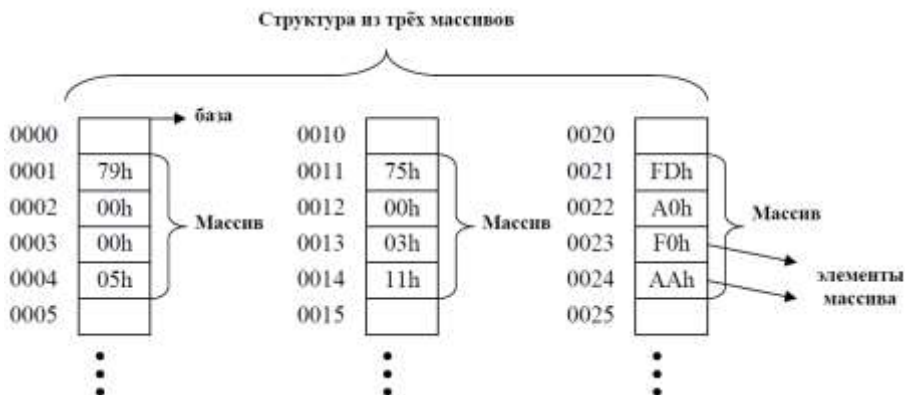


Рис. 2. Относительная индексная адресация на примере доступа к структуре из трех массивов

Процессор позволяет **масштабировать индекс**. Это означает, что если указать после имени индексного регистра символ **звездочки** (+) с последующей цифрой 2, 4 или 8, то содержимое индексного регистра будет умножаться на 2, 4 или 8, то есть масштабироваться. Применение масштабирования облегчает работу с массивами, которые имеют размер элементов, равный 2, 4 или 8 байт, так как процессор сам производит коррекцию индекса для получения адреса очередного элемента массива. Нам нужно лишь загрузить в индексный регистр значение требуемого индекса (считая от 0).

Двумерный массив

Специальных средств для описания двумерного массива в ассемблере нет. Его нужно моделировать. На описании самих данных это почти никак не отражается — память под массив выделяется с помощью директив резервирования и инициализации памяти.

Непосредственно моделирование обработки массива производится в сегменте кода, где программист, описывая алгоритм обработки на ассемблере, определяет, что некоторую область памяти необходимо трактовать как двумерный массив. При этом вы вольны в выборе того, как понимать расположение элементов двумерного массива в памяти: по строкам или по столбцам. Таким образом, структура хранения остается прежней — вектор. Наиболее естественен порядок

расположения элементов массива – по строкам. При этом наиболее быстро изменяется последний элемент последнего элемента индекса.

Например, рассмотрим представление на логическом уровне двумерного массива A_{ij} размерностью $n \times m$, где $0 \leq i \leq n - 1$,

$$0 \leq j \leq m - 1:$$

$a_{00} \ a_{01} \ a_{02} \ a_{03}$

$a_{10} \ a_{11} \ a_{12} \ a_{13}$

$a_{20} \ a_{21} \ a_{22} \ a_{23}$

$a_{30} \ a_{31} \ a_{32} \ a_{33}$

Соответственно этому массиву физическое представление в памяти – вектор будет выглядеть так:

$a_{00} \ a_{01} \ a_{02} \ a_{03} \ a_{10} \ a_{11} \ a_{12} \ a_{13} \ a_{20} \ a_{21} \ a_{22} \ a_{23} \ a_{30} \ a_{31} \ a_{32} \ a_{33} .$

Номер конкретного элемента массива в этой, уже ставшей линейной, последовательности определяется адресной функцией, которая устанавливает положение (адрес) в памяти этого $a_{ij} = n \cdot i + j$.

Для получения адреса элемента массива в памяти необходимо полученное знание умножить на размер элемента и сложить с базовым адресом массива.

Если последовательность однотипных элементов в памяти трактуется как двухмерный массив, расположенный по строкам, то адрес элемента (i, j) вычисляется по формуле:

$$(\text{база} + (\text{количество_элементов_в_строке} \cdot i + j) \cdot \text{размер_элемента})$$

Здесь $i = 0 \dots (n-1)$ — номер строки, а $j = 0 \dots (m-1)$ — номер столбца.

Например, пусть имеется массив чисел (размером в 2 байта) $\text{mas}(i, j)$ с размерностью $4 \cdot 4$ ($i = 0 \dots 3, j = 0 \dots 3$):

23 04 05 67

05 06 07 99

67 08 09 23

87 09 00 08

В памяти элементы этого массива будут расположены в следующей

последовательности:

23 04 05 67 05 06 07 99 67 08 09 23 87 09 00 08

Если мы хотим трактовать эту последовательность как двумерный массив, приведенный раньше, и извлечь, например, элемент $\text{mas}(2, 3) = 23$, то, проведя нехитрый подсчет, убедимся в правильности наших рассуждений:

Эффективный адрес $\text{mas}(2, 3) = \text{mas} + (4 \cdot 2 + 3) \cdot 2 = \text{mas} + 22$.

Посмотрите на представление массива в памяти и убедитесь, что по этому смещению действительно находится нужный элемент массива.

Логично организовать адресацию двумерного массива, используя базово-индексную адресацию. При этом возможны два основных варианта выбора компонентов для формирования эффективного адреса:

- сочетание прямого адреса как базового компонента адреса и двух индексных регистров для хранения индексов: `mov ax, [mas+ebx+esi]`
- сочетание двух индексных регистров, один из которых является и базовым, и индексным одновременно, а другой — только индексным: `mov ax, [ebx+esi]`

Таким образом, необходимо использовать для работы с двумерными массивами адресацию по базе с индексированием и самую полную адресацию - адресация по базе с индексированием и масштабированием.

Для записи адресов его элементов используют два регистра-модификатора, при этом один из регистров обязательно должен быть BX или BP, а другой - SI или DI (модифицировать по парам BX и BP или SI и DI нельзя!).

Адрес элемента в этом случае может выглядеть как $X[BX][SI]$, при этом регистр BX обеспечивает перемещение от строки к строке или от столбца к столбцу - внешний цикл, а SI — соответственно по элементам строки или столбца - внутренний цикл.

Макросредства

При использовании макроопределений код программы становится более читаемым.

Макроопределения Ассемблера аналогичны функциям в языках высокого уровня. Макроопределения позволяют значительно повысить производительность программирования. Особенно это заметно в операциях ввода/вывода.

Основные правила подготовки макроопределения:

1. Макроопределение должно выполнять только одну функцию.
2. Макроопределение должно обеспечить изоляцию данных. Это означает, что **все регистры**, используемые в Макроопределении, перед их использованием должны быть **сохранены в стеке**, а перед выходом из Макроопределения должны быть восстановлены.
3. Каждое макроопределение должно иметь одну точку входа и одну точку выхода.
4. Для эффективности программирования Макроопределения должны быть простыми и легко понимаемыми.
5. Для быстродействия **Макроопределения должны занимать минимум памяти**.
6. Если правила 4 и 5 конфликтуют, предпочтение отдается правилу 4.

Состав макроопределений

Каждое макроопределение имеет три части:

Заголовок - псевдооператор MACRO, в поле метки которого указано имя макроопределения, а в поле операнда - необязательный список **формальных параметров**. В списке формальных параметров указываются переменные - входные параметры, которые можно изменять при каждом вызове макроопределения.

Тело - последовательность операторов Ассемблера (команд и псевдооператоров), которые задают действия, выполняемые макроопределением.

Концевик - псевдооператор ENDM, который отмечает конец макроопределения.

Псевдооператоры макроассемблера

Если тело макроопределения содержит метку или имя в директиве резервирования и инициализации данных, и в программе данная макрокоманда вызывается несколько раз, то в процессе макрогенерации возникнет ситуация, когда в программе один идентификатор будет определен несколько раз, что, естественно, будет распознано транслятором как ошибка. Для выхода из подобной ситуации применяют директиву LOCAL, которая имеет следующий синтаксис:

LOCAL СписокИдентификаторов

Эту директиву необходимо задавать непосредственно за заголовком макроопределения. Результатом работы этой директивы будет генерация в каждом экземпляре макрорасширения уникальных имен для всех идентификаторов, перечисленных в списке. Контроль за правильностью размещения и использования этих уникальных имен берет на себя транслятор.

Для того чтобы использовать описанное макроопределение в нужном месте программы, оно должно быть активизировано с помощью макрокоманды указанием следующей синтаксической конструкции:

Макрокоманда (вызов макроса):

Имя_макроса [фактические параметры]

Результатом применения данной синтаксической конструкции в исходном тексте программы будет ее замещение строками из конструкции *ТелоМакроопределения*. Но это не простая замена. Обычно макрокоманда содержит некоторый список аргументов — *СписокФактическихАргументов*, которыми корректируется макроопределение. Места в теле макроопределения, которые будут замещаться фактическими аргументами из макрокоманды, обозначаются с помощью так называемых формальных аргументов. Таким образом, в результате применения макрокоманды в программе формальные аргументы в макроопределении замещаются

соответствующими фактическими аргументами; в этом и заключается учет контекста. Процесс такого замещения называется **макрогенерацией**, а результатом этого процесса является **макрорасширение**.

Размещения макроопределений

Существует три варианта размещения: **в начале исходного текста программы; в отдельном файле; в макробιβлиотеке.**

В домашнем задании следует использовать вариант: в начале исходного текста программы, **до кода и данных** с тем, чтобы не ухудшать читаемость программы, так как все определяемые макрокоманды актуальны только в пределах одной этой программы.

Приемы обработки массивов

Все операции которые приходится выполнять над элементами одномерных массивов, можно разбить на следующие классы:

- последовательная обработка элементов массивов;
- преформирование массивов;
- одновременная обработка нескольких массивов или подмассивов;
- поиск элементов массива по заданным критериям.

для выполнения этих операций разработаны соответствующие приемы.

Последовательная обработка элементов массивов

Особенностью операций данного класса является то, что количество обрабатываемых элементов массива и шаг изменения индексов известны. Это позволяет для выполнения операции использовать **счетный цикл**, через переменную которого обеспечивается косвенный доступ к элементам. Если просматриваются все элементы массива, то обращения выполняются, используя переменную цикла в качестве индекса, а если с заданным шагом, то для адресации элементов строится выражение, в которое входит переменная цикла, например $2*i+1$.

Примерами задач, требующих выполнения последовательной обработки, являются: ввод и вывод массивов, среднего арифметического, нахождение сумм элементов, как целиком массива, так и его определенной части, произведения элементов, подсчет количества элементов, отвечающих определенному условию или обладающих некоторыми признаками, а также их сумм, произведений и т.п. Кроме того, к этой группе могут быть отнесены задачи формирования значений и замены значений всех элементов значениями, подчиненными определенному закону.

Ввод-вывод элементов матрицы

Количество элементов массива в условии индивидуального варианта не определено, но ограничено. Для реального массива, который будет обрабатываться программой, это количество естественно должно быть известно. Следовательно, прежде чем вводить элементы массива, нужно запросить у пользователя ввод количества элементов n .

Алгоритм решения задачи представлен в виде блок-схемы на рис. 3.

а) блок-схема
ввода матрицы с клавиатуры



б) блок-схема
вывода матрицы на экран



Рис. 3 Алгоритмы ввода вывода матрицы

Макрос ввода матрицы с клавиатуры

```
mReadMatrix macro matrix, row, col
local rowLoop, colLoop
JUMPS                ; Директива, делающая возможным большие прыжки
    push bx          ; Сохранение регистров, используемых в макросе, в стек
    push cx
    push si

    xor bx, bx        ; Обнуляем смещение по строкам
    mov cx, row
rowLoop:              ; Внешний цикл, проходящий по строкам
    push cx

    xor si, si        ; Обнуляем смещение по столбцам
    mov cx, col
colLoop:              ; Внутренний цикл, проходящий по столбцам
    mReadAX buffer 4  ; Макрос ввода значения регистра AX с клавиатуры
                      ; \[Приложение 1\]

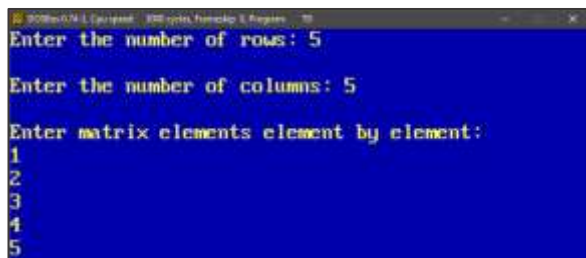
    mov matrix[bx][si], ax
    add si, 2          ; Переходим к следующему элементу (размером в слово)
    loop colLoop

    mWriteStr endl     ; Макрос вывода строки на экран \[Приложение 3\]
                      ; Перенос курсора и каретки на следующую строку
    add bx, col        ; Увеличиваем смещение по строкам
    add bx, col        ; (дважды, так как размер каждого элемента - слово)

    pop cx
    loop rowLoop

    pop si              ; Перенос сохранённых значений обратно в регистры
    pop cx
    pop bx
NOJUMPS              ; Прекращение действия директивы JUMPS
endm mReadMatrix
```

Результат работы макроса:



```
Enter the number of rows: 5
Enter the number of columns: 5
Enter matrix elements element by element:
1
2
3
4
5
```

Макрос вывода матрицы на экран

```
mWriteMatrix macro matrix, row, col
local rowLoop, colLoop
    push ax          ; Сохранение регистров, используемых в макросе, в стек
    push bx
    push cx
    push si

    xor bx, bx       ; Обнуляем смещение по строкам
    mov cx, row
rowLoop:            ; Внешний цикл, проходящий по строкам
    push cx

    xor si, si       ; Обнуляем смещение по столбцам
    mov cx, col
colLoop:            ; Внутренний цикл, проходящий по столбцам
    mov ax, matrix[bx][si] ; bx - смещение по строкам, si - по столбцам

    mWriteAX         ; Макрос вывода значения регистра AX на экран [Приложение 2]
                    ; Вывод текущего элемента матрицы

    xor ax, ax
    mWriteStr tab    ; Макрос вывода строки на экран [Приложение 3]
                    ; Вывод на экран табуляции, разделяющей элементы строки

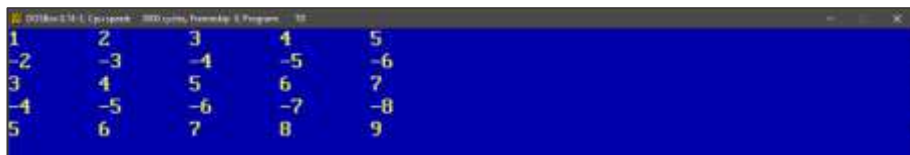
    add si, 2        ; Переходим к следующему элементу (размером в слово)
    loop colLoop

    mWriteStr endl    ; Макрос вывода строки на экран [Приложение 3]
                    ; Перенос курсора и каретки на следующую строку

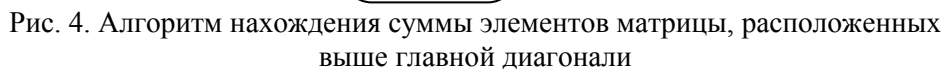
    add bx, col      ; Увеличиваем смещение по строкам
    add bx, col      ; (дважды, так как размер каждого элемента - слово)
    pop cx
    loop rowLoop

    pop si           ; Перенос сохранённых значений обратно в регистры
    pop cx
    pop bx
    pop ax
endm mWriteMatrix
```

Результат работы макроса:



Нахождение суммы элементов матрицы расположенных выше главной диагонали



Для решения поставленной задачи используются три вспомогательные переменные: `matrix` – адрес матрицы в памяти, `row` – количество строк матрицы, `col` – количество столбцов матрицы.

Перебор элементов матрицы происходит в два последовательно выполняемых цикла. При рассмотрении каждого элемента происходит проверка его расположения в матрице относительно главной диагонали. Если рассматриваемый элемент лежит выше главной диагонали, то значение этого элемента прибавляется к значению регистра `DX`. После рассмотрения всех элементов заданной матрицы значение регистра `DX` будет равно сумме элементов, лежащих выше главной диагонали. В конце работы алгоритма происходит вывод значения регистра `DX` на экран.

Макрос получения суммы элементов выше главной диагонали

`mGetSumAboveMainDiagonal` **macro** `matrix`, `row`, `col`

local `rowLoop`, `colLoop`, `belowTheDiagonal`

`push ax` ; Сохранение регистров, используемых в макросе, в стек

`push bx`

`push cx`

`push si`

`push di`

`push dx`

`xor dx, dx` ; В `dx` будет храниться окончательная сумма нужных элементов

`xor di, di` ; `di` - счётчик строк

`xor bx, bx` ; Обнуляем смещение по строкам

`mov cx, row`

`rowLoop:`

`push cx`

`xor si, si` ; `si` - счётчик столбцов

`mov cx, col`

`colLoop:`

`mov ax, matrix[bx][si]` ; `bx` - смещение по строкам, `si` - по столбцам

`cmp si, di` ; Сравниваем счётчики строк и столбцов

`jbe belowTheDiagonal` ; Если элемент выше главной диагонали,

`add dx, ax` ; то добавляем его к результату

`belowTheDiagonal:`

`add si, 2` ; Увеличиваем смещение по столбцу/счётчик столбцов

`loop colLoop`

`add di, 2` ; Увеличиваем счётчик строк

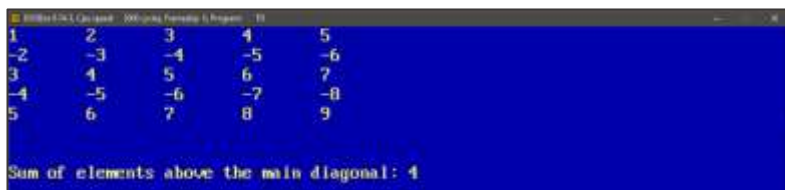
```

add bx, col      ; Увеличиваем смещение по строкам
add bx, col      ; (дважды, так как размер каждого элемента - слово)
pop cx
loop rowLoop

mov ax, dx
mWriteAX         ; Макрос вывода значения регистра AX на экран [Приложение 2]
                ; Вывод на экран результата
pop dx           ; Перенос сохранённых значений обратно в регистры
pop di
pop si
pop cx
pop bx
pop ax
endm mGetSumAboveMainDiagonal

```

Результат работы макроса:



Выборочная обработка элементов матрицы

Нахождение суммы элементов всех строк в пределах [min..max]

Для решения поставленной задачи используются пять вспомогательных переменных: `matrix` – адрес матрицы в памяти, `row` – количество строк матрицы, `col` – количество столбцов матрицы, `min` – минимальное число для поиска суммы, `max` – максимальное число для поиска суммы.

Перебор элементов матрицы происходит в два последовательно выполняемых цикла. Первый цикл осуществляет переход с текущей строки к следующей строке, а второй цикл - переход с текущего столбца к следующему столбцу (к следующему элементу текущей строки). При рассмотрении каждого элемента происходит его проверка на вхождение в диапазон чисел `[min..max]`. Если рассматриваемый элемент входит в данный диапазон, то его значение прибавляется к значению регистра `DX`. После рассмотрения всех элементов текущей строки матрицы (после завершения всех итераций второго цикла) значение регистра `DX` выводится на экран и обнуляется. Алгоритм продолжается до тех пор, пока не будут рассмотрены все элементы всех строк матрицы.

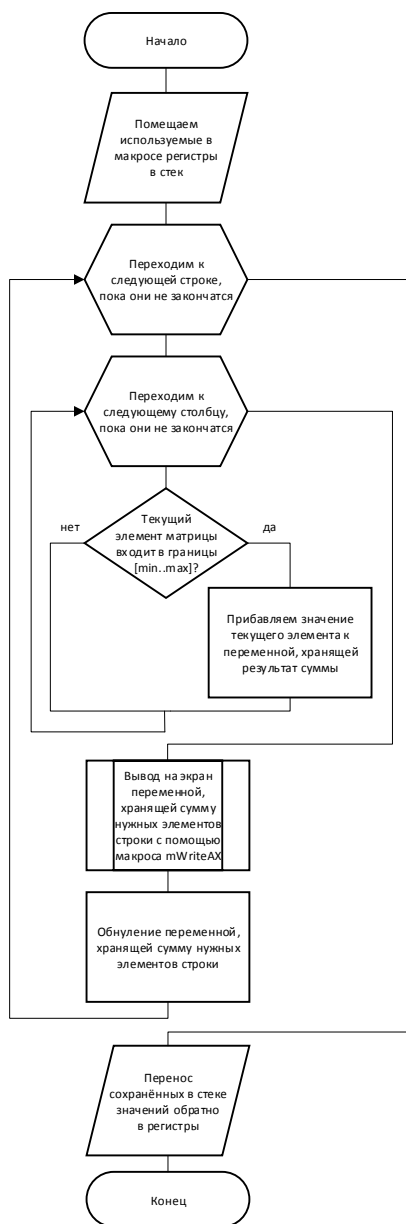


Рис. 5. Алгоритм нахождения суммы элементов всех строк в пределах [min..max]

Макрос получения суммы элементов всех строк в пределах [min..max]

```
mSumOfRowElements macro matrix, row, col, min, max
local rowLoop, colLoop, conditions
    push ax          ; Сохранение регистров, используемых в макросе, в стек
    push bx
    push cx
    push si
    push dx

    xor dx, dx       ; Обнуляем регистр для сохранения результата суммы элементов
    xor bx, bx       ; Обнуляем смещение по строкам
    mov cx, row
rowLoop:             ; Внешний цикл, проходящий по строкам
    push cx

    xor si, si       ; Обнуляем смещение по столбцам
    mov cx, col
colLoop:             ; Внутренний цикл, проходящий по столбцам
    mov ax, matrix[bx][si] ; bx - смещение по строкам, si - по столбцам

    cmp ax, max      ; Проверяем элемент массива на условие A >= min
    jg conditions
    cmp ax, min      ; Проверяем элемент массива на условие A <= max
    jl conditions
    add dx, ax       ; Суммируем все элементы, которые входят в границы [min..max]
    conditions:
    add si, 2        ; Переходим к следующему элементу (размером в слово)
    loop colLoop

    mov ax, dx
mWriteAX             ; Макрос вывода значения регистра AX на экран [Приложение 2]
                    ; Выводим на экран сумму элементов на данной строке матрицы

mWriteStr endl      ; Макрос вывода строки на экран [Приложение 3]
                    ; Перенос курсора и каретки на следующую строку

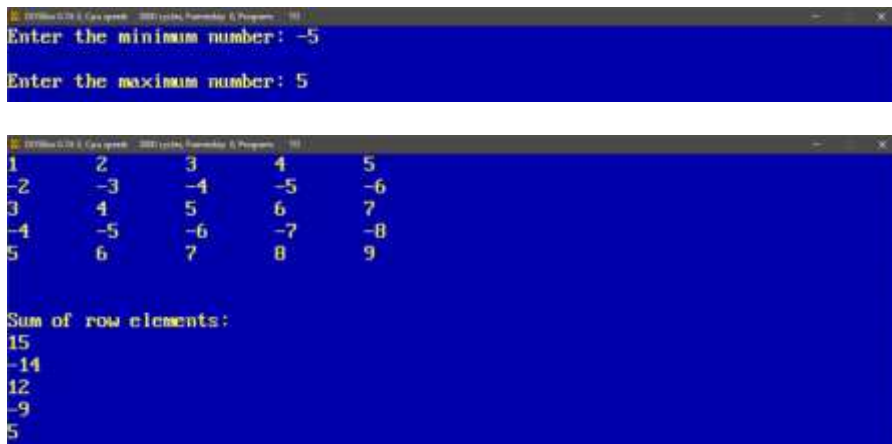
    xor dx, dx

    add bx, col      ; Увеличиваем смещение по строкам
    add bx, col      ; (дважды, так как размер каждого элемента - слово)

    pop cx
    loop rowLoop

    pop dx           ; Перенос сохранённых значений обратно в регистры
    pop si
    pop cx
    pop bx
    pop ax
endm mSumOfRowElements
```


Результат работы макроса:



```
Enter the minimum number: -5
Enter the maximum number: 5

1      2      3      4      5
-2     -3     -4     -5     -6
3       4       5       6       7
-4     -5     -6     -7     -8
5       6       7       8       9

Sum of row elements:
15
-14
12
-9
5
```

Переформирование массива

Переформирование массива предполагает изменение порядка элементов посредством их перемещения, удаления или вставки.

Получение транспонированной матрицы

Для решения поставленной задачи используются четыре вспомогательные переменные: `matrix` – адрес исходной матрицы в памяти, `row` – количество строк матрицы, `col` – количество столбцов матрицы, `resMatrix` – адрес матрицы, в которой будет сформирована транспонированная матрица исходной матрицы `matrix`.

Перебор элементов матрицы происходит в два последовательно выполняемых цикла. При рассмотрении каждого элемента происходит сохранение его значения в стек, после чего это значение заносится в новую матрицу с индексами (смещениями по строкам и столбцам), обратными индексам в исходной матрице. Алгоритм продолжается до тех пор, пока не будут просмотрены все элементы исходной матрицы и не будет полностью сформирована новая транспонированная матрица.



Рис. 6. Алгоритм получения транспонированной матрицы

Макрос получения транспонированной матрицы

mTransposeMatrix macro matrix, row, col, resMatrix

local rowLoop, colLoop

```
    push ax          ; Сохранение регистров, используемых в макросе, в стек
    push bx
    push cx
    push di
    push si
    push dx
```

```
    xor di, di        ; Обнуляем смещение по строкам
```

```
    mov cx, row
```

rowLoop: ; Внешний цикл, проходящий по строкам

```
    push cx
```

```
    xor si, si        ; Обнуляем смещение по столбцам
```

```
    mov cx, col
```

colLoop: ; Внутренний цикл, проходящий по столбцам

```
    mov ax, col
```

```
    mul di             ; Устанавливаем смещение по строкам
```

```
    add ax, si         ; Устанавливаем смещение по столбцам
```

```
    mov bx, ax
```

```
    mov ax, matrix[bx]
```

```
    push ax           ; Заносим текущий элемент в стек
```

```
    mov ax, row
```

```
    mul si             ; Устанавливаем смещение по строкам
```

```
    add ax, di         ; Устанавливаем смещение по столбцам
```

```
    ; (смещения по строкам и столбцам меняются
```

```
    ; местами по сравнению с оригинальной матрицей)
```

```
    mov bx, ax
```

```
    pop ax
```

```
    mov resMatrix[bx], ax ; Заносим в новую матрицу элемент,
```

```
    ; сохранённый в стеке
```

```
    add si, 2          ; Переходим к следующему элементу
```

```
    ; (размером в слово)
```

```
    loop colLoop
```

```
    add di, 2          ; Переходим к следующей строке
```

```
    pop cx
```

```
    loop rowLoop
```

```
    pop dx             ; Перенос сохранённых значений обратно в регистры
```

```
    pop si
```

```
    pop di
```

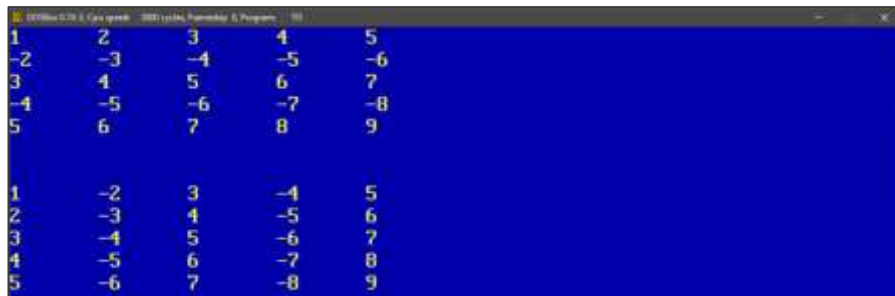
```
    pop cx
```

```
    pop bx
```

```
    pop ax
```

endm mTransposeMatrix

Результат работы макроса:



1	2	3	4	5
-2	-3	-4	-5	-6
3	4	5	6	7
-4	-5	-6	-7	-8
5	6	7	8	9

1	-2	3	-4	5
2	-3	4	-5	6
3	-4	5	-6	7
4	-5	6	-7	8
5	-6	7	-8	9

Поиск элемента массива по заданным критериям

Нахождение строк, состоящих только из положительных элементов

Для решения поставленной задачи используются три вспомогательные переменные: `matrix` – адрес матрицы в памяти, `row` – количество строк матрицы, `col` – количество столбцов матрицы.

Перебор элементов матрицы происходит в два последовательно выполняемых цикла. Первый цикл осуществляет переход с текущей строки к следующей строке, а второй цикл - переход с текущего столбца к следующему столбцу (к следующему элементу текущей строки). При рассмотрении каждого элемента происходит проверка его значения на отрицательность. Если найден отрицательный элемент, то значение регистра `DI` увеличивается на один. После рассмотрения всех элементов текущей строки матрицы (после завершения всех итераций второго цикла) происходит ещё одна проверка: если значение регистра `DI` = 0 (отрицательных элементов в рассмотренной строке не обнаружено), то на экран выводится номер текущей строки. После этого регистр `DI` обнуляется и алгоритм продолжается до тех пор, пока не будут рассмотрены все элементы всех строк матрицы.

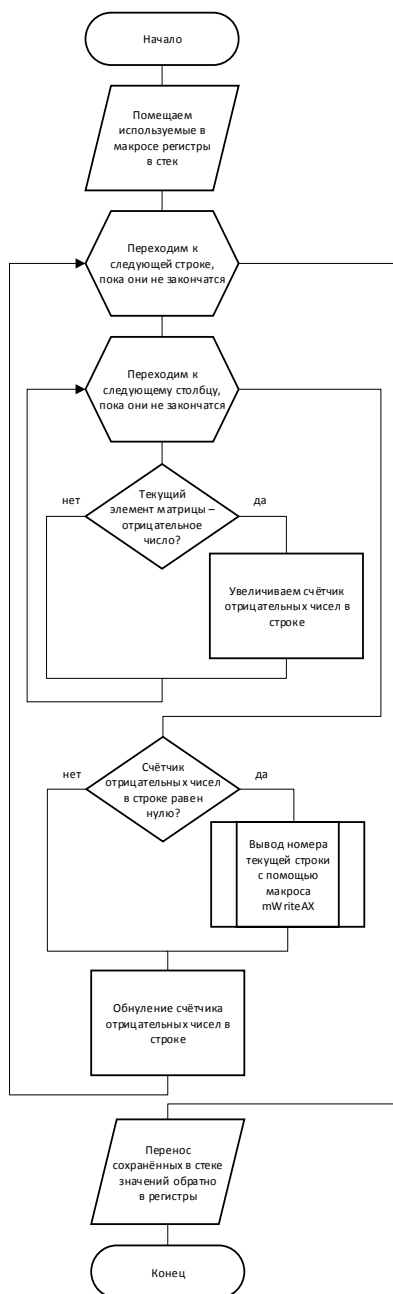


Рис. 7. Алгоритм нахождения строк с положительными элементами

Макрос отображения строк, состоящих только из положительных элементов

```
mSearchPositiveRows macro matrix, row, col
local rowLoop, colLoop, positiveNumber, nonzeroNumber
    push ax                ; Сохранение регистров, используемых в макросе, в стек
    push bx
    push cx
    push si
    push di
    push dx

    mov di, 1              ; di - счётчик строк, начиная с единицы
    xor dx, dx              ; dx - счётчик отрицательных чисел в строке
    xor bx, bx              ; Обнуляем смещение по строкам
    mov cx, row

rowLoop:
    push cx

    xor si, si              ; Обнуляем смещение по столбцам
    mov cx, col

colloop:
    mov ax, matrix[bx][si] ; bx - смещение по строкам, si - по столбцам

    or ax, ax               ; Проверяем, отрицательное ли число
    jns positiveNumber      ; Если найдено отрицательное число, то
    inc dx                  ; Увеличиваем счётчик отрицательных чисел в строке
    positiveNumber:

    add si, 2               ; Переходим к следующему элементу (размером в слово)
    loop colLoop

    or dx, dx               ; Проверяем счётчик отрицательных чисел на равенство 0
    jnz nonzeroNumber      ; Если счётчик отрицательных чисел строки пуст, то:
    mov ax, di
    mWriteAX                ; Макрос вывода значения регистра AX на экран [Приложение 2]
                           ; Выводим номер текущей строки
    mWriteStr space         ; Макрос вывода строки на экран [Приложение 3]
                           ; Выводим пробел между номерами строк
    nonzeroNumber:

    xor dx, dx              ; Обнуляем счётчик отрицательных чисел
    inc di                  ; Увеличиваем счётчик строк
    add bx, col              ; Увеличиваем смещение по строкам
    add bx, col              ; (дважды, так как размер каждого элемента - слово)
    pop cx
    loop rowLoop

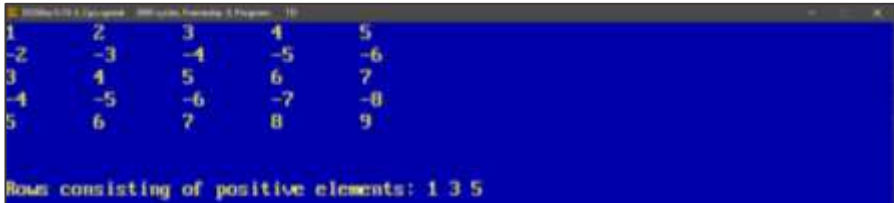
    pop dx                  ; Перенос сохранённых значений обратно в регистры
    pop di
    pop si
```

```

    pop cx
    pop bx
    pop ax
endm mSearchPositiveRows

```

Результат работы макроса:



Пример реализации основной части программы, осуществляющей работу интерфейса:

```

.MODEL small
.STACK 100h
.486 ; Включает сборку инструкций для процессора 80386
.DATA
endl
tab
space
inputRows
inputColumns
inputElements
inputMin
inputMax

menuInstruction
menu1
menu2
menu3
menu4
in each row'
menu5
menu6
menu0

sumRowElements
sumAboveDiagonal
posRowsFound

buffer
min
max
rows
cols
currentMatrix
transposedMatrix

```

```
.CODE
Start:
mov ax, @data
mov ds, ax
```

`include macros.asm` ; Подключение файла со всеми вышеописанными макросами

; Вывод на экран меню, а также осуществление выбора следующего пункта программы
menu:

```
mCLS ; Макрос очистки экрана и установки вида окна
; [Приложение 4]
mWriteStr menuInstruction ; Макрос вывода строки на экран [Приложение 3]
mWriteStr endl
mWriteStr menu1
mWriteStr menu2
mWriteStr menu3
mWriteStr menu4
mWriteStr menu5
mWriteStr menu6
mWriteStr menu0
```

```
mov ah, 00h
int 16h ; Ожидание нажатия символа и получение его значения в al
```

```
cmp al, "1"
je writeMatrix
```

```
cmp al, "2"
je consoleInput
```

```
cmp al, "3"
je transposeMatrix
```

```
cmp al, "4"
je task1
```

```
cmp al, "5"
je task2
```

```
cmp al, "6"
je task3
```

```
cmp al, "0"
je exit
```

```
jmp menu
```


; Вывод элементов матрицы на экран

writeMatrix:

```
mCLS ; Макрос очистки экрана и установки вида окна [Приложение 4]
mWriteMatrix currentMatrix, rows, cols
mov ah, 07h ; Задержка экрана
int 21h
jmp menu
```

; Ввод элементов матрицы из консоли

consoleInput:

```
mCLS ; Макрос очистки экрана и установки вида окна [Приложение 4]

mWriteStr inputRows ; Макрос вывода строки на экран [Приложение 3]
mReadAX buffer 2 ; Макрос ввода значения регистра AX с клавиатуры
; [Приложение 1]
; Ввод количества строк

mov rows, ax
mWriteStr endl ; Макрос вывода строки на экран [Приложение 3]

mWriteStr inputColumns ; Макрос вывода строки на экран [Приложение 3]
mReadAX buffer 2 ; Макрос ввода значения регистра AX с клавиатуры
; [Приложение 1]
; Ввод количества столбцов

mov cols, ax
mWriteStr endl ; Макрос вывода строки на экран [Приложение 3]

mWriteStr inputElements ; Макрос вывода строки на экран [Приложение 3]

mReadMatrix currentMatrix, rows, cols
jmp writeMatrix
```

; Получение и вывод транспонированной матрицы

transposeMatrix:

```
mCLS ; Макрос очистки экрана и установки вида окна
; [Приложение 4]
mWriteMatrix currentMatrix, rows, cols
mWriteStr endl ; Макрос вывода строки на экран [Приложение 3]
mWriteStr endl
mTransposeMatrix currentMatrix, rows, cols, transposedMatrix
mWriteMatrix transposedMatrix, cols, rows

mov ah, 07h ; Задержка экрана
int 21h
jmp menu
```

; Получение суммы элементов всех строк в пределах [min..max]

task1:

```

mCLS ; Макрос очистки экрана и установки вида окна
; [Приложение 4]

mWriteStr inputMin ; Макрос вывода строки на экран [Приложение 3]
mReadAX buffer 4 ; Макрос ввода значения регистра AX с клавиатуры
; [Приложение 1]
; Ввод нижнего предела

mov min, ax
mWriteStr endl

mWriteStr inputMax ; Макрос вывода строки на экран [Приложение 3]
mReadAX buffer 4 ; Макрос ввода значения регистра AX с клавиатуры
; [Приложение 1]
; Ввод верхнего предела

mov max, ax
mWriteStr endl ; Макрос вывода строки на экран [Приложение 3]

mov ax, max
cmp ax, min
jge notSwap ; Если максимальное и минимальное числа введены неверно,
xchg min, ax ; то меняем их местами
mov max, ax

notSwap:
mCLS ; Макрос очистки экрана и установки вида окна [Приложение 4]
mWriteMatrix currentMatrix, rows, cols
mWriteStr endl ; Макрос вывода строки на экран [Приложение 3]
mWriteStr endl
mWriteStr sumRowElements
mWriteStr endl
mSumOfRowElements currentMatrix, rows, cols, min, max

mov ah, 07h ; Задержка экрана
int 21h
jmp menu

```

; Получение строк, состоящих только из положительных элементов

task2:

```

mCLS ; Макрос очистки экрана и установки вида окна [Приложение 4]
mWriteMatrix currentMatrix, rows, cols
mWriteStr endl ; Макрос вывода строки на экран [Приложение 3]
mWriteStr endl
mWriteStr posRowsFound
mSearchPositiveRows currentMatrix, rows, cols

mov ah, 07h ; Задержка экрана
int 21h

```

`jmp menu`

; Получение суммы элементов выше главной диагонали

task3:

```
mCLS ; Макрос очистки экрана и установки вида окна [Приложение 4]
mWriteMatrix currentMatrix, rows, cols
mWriteStr endl ; Макрос вывода строки на экран [Приложение 3]
mWriteStr endl
mWriteStr sumAboveDiagonal
mGetSumAboveMainDiagonal currentMatrix, rows, cols
```

`mov ah, 07h` ; Задержка экрана

`int 21h`

`jmp menu`

; Завершение программы

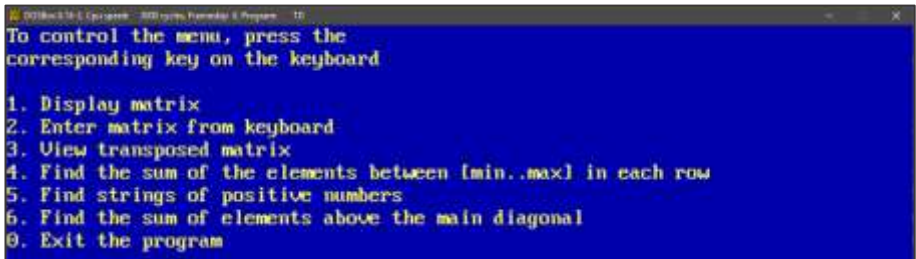
exit:

`mov ax, 4c00h`

`int 21h`

`end` Start

Результат работы:

A screenshot of a DOS command window titled "DOS Batch File - 386cpu, Pentium & Program - 10". The window has a blue background with white text. It displays a menu for controlling the program. The text is as follows:

```
To control the menu, press the
corresponding key on the keyboard

1. Display matrix
2. Enter matrix from keyboard
3. View transposed matrix
4. Find the sum of the elements between [min..max] in each row
5. Find strings of positive numbers
6. Find the sum of elements above the main diagonal
0. Exit the program
```

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Работа предусматривает применение основных приемов обработки массивов.

1. Ввести с клавиатуры и вывести на экран матрицу $m \times n$ (матрица не обязательно должна быть квадратная и может содержать нулевые и отрицательные элементы, если это предусмотрено условиями задания варианта);
2. Реализовать простейший интерфейс взаимодействия с пользователем для выполнения задания варианта до выбора команды «Выход»;
 - Транспонировать матрицу, результат вынести на экран;
 - Обработка элементов матрицы (задание а и б в условии варианта), результат выполнения вывести на экран;
 - Реализовать завершение выполнения программы.

ВАРИАНТЫ ЗАДАНИЙ

Вариант 1

Дана матрица.

- а) В каждой строке матрицы найти количество элементов, меньших заданного значения.
- б) Подсчитать количество ненулевых строк.
- в) Вычислить сумму элементов матрицы, выделенных чёрным цветом (матрица квадратная).



Вариант 2

Дана матрица.

- а) В каждой строке матрицы найти сумму элементов, находящихся в диапазоне между двумя заданными числами.
- б) Проверить, есть ли в матрице строка из положительных чисел.
- в) Вычислить сумму элементов матрицы, выделенных чёрным цветом (матрица квадратная).



Вариант 3

Дана матрица.

- а) В каждой строке матрицы найти произведение элементов, расположенных после максимального элемента в этой строке.
- б) Проверить, симметричны ли все строки относительно среднего элемента.
- в) Вычислить сумму элементов матрицы, выделенных чёрным цветом (матрица квадратная).



Вариант 4

Дана матрица.

- а) В каждой строке матрицы найти сумму модулей элементов, расположенных после первого отрицательного элемента в этой строке.
- б) Проверить, равны ли строки первая и последняя, вторая и предпоследняя и т. д.
- в) Вычислить сумму элементов матрицы, выделенных чёрным цветом (матрица квадратная).



Вариант 5

Дана матрица.

- а) В каждой столбце матрицы найти количество элементов, находящихся в диапазоне между двумя заданными числами.
- б) Найти среднее арифметическое в каждой ненулевой строке.
- в) Найдите первый положительный элемент среди элементов матрицы, выделенных чёрным цветом (матрица квадратная).



Вариант 6

Дана матрица.

- а) В каждом столбце матрицы найти сумму элементов, расположенных после первого отрицательного элемента в столбце.
- б) Поменять местами i и j строки.
- в) Найдите первый отрицательный элемент среди элементов матрицы, выделенных чёрным цветом (матрица квадратная).



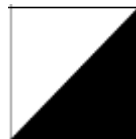
Вариант 7

Дана матрица.

а) В каждой строке матрицы найти сумму элементов, расположенных после минимального элемента в строке.

б) Заменить i строку на копию j строки.

в) Вычислить сумму отрицательных элементов матрицы среди элементов, выделенных чёрным цветом (матрица квадратная).



Вариант 8

Дана матрица.

а) В каждой строке матрицы расположить сначала все отрицательные элементы, затем все положительные, а потом – нулевые.

б) Заменить все нулевые строки на заданный вектор.

в) Вычислить сумму отрицательных элементов матрицы среди элементов, выделенных чёрным цветом.



Вариант 9

Дана матрица.

а) В каждом столбце матрицы найти произведение элементов, расположенных после максимального по модулю элемента.

б) Замените отрицательные элементы их квадратами и упорядочьте каждую строку по возрастанию.

в) Найдите максимальный элемент среди элементов матрицы, выделенных чёрным цветом (матрица квадратная).



ФОРМА ОТЧЕТА ПО ДОМАШНЕЙ РАБОТЕ

На выполнение домашней работы отводится 12 академических часа: 10 часов на выполнение и сдачу домашней работы и 2 часа на подготовку отчета.

Номер варианта студенту выдается преподавателем.

Отчет на защиту предоставляется в печатном виде.

Структура отчета (на отдельном листе(-ах)): титульный лист, цель выполнения домашней работы, формулировка задания (вариант), блок-схемы обработки элементов массива, листинг программы, пояснения к программе, результаты выполнения программы, вывод.

КОНТРОЛЬНЫЕ ВОПРОСЫ:

1. Что такое массив?
2. Директива `per`. Назначение, применение.
3. Как осуществляется базовая индексация со смещением?
4. Синтаксис массива. Оператор `dup`, перечисление элементов, использование цикла.
5. Как описать массив в программе?
6. Как организовать доступ к элементам массива?
7. Как организовать массивы с размерностью более одной?

ЛИТЕРАТУРА

Основная литература

1. Калашников О.А. Ассемблер- это просто. Учимся программировать [Текст] / О.А. Калашников.- СПб. БХВ-Петербург,2012.- 336 с.
2. Кирнос В. Н. Введение в вычислительную технику: основы организации ЭВМ и программирование на Ассемблере[Электронный ресурс]: учеб.пособие /В. Н. Кирнос. - Томск: Эль Контент, 2011. -172с.
[URL://biblioclub.ru/index.php?page=book_red&id=208652](http://biblioclub.ru/index.php?page=book_red&id=208652)

Дополнительная литература

3. Юров В. И. ASSEMBLER[Текст]. Учебник для вузов /В. И. Юров. 2-е изд.– Спб.:Питер 2010. – 637с.: ил.
4. Юров В. И. ASSEMBLER[Текст]. Практикум. / В. И. Юров. 2-е изд.– Спб.:Питер 2007. – 399 с.
5. Зубков С.В. ASSEMBLER для DOS, WINDOWS, UNIX [Текст] / С.В. Зубков-3-е изд., М.:ДМК Пресс; 2004. – 608 с.: ил.

Электронные ресурсы:

1. Научная электронная библиотека <http://eLIBRARY.ru>.
2. Электронно-библиотечная система <http://e.lanbook.com>.
3. Электронно-библиотечная система «Университетская библиотека онлайн» <http://biblioclub.ru>.

Приложение 1

Макрос ввода целого числа в регистр ax в 10-ричной системе счисления

```
mReadAX macro buffer, size
local input, startOfConvert, endOfConvert
    push bx ; Сохранение регистров, используемых в макросе, в стек
    push cx
    push dx

input:
    mov [buffer], size ; Задаём размер буфера
    mov dx, offset [buffer]
    mov ah, 0Ah ; 0Ah - функция чтения строки из консоли
    int 21h

    mov ah, 02h ; 02h - функция вывода символа на экран
    mov dl, 0Dh
    int 21h ; Переводим каретку на новую строку

    mov ah, 02h ; 02h - функция вывода символа на экран
    mov dl, 0Ah
    int 21h ; Переносим курсор на новую строку

    xor ah, ah
    cmp ah, [buffer][1] ; Проверка на пустую строку
    jz input ; Если строка пустая - переходим обратно к вводу

    xor cx, cx
    mov cl, [buffer][1] ; Инициализируем переменную счетчика

    xor ax, ax
    xor bx, bx
    xor dx, dx
    mov bx, offset [buffer][2] ; bx = начало строки
                                ; (строка начинается со второго байта)

    cmp [buffer][2], '-' ; Проверяем, отрицательное ли число
    jne startOfConvert ; Если отрицательное - пропускаем минус
    inc bx
    dec cl

startOfConvert:
    mov dx, 10
    mul dx ; Умножаем на 10 перед сложением с младшим разрядом
    cmp ax, 8000h ; Если число выходит за границы, то
    jae input ; возвращаемся на ввод числа
```

```

mov dl, [bx]           ; Получаем следующий символ
sub dl, '0'            ; Переводим его в числовой формат

add ax, dx             ; Прибавляем к конечному результату
cmp ax, 8000h          ; Если число выходит за границы, то
jae input              ; возвращаемся на ввод числа

inc bx                 ; Переходим к следующему символу
loop startOfConvert

cmp [buffer][2], '-'   ; Ещё раз проверяем знак
jne endOfConvert       ; Если знак отрицательный, то
neg ax                 ; инвертируем число

endOfConvert:
pop dx                 ; Перенос сохранённых значений обратно в регистры
pop cx
pop bx
endm mReadAX

```

Приложение 2

Макрос вывода на экран содержания регистра ax в 10-ричной системе счисления

```
mWriteAX macro
local convert, write
    push ax          ; Сохранение регистров, используемых в макросе, в стек
    push bx
    push cx
    push dx
    push di

    mov cx, 10       ; cx - основание системы счисления
    xor di, di       ; di - количество цифр в числе

    or ax, ax        ; Проверяем, равно ли число в ax нулю и устанавливаем флаги
    jns convert      ; Переход к конвертированию, если число в ax положительное

    push ax

    mov dx, '-'
    mov ah, 02h      ; 02h - функция вывода символа на экран
    int 21h          ; Вывод символа "-"

    pop ax
    neg ax           ; Инвертируем отрицательное число

convert:
    xor dx, dx

    div cx           ; После деления di = остатку от деления ax на cx
    add dl, '0'      ; Перевод в символьный формат
    inc di           ; Увеличиваем количество цифр в числе на 1

    push dx          ; Складываем в стек

    or ax, ax        ; Проверяем, равно ли число в ax нулю и устанавливаем флаги
    jnz convert      ; Переход к конвертированию, если число в ax не равно нулю

write:
    pop dx           ; Вывод значения из стека на экран
    ; dl = очередной символ

    mov ah, 02h
    int 21h          ; Вывод очередного символа
    dec di           ; Повторяем, пока di <> 0
    jnz write
```

```
    pop di      ; Перенос сохранённых значений обратно в регистры
    pop dx
    pop cx
    pop bx
    pop ax
endm mWriteAX
```

Макрос вывода заданной строки string на экран

```
mWriteStr macro string
    push ax                ; Сохранение регистров, используемых в макросе, в стек
    push dx

    mov ah, 09h           ; 09h - функция вывода строки на экран
    mov dx, offset string
    int 21h

    pop dx                ; Перенос сохранённых значений обратно в регистры
    pop ax
endm mWriteStr
```

Приложение 4

Макрос очистки экрана и установки цветов фона и текста

```
mCLS macro
    push ax                ; Сохранение регистров, используемых в макросе, в стек
    push bx
    push cx
    push dx

    mov ah, 10h
    mov al, 3h
    int 10h                ; Включение режима видеоадаптора с 16-ю цветами
    mov ax, 0600h          ; ah = 06 - прокрутка вверх
                           ; al = 00 - строки появляются снизу и заполняются нулями

    mov bh, 00011110b      ; 0001 - синий (фон), 1110 - желтый (текст)
    mov cx, 0000b          ; ah = 00 - строка верхнего левого угла
                           ; al = 00 - столбец верхнего левого угла
    mov dx, 184Fh          ; dh = 18h - строка нижнего правого угла
                           ; dl = 4Fh - столбец нижнего правого угла
    int 10h                ; Очистка экрана и установка цветов фона и текста

    mov dx, 0              ; dh - строка, dl - столбец
    mov bh, 0              ; Номер видео-страницы
    mov ah, 02h            ; 02h - функция установки позиции курсора
    int 10h                ; Устанавливаем курсор на позицию (0, 0)

    pop dx                 ; Перенос сохранённых значений обратно в регистры
    pop cx
    pop bx
    pop ax
endm mCLS
```