

Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный университет
имени Франциска Скорины»

Е. А. РУЖИЦКАЯ, О. Г. ОСИПОВА, В. А. КОВАЛЁВА

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ASSEMBLER

Программирование ветвящихся и циклических вычислительных процессов, обработка массивов

Практическое пособие

для студентов специальностей

1–40 01 01 «Программное обеспечение

информационных технологий»,

1–40 04 10 «Информатика и технологии программирования»

Гомель
ГГУ им. Ф. Скорины
2016

УДК 004.431.4(076)
ББК 32.973.21я73
Р837

Рецензенты:

кандидат физико-математических наук Т. В. Тихоненко,
кандидат технических наук В. Д. Левчук

Рекомендовано к изданию научно-методическим советом
учреждения образования «Гомельский государственный
университет имени Франциска Скорины»

Ружицкая, Е. А.

Р837 Программирование на языке Assembler : программирование ветвящихся и циклических вычислительных процессов, обработка массивов : практическое пособие / Е. А. Ружицкая, О. Г. Осипова, В. А. Ковалёва ; М-во образования Республики Беларусь, Гомельский гос. ун-т им. Ф. Скорины. – Гомель: ГГУ им. Ф. Скорины, 2016. – 47 с.
ISBN 978-985-577-111-2

Практическое пособие предназначено для оказания помощи студентам в обладении машинно-ориентированным языком программирования Assembler. Излагается теоретический материал и дается практическое руководство по таким темам, как программирование ветвящихся и циклических вычислительных процессов, обработка массивов.

Адресовано студентам 1 курса специальностей 1–40 01 01 «Программное обеспечение информационных технологий», 1–40 04 10 «Информатика и технологии программирования».

УДК 004.431.4(076)
ББК 32.973.21я73

ISBN 978-985-577-111-2 © Ружицкая Е. А., Осипова О. Г.,
Ковалёва В. А., 2016
© Учреждение образования
«Гомельский государственный
университет имени Франциска Скорины»,
2016

Оглавление

Предисловие	4
Тема 1. Программирование ветвящихся вычислительных процессов	5
1.1 Команда вычитания	5
1.2 Команды передачи управления	5
1.3 Команды условной передачи управления	5
1.4 Команда безусловного перехода	6
1.5 Псевдооператоры определения процедур	6
1.6 Псевдооператоры внешних ссылок	6
1.7 Ассемблерные подпрограммы	7
1.8 Оператор asm	8
1.9 Модели структуры программы	9
1.10 Практическое задание	23
Тема 2. Программирование циклических вычислительных процессов. Обработка массивов	26
2.1 Команды управления циклами	26
2.2 Псевдооператоры описания переменных, используемые для описания массивов	26
2.3 Команды пересылки адреса	27
2.4 Операции, возвращающие значения	27
2.5 Операции присваивания атрибутов	28
2.6 Режимы адресации	29
2.7 Примеры обработки одномерных массивов	31
2.8 Примеры обработки двумерных массивов	38
2.9 Практические задания	44
Литература	47

Предисловие

Несмотря на обилие языков высокого уровня, таких как C/C++, Delphi и других, ни один язык, даже такой популярный как C++, не может претендовать на то, чтобы на нем можно было написать действительно «все». На ассемблере пишут:

– все, что требует максимальной скорости выполнения: основные компоненты компьютерных игр, ядра операционных систем реального времени и просто критические участки программ;

– все, что взаимодействует с внешними устройствами: драйверы, программы, работающие напрямую с портами, звуковыми и видеоплатами;

– все, что использует полностью возможности процессора: ядра многозадачных операционных систем, DPМI-серверы и любые программы, переводящие процессор в защищенный режим;

– все, что полностью использует возможности операционной системы: вирусы и антивирусы, защиты от несанкционированного доступа, программы, обходящие эти защиты, и программы, защищающиеся от этих программ и многое другое.

Стоит познакомиться с ассемблером поближе, как оказывается, что многое из того, что обычно пишут на языках высокого уровня, лучше, проще и быстрее написать на ассемблере.

Знание ассемблера часто помогает отлаживать программы на других языках, потому что оно дает представление о том, как на самом деле функционирует компьютер и что происходит при выполнении команд языка высокого уровня.

Практическое пособие предназначено для оказания помощи студентам в обладении машинно-ориентированным языком программирования Assembler. Излагается теоретический материал и дается практическое руководство по таким темам, как программирование ветвящихся и циклических вычислительных процессов, обработка массивов.

Язык программирования Assembler изучается студентами 1 курса специальности 1–40 01 01 «Программное обеспечение информационных технологий» в рамках дисциплины «Языки программирования» и 1–40 04 10 «Информатика и технологии программирования» в курсе «Программирование».

Тема 1. Программирование ветвящихся вычислительных процессов

Для программирования ветвящихся вычислительных процессов, используются команда вычитания `cmp` и команды передачи управления.

1.1 Команда вычитания

Формат команды вычитания:

`cmp П, И`

Эта команда вычитает операнд-источник из операнда-приемника, но не сохраняет результат вычитания в операнде-приемнике, а только соответствующим образом воздействует на флаги.

1.2 Команды передачи управления

Команды передачи управления делятся на 4 группы:

- 1 Команды безусловной передачи управления.
- 2 Команды условной передачи управления.
- 3 Команды управления циклами.
- 4 Команды работы с процедурами.

1.3 Команды условной передачи управления

Команды условной передачи управления позволяют принять решение в зависимости от определенного условия. Если условие истинно, то осуществляется переход по указанной в команде метке. В противном случае выполняется команда, следующая за командой перехода.

Обычно команды условной передачи управления используются совместно с командой сравнения `cmp`.

Формат команды условного перехода:

`jx метка`

где `x` – модификатор команды,

`метка` – метка перехода, которая находится не далее –128 или +127 байтов от команды условной передачи. В таблице 1.1. представлены некоторые команды условного перехода.

Пример:

`cmp ax, bx`

`;сравниваем содержимое регистров ax и bx`

`jg met1 ;если ax>bx то переход на met1`

Таблица 1.1 – Команды условного перехода

Условие перехода	Следующая за <code>cmp</code> команда	
	для чисел без знака	для чисел со знаком
<code>П > И</code>	<code>ja</code>	<code>jg</code>
<code>П = И</code>	<code>je</code>	<code>je</code>
<code>П < И</code>	<code>jb</code>	<code>jl</code>
<code>П ≥ И</code>	<code>jae</code>	<code>jge</code>
<code>П ≤ И</code>	<code>jbe</code>	<code>jle</code>
<code>П ≠ И</code>	<code>jne</code>	<code>jne</code>

1.4 Команда безусловного перехода

Эта команда используется для обхода группы команд, которым передается управление из другой части программы.

Формат команды:

`jmp метка`

где `метка` – имя метки перехода, которая находится не далее –128 или +127 байтов от команды безусловного перехода.

1.5 Псевдооператоры определения процедур

Формат описания процедуры:

`имя_процедуры proc атрибут_дистанции`

`...`

`ret`

`имя_процедуры endp`

Каждая процедура должна начинаться оператором `proc` и заканчиваться оператором `endp`. Процедура должна содержать команду возврата из процедуры `ret`.

Атрибуты дистанции:

`far` – дальняя процедура,

`near` – близкая процедура.

Процедура с атрибутом `near` может быть вызвана только из того сегмента команд, в котором она определена.

Формат вызова процедуры:

`call имя_процедуры`

1.6 Псевдооператоры внешних ссылок

Псевдооператор `public` делает указанный в нем идентификатор доступным для других программных модулей, которые впоследствии

могут загружаться вместе с данным модулем. Идентификатор может быть именем переменной, меткой или именем, определенным псевдооператором = или EQU.

Псевдооператор `extrn` описывает идентификаторы, которые объявлены в операторе `public` в других программных модулях.

Формат оператора:

```
extrn имя:тип [, имя:тип, ...]
```

где `имя` – идентификатор, определенный в другом программном модуле, а тип задается следующим образом:

– если `имя` является идентификатором, определенным в сегменте данных или в дополнительном сегменте, то тип может принимать значения `byte`, `word`, `dword`;

– если `имя` – метка процедуры, то тип может быть `near` или `far`;

– если `имя` относится к константе, определенной псевдооператорами = или EQU, то тип должен быть `abs`.

Псевдооператор `include` во время трансляции вставляет в текущий файл исходной программы файл исходных операторов, указанный в команде `include`.

Формат команды:

```
include имя_файла
```

Пример:

```
include file.asm
```

1.7 Ассемблерные подпрограммы

Ассемблерные подпрограммы – это процедуры и функции, объявленные с директивой `assembler`. В таких подпрограммах исполняемая часть не содержит `begin...end` и состоит из единственного ассемблерного оператора `asm...end`.

```
Function F(x,y:byte):byte; assembler;
asm
    mov al,x
    imul y
end;
```

```
Procedure Proc; assembler;
var
    x,y:byte;
asm
    ...
end;
```

Ассемблерные функции должны следующим образом возвращать результат своей работы:

– длиной 1 байт (`byte`, `char`) в регистре `al`;

– длиной 2 байта (`integer`, `word`) в регистре `ax`;

– длиной 4 байта (`Pointer`, `LongInt`) в регистрах `dx` (старшее слово) и `ax` (младшее слово).

Во встроенном ассемблере могут использоваться 3 предопределенных имени:

`@code` – текущий сегмент кода;

`@data` – текущий сегмент данных;

`@result` – ссылка внутри функции на её результат.

Имена `@code` и `@data` могут использоваться только в сочетании с директивой `seg` для ссылки на нужный сегмент.

```
asm
    mov ax, seg @data
    mov ds, ax
end
```

Имя `@result` используется для присвоения результата функции:

```
Function min (x,y:integer):integer;
begin
    asm
        mov ax,x
        cmp ax,y
        jl met
        mov ax,y
    met: mov @result, ax
    end
end;
```

1.8 Оператор `asm`

Зарезервированное слово `asm` открывает доступ к средствам встроенного ассемблера. Этот оператор может располагаться только внутри исполняемой части программы (подпрограммы). Область действия оператора `asm` ограничивается ближайшим словом `end`.

```
if x>10 then asm
    ...
end
else asm
    ...
end
```

Каждая ассемблерная команда должна быть в отдельной строке или отделяться ;

```
asm
mov ax,bx
mov cx,dx; mov ax,a {комментарии}
end
```

1.9 Модели структуры программы

Рассмотрим различные модели структуры программы на примере вычисления значения ветвящейся функции:

$$f = \begin{cases} \frac{x^2 + y^2 - 5}{3 + x^2}, & xy < 0; \\ 3y^2 + 4, & xy > 10; \\ \frac{3y - x}{5 + y + x^2}, & 0 \leq xy \leq 10. \end{cases}$$

Реализуем 5 вариантов структуры программы:

- 1) без использования внутренних процедур;
- 2) с использованием внутренних процедур;
- 3) с использованием внешних процедур;
- 4) с использованием процедур ввода-вывода;
- 5) с использованием ассемблерных подпрограмм.

1-й вариант – без использования внутренних процедур

```
;Пример программы, вычисляющей значение
;ветвящейся функции, ветви вычисляются по меткам
;сегмент данных
Dseg segment para public 'data'
x db 1
y db 2
f db ?
mes db 'конец программы$'
Dseg ends
;сегмент стека
Sseg segment para stack 'stack'
dw 30 dup(0)
Sseg ends
;сегмент кода
Cseg segment para public 'code'
osn proc near
assume cs:cseg, ds:dseg, ss:sseg
mov ax,dseg
mov ds,ax
mov al,x
```

```
imul y
cmp al,10 ;сравниваем содержимое
;регистра al с 10
jg m1 ;если al>10 перейти на метку m1
cmp al,0 ;сравниваем содержимое регистра al с 0
jl m2 ;если al<0 перейти на метку m2
;вычисляем значение функции при 0<=al<=10
mov bl,y
add bl,5 ;bl=y+5
mov al,x ;al=x
imul x ;al=x*x
add bl,al ;bl=y+5+x*x
mov al,3 ;al=3
imul y ;al=3*y
sub al,x ;al=3*y-x
cbw
idiv bl ;al=(3*y-x)/(5+y+x*x)
jmp m3 ;переход на вывод результатов
;и конец программы
;вычисляем значение функции при al>10
m1:mov al,3
imul y ;al=3*y
imul y ;al=3*y*y
add al,4 ;al=3*y*y+4
jmp m3 ;переход на вывод результатов и
;и конец программы
;вычисляем значение функции при al<0
m2:mov al,x
imul x
add al,3 ;al=x*x+3
mov cl,al ;cl=x*x+3
mov al,x ;al=x
imul x ;al=x*x
mov bl,al ;bl=x*x
mov al,y ;al=y
imul y ;al=y*y
add al,bl ;al=y*y+x*x
sub al,5 ;al=y*y+x*x-5
cbw
idiv cl ;al=(y*y+x*x-5)/(x*x+3)
m3:mov f,al
lea dx,mes ;вывод сообщения 'конец программы'
mov ah,9
int 21h
mov ax, 4c00h ;завершение программы
```

```

int 21h
osn endp
Cseg ends
end osn

```

2-й вариант – использование внутренних процедур

```

;Пример программы, вычисляющей значение
;ветвящейся функции.
;Ветви находятся во внутренних процедурах
;сегмент данных
Dseg segment para public 'data'
    x db 1
    y db 2
    f db ?
    mes db 'конец программы$'
Dseg ends
;сегмент стека
Sseg segment para stack 'stack'
    dw 30 dup(0)
Sseg ends
;сегмент кода
Cseg segment para public 'code'
;основная программа
osn proc near
    assume cs:cseg,ds:dseg,ss:sseg
    mov ax,dseg
    mov ds,ax
    mov al,x
    imul y
    cmp al,10 ;сравниваем содержимое
               ;регистра al с 10
    jg m1     ;если al>10 перейти на метку m1
    cmp al,0  ;сравниваем содержимое регистра al с 0
    jl m2     ;если al<0 перейти на метку m2
    ;вызываем процедуру для вычисления
    ;значения функции при 0<=al<=10
    call p3
    jmp m3
    ;вызываем процедуру для вычисления
    ;значения функции при al>10
m1: call p1
    jmp m3
    ;вызываем процедуру для вычисления
    ;значения функции при al<0
m2: call p2

```

```

m3: mov f,al ;завершение программы
    lea dx,mes
    mov ah,9
    int 21h
    mov ax, 4c00h
    int 21h
osn endp

```

;вычисляем значение функции при al>10

```

p1 proc near
    mov al,y
    imul y
    mov bl,3
    imul bl
    add al,4
    ret
p1 endp

```

;вычисляем значение функции при al<0

```

p2 proc near
    mov al,x
    imul x
    add al,3
    mov cl,al
    mov al,x
    imul x
    mov bl,al
    mov al,y
    imul y
    add al,bl
    sub al,5
    cbw
    idiv cl
    ret
p2 endp

```

;вычисляем значение функции при 0<=al<=10

```

p3 proc near
    mov bl,y
    add bl,5
    mov al,x
    imul x
    add al,bl
    mov bl,al
    mov al,y

```

```

mov bl,3
imul bl
sub al,x
cbw
idiv bl
ret
p3 endp
Cseg ends
end osn

```

3-й вариант – использование внешних процедур

**;Пример программы, вычисляющей значение
;ветвящейся функции.
;Ветви находятся во внешних процедурах**

```

;сегмент данных
Dseg segment para public 'data'
x db 1
y db 2
f db ?
mes db 'конец программы$'
Dseg ends
;сегмент стека
Sseg segment para stack 'stack'
dw 30 dup(0)
Sseg ends
public x,y
extrn p1:near, p2:near, p3:near
;объявление внешних процедур
;сегмент кода
Cseg segment para public 'code'
;основная программа
osn proc near
assume cs:cseg,ds:dseg,ss:sseg
mov ax,dseg
mov ds,ax
mov al,x
imul y
cmp al,10 ;сравниваем содержимое
;регистра al с 10
jg m1 ;если al>10 перейти на метку m1
cmp al,0 ;сравниваем содержимое регистра al с 0
jl m2 ;если al<0 перейти на метку m2
call p3
jmp m3
m1: call p1

```

```

jmp m3
m2: call p2
m3: mov f,al ;завершение программы
lea dx,mes
mov ah,9
int 21h
mov ax, 4c00h
int 21h
osn endp
Cseg ends
end osn

```

Каждая внешняя процедура должна находиться в отдельном внешнем файле.

```

;внешняя процедура p1
;должна находиться во внешнем файле p1.asm
;вычисляем значение функции при al>10
extrn x:byte,y:byte
public p1
Cseg segment para public 'code'
p1 proc near
assume cs:cseg
mov al,3
imul y
imul y
add al,4
ret
p1 endp
Cseg ends
end

```

```

;внешняя процедура p2
;должна находиться во внешнем файле p2.asm
;вычисляем значение функции при al<0
extrn x:byte,y:byte
public p2
Cseg segment para public 'code'
p2 proc near
assume cs:cseg
mov al,x
imul x
add al,3
mov cl,al
mov al,x

```

```

    imul x
    mov bl,al
    mov al,y
    imul y
    add al,bl
    sub al,5
    cbw
    idiv cl
    ret
p2 endp
Cseg ends
end

;внешняя процедура p3
;должна находиться во внешнем файле p3.asm
;вычисляем значение функции при 0<=a1<=10
extrn x:byte,y:byte
public p3
Cseg segment para public 'code'
p3 proc near
    assume cs:cseg
    mov bl,y
    add bl,5
    mov al,x
    imul x
    add al,bl
    mov bl,al
    mov al,y
    mov bl,3
    imul bl
    sub al,x
    cbw
    idiv bl
    ret
p3 endp
Cseg ends
end

```

При использовании внешних процедур, изменяется порядок работы с программой. Создадим исполняемый файл с расширением .bat, содержащий следующие команды работы с программой:

```

cmd
tasm lab3.asm
tasm p1.asm

```

```

tasm p2.asm
tasm p3.asm
tlink lab3.obj+p1.obj+p2.obj+p3.obj
td lab3.exe
pause

```

4-й вариант – подключение внешних процедур ввода и вывода

```

;Пример программы, вычисляющей значение
;ветвящейся функции.
;Ветви находятся во внешних процедурах.
;Используются внешние процедуры ввода и вывода
;сегмент данных
Dseg segment para public 'data'
    x db ?
    y db ?
    f db ?
    mes1 db 10,13,'$'
    mes db 'конец программы$'
    mes2 db 'Введите x-->$'
    mes3 db 'Введите y-->$'
    mes4 db 'f=$'
Dseg ends
;сегмент стека
Sseg segment para stack 'stack'
    dw 30 dup(0)
Sseg ends
    public x,y
    extrn p1:near, p2:near, p3:near,
        disp:near, vvod:near
;объявление внешних процедур
;сегмент кода
Cseg segment para public 'code'
;основная программа
osn proc near
    assume cs:cseg,ds:dseg,ss:sseg
    mov ax,dseg
    mov ds,ax
    lea dx,mes2 ;вывод сообщения 'Введите x-->'
    mov ah,9
    int 21h
    call vvod ;ввод значения x
    mov x,bl ;введенное значение из регистра bl
                ;заносим в переменную x

```



```

lea dx,mes3 ;вывод сообщения 'Введите у-->'
mov ah,9
int 21h
call vvod ;ввод значения у
mov y,b1 ;введенное значение из регистра b1
;заносим в переменную у
mov al,x ;вычисляем значение функции
imul y
cmp al,10 ;сравниваем содержимое р-ра al с 10
jg m1 ;если al>10 перейти на метку m1
cmp al,0 ;сравниваем содержимое р-ра al с 0
jl m2 ;если al<0 перейти на метку m2
call p3
jmp m3
m1: call p1
jmp m3
m2: call p2
m3: mov f,al ;вывод результатов
lea dx,mes4 ;вывод сообщения 'f='
mov ah,9
int 21h
mov al,f ;вывод значения f
cbw
call disp
lea dx,mes1 ;переход на новую строку
mov ah,9
int 21h
lea dx,mes ;вывод сообщения 'конец программы'
mov ah,9
int 21h
mov ax, 4c00h ;завершение работы программы
int 21h
osn endp
Cseg ends
end osn

```

Внешняя процедура vvod.asm используется для ввода значения длиной байт или слово. Введенное значение помещается в регистр bx.

;внешняя процедура vvod

;должна находится во внешнем файле vvod.asm

```

public vvod
dseg segment para public 'data'
mes db 'Переполнение'
mes1 db ',',10,13,'$'

```

```

mes2 db 'Ошибка ввода',10,13,'$'
dseg ends
cseg segment para public 'code'
assume cs:cseg,ds:dseg
vvod proc near
mov ax,dseg
mov ds,ax
push ax
push cx
push dx
push si
push di
push bp
maska=00001111b
mov bx,0
mov cx,10
mov si,0
mov di,1
mov bp,1
met4: mov ax,0800h
int 21h
cmp al,0dh
je met1
cmp si,0
jne met2
cmp al,2dh
jne met3
mov dl,2dh
mov di,0
met6: mov ax,0200h
int 21h
mov si,1
mov bp,0
jmp met4
met3: cmp al,2bh
jne met2
mov dl,2bh
jmp met6
met2: cmp al,30h
jae met5
met7: mov dl,07h
mov ax,0200h
int 21h
jmp met4
met5: cmp al,39h

```

```

ja met7
mov dl,al
mov ax,0200h
int 21h
mov bp,1
mov si,1
and al,maska
cbw
push ax
mov ax,bx
cwd
imul cx
jo met11
mov bx,ax
pop ax
cmp di,1
je met9
neg ax
met9: add bx,ax
      jno met4
met11:lea dx,mes1
      mov ax,0900h
      int 21h
      lea dx,mes
      mov ax,0900h
      int 21h
      jmp stop
met1:  cmp bp,1
      je met10
      lea dx,mes1
      mov ax,0900h
      int 21h
      lea dx,mes2
      mov ax,0900h
      int 21h
      jmp stop
met10:lea dx,mes1
      mov ax,0900h
      int 21h
stop:  pop bp
      pop di
      pop si
      pop dx
      pop cx
      pop ax

```

```

ret
vvod endp
cseg ends
end

```

Внешняя процедура disp.asm используется для вывода значения длинной строки, находящегося в регистре ax на экран.
;внешняя процедура disp вывода двоичного числа
;которое находится в регистре ax на экран

```

public disp
Dseg segment para public 'data'
tab db 6 dup(?)
Dseg ends
Code segment para public 'code'
      assume cs:code,ds:dseg
Disp  proc near
      push bx
      push si
      push cx
      push ax
Maska equ 00110000B
      mov si,0
      mov bx,10
      mov cx,0
      cmp ax,0
      jge metka
      mov tab[si], '-'
      inc si
      neg ax
metka:  cwd
      div bx
      or dl,maska
      mov tab[si],dl
      inc cx
      inc si
      cmp ax,0
      jne metka
      dec si
      pop ax
      cmp ax,0
      jge metka1
      mov dl,tab
      mov ax,200h
      int 21h

```

```

metka1: mov dl,tab[si]
        mov ax,200h
        int 21h
        dec si
        loop metka1
        pop cx
        pop si
        pop bx
        ret
disp endp
Code ends
end

```

При использовании внешних процедур ввода и вывода изменяется порядок работы с программой. Создадим исполняемый файл с расширением .bat, содержащий следующие команды работы с программой:

```

cmd
tasm lab4.asm
tasm p1.asm
tasm p2.asm
tasm p3.asm
tasm vvod.asm
tasm disp.asm
tlink lab4.obj+p1.obj+p2.obj+p3.obj+vvod.asm+disp.asm
lab4.exe
pause

```

5-й вариант – организация ассемблерных подпрограмм

Ассемблерные вставки можно использовать в программах высокого уровня, написанных на любых языках программирования. Приведем пример использования ассемблерных подпрограмм в языке программирования Pascal.

```

Program lab2_5;
uses crt;
var
    x,y,f:integer;
{вычисление функции при x*y<0}
Function P1(x,y:integer):integer; assembler;
asm
    mov ax,y
    imul y
    mov bx,3
    imul bx

```

```

        add ax,4
end;
{вычисление функции при x*y>10}
Function P2(x,y:integer):integer; assembler;
asm
    mov ax,x
    imul x
    add ax,3
    mov cx,ax
    mov ax,x
    imul x
    mov bx,ax
    mov ax,y
    imul y
    add ax,bx
    sub ax,5
    cwd
    idiv cx
end;
{вычисление функции при 0<=x*y<=10}
Function P3(x,y:integer):integer; assembler;
asm
    mov bx,y
    add bx,5
    mov ax,x
    imul x
    add ax,bx
    mov bx,ax
    mov ax,y
    mov bx,3
    imul bx
    sub ax,x
    cwd
    idiv bx
end;
begin
    clrscr;
    Write('Введите x,y -->');
    Readln(x,y);
    if x*y<0 then f:=p1(x,y)
        else if x*y>10 then f:=p2(x,y)
            else f:=p3(x,y);

    Writeln('f=',f);
end.

```

1.10 Практическое задание

Вычислить значение ветвящейся функции, используя различные модели структуры программы. Реализовать 5 вариантов структуры программы:

- 1) без использования внутренних процедур;
 - 2) с использованием внутренних процедур;
 - 3) с использованием внешних процедур;
 - 4) с использованием процедур ввода-вывода;
 - 5) с использованием ассемблерных подпрограмм.
- Варианты заданий даны в таблице 1.2.

Таблица 1.2 – Варианты заданий для вычисления значения ветвящейся функции

<p>Вариант 1</p> $y = \begin{cases} 6xy - 4y, & x + y > 9; \\ \frac{2xy + 3 + x}{x^2 + 3y^2 + 1}, & x + y < -1; \\ 3x^2 - 2y + 6, & -1 \leq x + y \leq 9. \end{cases}$	<p>Вариант 2</p> $y = \begin{cases} 4xy + 5, & x + y > 10; \\ \frac{3xy + 2y}{x^2 + y^2 + 1}, & x + y < -2; \\ 6x - 2y^2 + 1, & -2 \leq x + y \leq 1. \end{cases}$
<p>Вариант 3</p> $y = \begin{cases} 4xy + 4x, & xy > 8; \\ \frac{2xy + 3}{x^2 + 2y^2 - 1}, & xy < -12; \\ 3x^2 - 2y^2 + 1, & -12 \leq xy \leq 8. \end{cases}$	<p>Вариант 4</p> $y = \begin{cases} 2xy + 3, & x - y > 9; \\ \frac{3y^2}{x^2 + 2y + 1}, & x - y < -6; \\ 2x^2 - 2y + 1, & -6 \leq x - y \leq 9. \end{cases}$
<p>Вариант 5</p> $y = \begin{cases} \frac{y + 2x}{x^2 + 3y^2 + 1}, & x + y > 11; \\ \frac{4xy}{x^2 + 2y^2 + 1}, & x + y < -4; \\ 5x^2 + 2y^2, & -4 \leq x + y \leq 11. \end{cases}$	<p>Вариант 6</p> $y = \begin{cases} \frac{x^2 - 4y + 5}{x^2 + 2}, & x - y > 2; \\ \frac{2x + y}{3x^2 + 4}, & x - y < -4; \\ x^2 - 2y + 3, & -4 \leq x - y \leq 2. \end{cases}$
<p>Вариант 7</p> $y = \begin{cases} \frac{2x + y + 4}{xy^2 + 5}, & x - y > 8; \\ 2 + 3x^2 + y, & x - y < -2; \\ 5x^2 - 2y, & -2 \leq x - y \leq 8. \end{cases}$	<p>Вариант 8</p> $y = \begin{cases} \frac{1 + x}{5x^2 - y^2 + 1}, & x - y < -4; \\ 3x + 5y^2, & x - y > 6; \\ x^3 - 2y, & -4 \leq x - y \leq 6. \end{cases}$

<p>Вариант 9</p> $y = \begin{cases} \frac{x^2 + 2y}{x^2 + y^2 + 1}, & x - y > 15; \\ y^2 - x, & x - y < -5; \\ x^2 + 2y^2 + 4, & -5 \leq x - y \leq 15. \end{cases}$	<p>Вариант 10</p> $y = \begin{cases} \frac{x^2 + 3y}{x^2 + 2y^2 + 1}, & x + y > 6; \\ 3y^2 - 1, & x + y < -5; \\ 2x^2 + 3y^2 - 4, & -5 \leq x + y \leq 6. \end{cases}$
<p>Вариант 11</p> $y = \begin{cases} 3x + 2y^2 - 5, & xy > 8; \\ \frac{xy + 7}{2x^2 + 4}, & xy < -12; \\ 4x^2 - 3y^2 + 4, & -12 \leq xy \leq 8. \end{cases}$	<p>Вариант 12</p> $y = \begin{cases} 5x + 2xy + 1, & x + y > 9; \\ \frac{2xy + 3}{y^2 + 4}, & x + y < -5; \\ 3x^2 - 2y^2 + 1, & -5 \leq x + y \leq 9. \end{cases}$
<p>Вариант 13</p> $y = \begin{cases} \frac{x + 2y - 3}{x^2 + 1 + xy}, & xy > 12; \\ \frac{xy - 2y}{x^2 + 1}, & xy < -4; \\ x^2 + 3y^2, & -4 \leq xy \leq 12. \end{cases}$	<p>Вариант 14</p> $y = \begin{cases} \frac{2y + 3x}{x^2 + y^2 + 5}, & x - y > 12; \\ \frac{x - y}{x^2 + 2y^2 + 1}, & x - y < -4; \\ x^2 + 3y^2, & -4 \leq x - y \leq 12. \end{cases}$
<p>Вариант 15</p> $y = \begin{cases} 5xy - 3, & x + y > 8; \\ \frac{4xy^2}{x^2 + 1}, & x + y < -5; \\ 2x^2 - 4y^2 + 3, & -5 \leq x + y \leq 8. \end{cases}$	<p>Вариант 16</p> $y = \begin{cases} x^2 - 2y, & x + y \leq -3; \\ \frac{xy + 5}{x^2 + y^2 + 1}, & -3 < x + y < 5; \\ 2xy - 3y^2, & x + y \geq 5. \end{cases}$
<p>Вариант 17</p> $y = \begin{cases} 4x + y - 3, & x + y > 2; \\ \frac{3xy + 2y}{x^2 + 2y^2 + 1}, & x + y < -10; \\ 5x - 3y^2 + 7, & -10 \leq x + y \leq 2. \end{cases}$	<p>Вариант 18</p> $y = \begin{cases} 5xy + 4y, & x + y > 9; \\ \frac{xy - 5}{x^2 + 2y^2 - 1}, & x + y < -1; \\ 3x^2 - 2y + 6x, & -1 \leq x + y \leq 9. \end{cases}$
<p>Вариант 19</p> $y = \begin{cases} xy + 5x, & xy > 9; \\ \frac{2xy - 3}{x^2 + 2y^2 + 2}, & xy < -1; \\ 4x - y^2 + 8, & -1 \leq xy \leq 9. \end{cases}$	<p>Вариант 20</p> $y = \begin{cases} 7xy - 4, & xy > 6; \\ \frac{xy + 1}{x^2 + y^2 + 3}, & xy < -10; \\ 3x^2 - 2y^2 - 1, & -10 \leq xy \leq 6. \end{cases}$

<p>Вариант 21</p> $y = \begin{cases} 3xy - 1 + y, & x + y > 7; \\ \frac{x + y}{x^2 + 4}, & x + y < -10; \\ 3x^2 - 2y^2 + 1, & -10 \leq x + y \leq 7. \end{cases}$	<p>Вариант 22</p> $y = \begin{cases} 5xy - x^2, & x + y > 9; \\ \frac{4xy}{y^2 + 1}, & x + y < -4; \\ 3x^2 - y^2 + 4, & -4 \leq x + y \leq 9. \end{cases}$
<p>Вариант 23</p> $y = \begin{cases} \frac{5xy - y + 5}{x^2 + y^2}, & xy > 9; \\ \frac{2xy - 1}{y^2 + 1}, & xy < -2; \\ x^2 + y^2 - 6, & -2 \leq xy \leq 9. \end{cases}$	<p>Вариант 24</p> $y = \begin{cases} \frac{3xy - 1}{x^2 + 1}, & x + y > 8; \\ \frac{3y^2 + x^2}{y^2 + 1}, & x + y < -2; \\ x^2 + 4y^2 - 3, & -2 \leq x + y \leq 8. \end{cases}$
<p>Вариант 25</p> $y = \begin{cases} \frac{x^2 + 2y^2 - 3}{3 + 2x^2}, & x + y < 1; \\ 3x^2y + 4, & x + y > 10; \\ \frac{3y + 2x}{5 + y + x^2}, & 1 \leq x + y \leq 10. \end{cases}$	<p>Вариант 26</p> $y = \begin{cases} \frac{x + xy + 3y}{y^2 + 1}, & x - y > 10; \\ 2xy - y^2, & x - y < 0; \\ x^2 - y^2, & 0 \leq x - y \leq 10. \end{cases}$
<p>Вариант 27</p> $y = \begin{cases} 3x + 2y^2 - 1, & x + y > 8; \\ \frac{2xy + 3}{y^2 + 2x + 1}, & x + y < -5; \\ 2x^2 - 3y^2 + 2, & -5 \leq x + y \leq 8. \end{cases}$	<p>Вариант 28</p> $y = \begin{cases} 3x + 4y + 3, & x + y > 7; \\ \frac{3y - 2x^2}{y^2 + 4}, & x + y < -5; \\ 3x^2 - 4y^2 - 3, & -5 \leq x + y \leq 7. \end{cases}$
<p>Вариант 29</p> $y = \begin{cases} \frac{5xy}{x + y^2 + 1}, & xy < -5; \\ 2x - y, & xy > 8; \\ x + 3y, & -5 \leq xy \leq 8. \end{cases}$	<p>Вариант 30</p> $y = \begin{cases} \frac{x^2 - y^2 + 5}{3 + xy}, & x + y < -7; \\ xy - 1, & x + y > 10; \\ \frac{3y + x}{11y - x^2}, & -7 \leq x + y \leq 10. \end{cases}$

Тема 2. Программирование циклических вычислительных процессов. Обработка массивов

2.1 Команды управления циклами

Команда **loop**. Эта команда уменьшает содержимое регистра *cx* на 1 и передает управление оператору, помеченному меткой, если содержимое регистра *cx* ≠ 0. Завершение выполнения цикла происходит в том случае, если содержимое регистра *cx* уменьшается до нуля.

Формат команды:

loop метка

Пример:

```
mov cx, 10
```

```
start:...
```

```
; тело цикла
```

```
loop start
```

Команда **loopz (loopz)**. Эта команда уменьшает содержимое регистра *cx* на 1, а затем осуществляет переход, если содержимое регистра *cx* ≠ 0 или флаг нуля *zf* = 1. Повторение цикла завершается, если либо содержимое регистра *cx* = 0, либо флаг *zf* = 0, либо оба они равны 0. Команда **loopz** обычно используется для поиска первого ненулевого результата в серии операций. Синонимом команды **loopz** является команда **loopz**.

Команда **loopne (loopnz)**. Эта команда уменьшает содержимое регистра *cx* на 1, а затем осуществляет переход, если содержимое регистра *cx* ≠ 0 и флаг нуля *zf* = 0. Повторение цикла завершается, если либо содержимое регистра *cx* = 0, либо флаг *zf* = 1, либо будет выполнено и то и другое. Команда **loopne** обычно используется для поиска первого нулевого результата в серии операций. Синонимом команды **loopne** является команда **loopnz**.

2.2 Псевдооператоры описания переменных, используемые для описания массивов

Псевдооператор определения переменных можно использовать для создания в памяти массивов, перечисляя элементы массива через запятую. В одном псевдооператоре можно указать любое число значений, лишь бы они поместились в строке длиной 132 позиции.

Пример:

```
mas db 1,2,-4,5,6
```

Операция **dup** позволяет повторять операнды, не набирая их заново:

```
mas1 db 4 dup(5),1,2,5 dup(0)
```

При определении переменной без присваивания ей начального значения необходимо указать в поле выражения вопросительный знак:

```
db 4 dup(?)
```

Если имя переменной не указывается, то просто резервируется место в памяти

```
db 1,2,3,4,5
```

2.3 Команды пересылки адреса

Команда **lea**. Пересылает смещение ячейки памяти в любой 16-битовый регистр общего назначения, регистр указателя или индексный регистр.

Формат команды:

```
lea регистр_16, память_16
```

где операнд `память_16` должен иметь атрибут типа `word`.

Пример:

```
lea bx, mas  
lea bx, mas[si]
```

Команда **lds**. Эта команда загружает указатель с использованием регистра `ds`, т. е. считывает 32-битовое двойное слово и загружает 16 бит в заданный регистр, а следующие 16 бит – в регистр `ds`.

Формат команды:

```
lds регистр_16, память_32
```

где `регистр_16` – любой 16-битовый регистр общего назначения, `память_32` – ячейка памяти с атрибутом `dd`.

Команда **les**. Эта команда аналогична команде `lds`, но вместо регистра `ds` работает с регистром `es`.

Формат команды:

```
les регистр16, память32
```

2.4 Операции, возвращающие значения

Операция **\$** возвращает текущее значение счетчика адреса, т. е. смещение адреса текущего оператора относительно начала сегмента. С помощью этой операции можно определить число символов в строке или число элементов в массиве.

Пример:

```
mes db 'hello'  
len equ $-mes  
mas db 1,6,7,4,3,7  
count equ $-mas
```

Операции **seg** и **offset** возвращают адрес начала сегмента и смещение адреса переменной или метки внутри сегмента, где они находятся.

Пример:

```
mov ax, seg mas  
mov bx, offset mas
```

Последняя команда аналогична команде

```
lea bx, mas
```

Так как адрес начала сегмента и смещение имеют 16-битовые значения, то они должны загружаться в 16-битовые регистры.

Операция **type** возвращает числовое значение, определяющее тип атрибута переменной или тип атрибута дистанции меток. Для переменной операция `type` возвращает 1, если переменная имеет тип `byte`, 2, если – тип `word` и 4, если – тип `dword`. Для метки операция `type` возвращает –1, если она имеет атрибут `near`, и –2, если она имеет атрибут `far`.

Операция **length** возвращает число основных единиц памяти (байт, слов, двойных слов), распределенных в строке с определенной переменной.

Пример:

```
mas dw 5 dup(?)  
mov ax, length mas ;ax=5
```

Операция **size** сообщает, сколько байт памяти распределено при определении переменной

Пример:

```
mas dw 1,3,4,5  
mov cx, size mas ;cx=8
```

2.5 Операции присваивания атрибутов

Операции **high** и **low** возвращают соответственно старший и младший байты 16-битного выражения.

Пример:

```
const = 0abcdh
```

```
mov ah, high const ;ah=0abh
```

Операция указателя **ptr** позволяет изменить у операнда атрибут типа (byte или word) или атрибут дистанции (near или far). Она изменяет атрибут только в одной команде. Этой операцией можно воспользоваться для доступа к байтам в таблице слов. Если таблица определена оператором

```
table dw 100 dup(?)
```

то оператор

```
mov cl, byte ptr table
```

перешлет в cl содержимое первого байта таблицы table.

Операцию ptr можно использовать для изменения атрибута дистанции. Если программа содержит оператор

```
start: mov cx, 5
```

то метка start имеет атрибут near. Это позволяет сослаться на нее командой jmp, находящейся в том же сегменте. Чтобы сослаться на метку могли команды jmp, находящиеся в других сегментах, надо дать приведенному оператору альтернативную метку, имеющую атрибут far. Это можно сделать оператором

```
far_start equ far ptr start
```

2.6 Режимы адресации

Под режимами адресации понимаются способы доступа к данным. Все режимы адресации можно условно разделить на 7 групп.

Для доступа к операнду используется 20-битовый физический адрес.

Физический адрес операнда получается сложением значения смещения адреса операнда с содержимым сегментного регистра, предварительно дополненного четырьмя нулями.

Смещение адреса операнда называется **исполнительным адресом**. Исполнительный адрес показывает, на каком расстоянии в байтах располагается операнд от начала сегмента, в котором он находится. Будучи 16-битовым числом без знака, исполнительный адрес позволяет получить доступ к операндам, находящимся выше начала сегмента на расстоянии 64 Кбайт.

При работе с адресацией операндов необходимо помнить, что микропроцессор хранит 16-битовые числа в обратном порядке, а именно: младшие биты числа в байте с меньшим адресом.

Регистровая. Этот режим предполагает, что микропроцессор извлекает операнд из регистра или загружает его в регистр.

```
mov cx, ax
```

```
inc di
```

Непосредственная. Этот режим адресации позволяет указывать 8- или 16-битовые значения константы в операнде-источнике.

```
mov cx, 100
```

```
mov al, -10
```

Непосредственный операнд может быть идентификатором, определенным операторами equ или =

```
k equ 100
```

```
...
```

```
mov cx, k
```

Прямая. При прямой адресации исполнительный адрес является составной частью команды. Микропроцессор добавляет этот исполнительный адрес к сдвинутому содержимому регистра данных ds и получает 20-битовый физический адрес. По этому адресу и находится операнд. Обычно прямая адресация применяется, если операндом служит метка переменной.

```
mov ax, table
```

Косвенная регистровая адресация. В этом случае исполнительный адрес операнда содержится в базовом регистре bx, регистре указателя базы bp, регистре sp или индексных регистрах si и di.

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

Смещение адреса в регистр можно поместить оператором offset или lea

```
mov bx, offset table
```

```
lea bx, table
```

Адресация по базе. Ассемблер вычисляет исполнительный адрес с помощью сложения значения сдвига с содержимым регистров bx или bp. Адресация по базе используется при доступе к структурированным записям данных, расположенных в разных участках памяти. В этом случае базовый адрес записи помещается в базовый регистр bx или bp и доступ к отдельным его элементам записи осуществляется по сдвигу относительно базы. А для доступа к разным записям одной и той же структуры достаточно соответствующим образом изменить содержимое базового регистра.

```
mas db 1,2,3,4,5,6,7,8,9,10
```

```
mov bx, offset mas
```

```
mov al, [bx+4] ;загрузка в al mas[4]=5
```

Прямая адресация с индексированием. Исполнительный адрес вычисляется как сумма значений смещения и индексного регистра di или si. Этот тип адресации удобен для доступа к элементам таблицы,

когда смещение указывает на начало таблицы, а индексный регистр на номер элемента.

```
table db 10 dup (?)
mov di, 5
mov al, table[di] ;загрузка 6 элемента таблицы
```

Адресация по базе с индексированием. Исполнительный адрес вычисляется как сумма значений базового регистра, индексного регистра и сдвига. Этот режим адресации удобен при адресации двумерного массива. Пусть задан двумерный массив A :

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}.$$

;выделение памяти под элементы матрицы

```
A db 1,2,3,4,5,6,7,8,9,10,11,12
```

;доступ к элементу $a[1,2]=7$ двумерного массива

```
mov bx, offset a
mov di, 4
mov al, [bx][di+2]
```

Допускаются следующие формы записи адресации по базе с индексированием:

```
mov al, [bx+2+di]
mov al, [di+bx+2]
mov al, [bx+2][di]
```

Из семи режимов адресации самыми быстрыми являются регистровая и непосредственная адресации операндов. В других режимах адресация выполняется дольше, так как вначале необходимо вычислить адрес ячейки памяти, извлечь операнд, а затем передать его операционному блоку.

2.7 Примеры обработки одномерных массивов

Пример 1. Найти минимальный элемент вектора.

```
Dseg segment para public 'data'
mas db -1,3,5,2,-7
n dw 5
min db ?
Dseg ends
Sseg segment para stack 'stack'
dw 30 dup(0)
Sseg ends
```

```
Cseg segment para public 'code'
osn proc near
assume cs:cseg,ds:dseg,ss:sseg
mov ax,dseg
mov ds,ax
mov cx,n ;cx=n
mov al,mas ;al=mas[0]
mov di,0 ;di=0
start: cmp al,mas[di]
jle met
mov al,mas[di]
met: inc di
loop start
mov min,al
mov ax,4c00h
int 21h
osn endp
cseg ends
end osn
```

Пример 2. Найти сумму положительных элементов вектора.

```
Dseg segment para public 'data'
mas db -1,3,5,2,-7
n dw 5
sum db ?
Dseg ends
Sseg segment para stack 'stack'
dw 30 dup(0)
Sseg ends
Cseg segment para public 'code'
osn proc near
assume cs:cseg,ds:dseg,ss:sseg
mov ax,dseg
mov ds,ax
mov cx,n
mov al,0
mov di,0
start: cmp mas[di],0
jle met
add al,mas[di]
met: inc di
loop start
mov sum,al
mov ax,4c00h
int 21h
```



```
osn endp
cseg ends
end osn
```

Пример 3. Найти произведение элементов вектора, кратных 5.

```
Dseg segment para public 'data'
mas db -1,3,5,2,10
n dw 5
five db 5
prod db ?
Dseg ends
Sseg segment para stack 'stack'
dw 30 dup(0)
Sseg ends
Cseg segment para public 'code'
osn proc near
assume cs:cseg,ds:dseg,ss:sseg
mov ax,dseg
mov ds,ax
mov cx,n
mov bl, 1
mov di,0
start: mov al,mas[di]
cbw
idiv five
cmp ah,0
jne met
mov al, mas[di]
imul bl
mov bl,al
met: inc di
loop start
mov prod, bl
mov ax, 4c00h
int 21h
osn endp
cseg ends
end osn
```

Пример 4. Найти сумму элементов вектора, которые при делении на 5 дают остаток 1.

```
Dseg segment para public 'data'
mas db -1,3,5,2,-7
n dw 5
```

```
sum db ?
five db 5
Dseg ends
Sseg segment para stack 'stack'
dw 30 dup(0)
Sseg ends
Cseg segment para public 'code'
osn proc near
assume cs:cseg,ds:dseg,ss:sseg
mov ax,dseg
mov ds,ax
mov cx,n
mov bl, 0
mov si,0
start: mov al, mas[si]
cbw
idiv five
cmp ah,1
jne met
add bl, mas[si]
met: inc si
loop start
mov sum, bl
mov ax, 4c00h
int 21h
osn endp
cseg ends
end osn
```

Пример 5. Поменять местами первый и минимальный элементы вектора.

```
Dseg segment para public 'data'
mas db -1,3,5,2,-7
n dw 5
min db ?
imin dw ?
Dseg ends
Sseg segment para stack 'stack'
dw 30 dup(0)
Sseg ends
Cseg segment para public 'code'
osn proc near
assume cs:cseg,ds:dseg,ss:sseg
mov ax,dseg
```

```

mov ds,ax
mov cx,n      ;cx=n
mov al,mas    ;al=mas[0]
mov si,0
mov di,0      ;di=0
start: cmp al,mas[di]
      jle met
      mov al,mas[di]
      mov si,di
met: inc di
     loop start
     mov min,al
     mov imin,si
     mov di,0
     mov bl,mas[di]
     mov mas[di],al
     mov mas[si],bl
     mov ax,4c00h
     int 21h
osn endp
cseg ends
end osn

```

Пример 6. Сортировка элементов вектора по возрастанию методом стандартного обмена.

Код программы на языке Assembler

```

Dseg segment para public 'data'
  mas db 5,4,3,2,1
  n dw 5
Dseg ends
Sseg segment para stack 'stack'
  dw 30 dup(0)
Sseg ends
Cseg segment para public 'code'
osn proc near
assume cs:cseg,ds:dseg,ss:sseg
  mov ax,dseg
  mov ds,ax
  mov cx,n
Cikl2: push cx
      mov cx,n
      dec cx

```

Код программы на языке Pascal

```

For i=1 to n do
For j:=1 to n-1 do
If a[j]>a[j+1] then
  Begin

```

<pre> mov si,0 ;si=j cikl1: mov di,si inc di ;di=j+1 mov al,mas[si] mov bl,mas[di] cmp al,bl jle met1 mov mas[si],bl mov mas[di],al met1: inc si loop cikl1 pop cx loop cikl2 mov ax,4c00h int 21h osn endp cseg ends end osn </pre>	<pre> R:=a[j]; A[j]:=a[j+1]; A[j+1]:=r; End; </pre>
---	---

Пример 7. Найти максимальный элемент вектора. Размерность вектора и элементы вектора вводятся с клавиатуры, результаты выводятся на экран.

```

extrn vvod:near,disp:near
Dseg segment para public 'data'
  mas db 10 dup (?)
  max db ?
  mes1 db 'Введите n=$'
  mes2 db 'mas[$'
  mes3 db ']=$'
  mes4 db 'Исходный вектор$'
  mes5 db 10,13,'$' ; переход на новую строку
  mes6 db '_$'      ; пробел
  mes7 db 'Максимальный элемент=$'
  n dw ?
Dseg ends
Sseg segment para stack 'stack'
  db 30 dup(0)
Sseg ends
Cseg segment para public 'code'
osn proc near
  assume cs:cseg,ds:dseg,ss:sseg
  mov ax,dseg
  mov ds,ax
;ввод вектора

```

```

;очистка экрана
    mov ax,0002h
    int 10h
;ввод размерности вектора n
    lea dx,mes1
    mov ax,0900h
    int 21h
    call vvod
    mov n,bx
;ввод элементов вектора
    mov cx,n
    mov si,0
zikl1:
    lea dx,mes2 ;вывод mas[
    mov ax,0900h
    int 21h
    mov ax,si
    call disp
    lea dx,mes3 ;вывод ]=
    mov ax,0900h
    int 21h
    call vvod
    mov mas[si],bl
    inc si
    loop zikl1
;вывод вектора
    lea dx,mes4 ;исходный вектор
    mov ax,0900h
    int 21h
    lea dx,mes5 ;переход на новую строку
    mov ax,0900h
    int 21h
    mov cx,n
    mov si,0
zikl2:
    mov al, mas[si]
    cbw
    call disp
    lea dx,mes6 ;пробел
    mov ax,0900h
    int 21h
    inc si
    loop zikl2
;нахождение максимального элемента
    mov cx,n

```

```

    mov si,0
    mov al,mas
zikl3:
    cmp al,mas[si]
    jg m1
    mov al, mas[si]
m1: inc si
    loop zikl3
    mov max, al
;вывод максимального элемента
    lea dx,mes5 ;переход на новую строку
    mov ax,0900h
    int 21h
    lea dx,mes7 ;максимальный элемент
    mov ax,0900h
    int 21h
    mov al,max
    cbw
    call disp
;завершение программы
    mov ax,4c00h
    int 21h
osn endp
Cseg ends
end osn

```

2.8 Примеры обработки двумерных массивов

Специальных средств для описания двумерных массивов в ассемблере нет. Двухмерный массив нужно моделировать. Память под массив выделяется с помощью директив резервирования и инициализации памяти.

Непосредственно моделирование обработки массива производится в сегменте кода, где программист определяет, что некоторую область памяти необходимо трактовать как двухмерный массив. При этом данную область памяти можно рассматривать как элементы двухмерного массива, расположенного по строкам или по столбцам.

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

(база + (количество_элементов_в_строке $\cdot i + j$) \cdot размер_элемента).

Здесь $i = 0..n - 1$ – номер строки, $j = 0..m - 1$ – номер столбца.

Пусть имеется массив чисел (размером в 1 байт) $mas(i, j)$ размерностью 4×4 ($i = 0 \dots 3, j = 0 \dots 3$):

$$mas = \begin{pmatrix} 23 & 04 & 05 & 67 \\ 05 & 06 & 07 & 99 \\ 67 & 08 & 09 & 23 \\ 87 & 09 & 00 & 08 \end{pmatrix}$$

В памяти элементы этого массива будут расположены в такой последовательности:

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

Если мы хотим трактовать эту последовательность как двухмерный массив и извлечь, например, элемент $mas(2, 3) = 23$, то

эффективный адрес $mas(2, 3) = mas + (4 \cdot 2 + 3) \cdot 1 = mas + 11$.

По этому смещению действительно находится нужный элемент массива.

Логично организовать адресацию двухмерного 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[bx][si]
```

– сочетание двух индексных регистров, один из которых является и базовым, и индексным одновременно, а другой – только индексным:

```
mov ax, [bx][si]
```

Пример 1. Фрагмент программы выборки элемента массива $mas(2,3)$.

```
.data
mas db 23,4,5,67,5,6,7,99,67,8,9,23,87,9,0,8
i = 2
j = 3
el_size=1
.code
...
mov si, 4*el_size*i
mov di, j*el_size
mov al,mas[si][di]; в al элемент mas(2,3)
```

Пример 2. Доступ к элементам массива с использованием различных вариантов записи адресации по базе.

```
Dseg segment para public 'data'
mas db 1,2,3,4,5,6,7,8,9,10
Dseg ends
```

```
Sseg segment para stack 'stack'
db 30 dup(0)
Sseg ends
Cseg segment para public 'code'
osn proc near
assume cs:cseg,ds:dseg,ss:sseg
mov ax,dseg
mov ds,ax
...
;1 способ
mov bx,offset mas
mov al,[bx+4] ;загрузка в al mas[4]=5
;2 способ
mov bx, offset mas
mov di,4
mov al,[bx][di+2] ;загрузка в al mas[6]=7
;вывод значения на экран
mov ah,02h
;функция вывода значения из dx на экран
mov dx,di
add dx,0030h ;преобразование числа в символ
int 21h
...
mov ax,4c00h
int 21h
osn endp
Cseg ends
end osn
```

Пример 3. Дана прямоугольная матрица. Построить вектор из сумм элементов каждой строки.

```
extrn vvod:near,disp:near
Dseg segment para public 'data'
mas db 10 dup (?)
sum db 5 dup(0)
mes1 db 'n=$'
mes2 db 'm=$'
mes3 db 'mas[$'
mes4 db ', $'
mes5 db ']= $'
mes6 db 'Исходный массив$'
mes7 db 10,13, '$'
mes9 db '$'
mes10 db 'Полученный вектор$'
```

```

    n dw ?
    m dw ?
Dseg ends
Sseg segment para stack 'stack'
    db 30 dup(0)
Sseg ends
Cseg segment para public 'code'
osn proc near
    assume cs:cseg,ds:dseg,ss:sseg
    mov ax,dseg
    mov ds,ax
;ВВОД n
    lea dx,mes1
    mov ax,0900h
    int 21h
    call vvod
    mov n,bx
;ВВОД m
    lea dx,mes2
    mov ax,0900h
    int 21h
    call vvod
    mov m,bx
;ВВОД элементов массива
    mov cx,n ;число для внешнего цикла по строкам
    mov si,0 ;строки в матрице
    mov bx,offset mas ;строки в матрице (смещение
;адреса переменной внутри сегмента)
zikl_i:
    push cx ;сохраняем содержимое р-ра cx в стеке
    mov cx,m ;число для внутреннего цикла
; (по столбцам)
    mov di,0 ;столбцы в матрице
zikl_j:
    lea dx,mes3 ;mas[
    mov ax,0900h
    int 21h
    mov ax,si
    call disp
    lea dx,mes4 ; ,
    mov ax,0900h
    int 21h
    mov ax,di
    call disp
    lea dx,mes5 ;]=

```

```

    mov ax,0900h
    int 21h
    push bx
    call vvod
    mov dl,bl
    pop bx
    mov mas[bx][di],dl ;первый способ обращения к
;элементам двумерного массива
    inc di
    loop zikl_j
    pop cx
    add bx,m ;увеличиваем на кол-во эл-тов в строке
    inc si
    loop zikl_i
;Вывод элементов массива
    lea dx,mes6 ;Исходная матрица
    mov ax,0900h
    int 21h
    lea dx,mes7 ;перевод курсора на начало строки
    mov ax,0900h
    int 21h
    mov cx,n ;число для внешнего цикла по строкам
    mov si,0 ;строки в матрице
    mov bx,offset mas ;строки в матрице (смещение
;адреса переменной внутри сегмента)
zikl_il:
    push cx ;сохраняем содержимое р-ра cx в стеке
    mov cx,m ;число для внутреннего цикла
;ж(по столбцам)
    mov di,0 ;столбцы в матрице
zikl_jl: mov al,[bx][di] ;второй способ обращения
;к эл-там двумерного массива
    cbw
    call disp
    lea dx,mes9 ;пробел
    mov ax,0900h
    int 21h
    inc di
    loop zikl_jl
    pop cx
    add bx,m ;увеличиваем на кол-во эл-тов в строке
    inc si
    lea dx,mes7 ;перевод курсора на новую строку
    mov ax,0900h
    int 21h

```

```

loop zikl_i1
;нахождение суммы элементов каждой строки
mov cx,n ;число для внешнего цикла по строкам
mov si,0 ;строки в матрице
mov bx,offset mas ;строки в матрице (смещение
;адреса переменной внутри сегмента)
zikl_i2:
push cx ;сохраняем содержимое p-ра cx в стеке
mov cx,m ;число для внутреннего цикла
; (по столбцам)
mov di,0 ;столбцы в матрице
mov al,0
zikl_j2:
add al,[bx+di] ;третий способ обращения к эл-там
;двумерного массива
inc di
loop zikl_j2
mov sum[si],al
pop cx
add bx,m ;увеличиваем на кол-во эл-тов в строке
inc si
loop zikl_i2
;вывод полученного вектора
lea dx,mes10 ;Полученный вектор
mov ax,0900h
int 21h
lea dx,mes7 ;перевод курсора на начало строки
mov ax,0900h
int 21h
mov cx,n ;число для внешнего цикла по строкам
mov si,0 ;строки в векторе
zikl_i3:
mov al,sum[si]
mov ah,0
call disp
lea dx,mes9 ;пробел
mov ax,0900h
int 21h
inc si
loop zikl_i3
mov ax,4c00h ;завершение программы
int 21h
osn endp
Cseg ends
end osn

```

2.9 Практические задания

Задание 1. Написать программу обработки и сортировки вектора. Размерность и элементы вектора вводятся с клавиатуры, результаты выводятся на экран. Метод сортировки – любой.

Варианты

1 Если максимальный элемент вектора является нечетным числом, то отсортировать элементы вектора по возрастанию.

2 Если минимальный элемент вектора является четным числом, то отсортировать элементы вектора по убыванию.

3 Если максимальный элемент вектора является первым элементом вектора, то отсортировать элементы вектора по возрастанию.

4 Если минимальный элемент вектора является последним элементом вектора, то отсортировать элементы вектора по убыванию.

5 Если сумма всех элементов вектора больше нуля, то отсортировать элементы вектора по возрастанию.

6 Если сумма всех элементов вектора меньше нуля, то отсортировать элементы вектора по убыванию.

7 Если количество положительных элементов вектора больше количества отрицательных элементов вектора, то отсортировать элементы вектора по возрастанию.

8 Если количество отрицательных элементов вектора меньше 5, то отсортировать элементы вектора по убыванию.

9 Если все элементы вектора четные числа, то отсортировать элементы вектора по возрастанию.

10 Если все элементы вектора нечетные числа, то отсортировать элементы вектора по убыванию.

11 Если произведение всех элементов вектора больше нуля, то отсортировать элементы вектора по возрастанию.

12 Если в векторе есть хотя бы один отрицательный элемент, то отсортировать элементы вектора по убыванию.

13 Если сумма четных элементов вектора меньше последнего элемента вектора, то отсортировать элементы вектора по возрастанию.

14 Если сумма нечетных элементов вектора больше первого элемента вектора, то отсортировать элементы вектора по убыванию.

15 Если среднее арифметическое значение первого и последнего элемента меньше 8, то отсортировать элементы вектора по возрастанию.

16 Если среднее арифметическое значение второго и предпоследнего элементов больше 6, то отсортировать элементы вектора по убыванию.

17 Если среднее арифметическое значение положительных элементов меньше 7, то отсортировать элементы вектора по возрастанию.

18 Если среднее арифметическое значение отрицательных элементов вектора больше 5, то отсортировать элементы вектора по убыванию.

19 Если последний элемент вектора четное число, то отсортировать элементы вектора по возрастанию.

20 Если первый элемент вектора нечетное число, то отсортировать элементы вектора по убыванию.

21 Если количество максимальных элементов вектора равно 3, то отсортировать элементы вектора по возрастанию.

22 Если количество минимальных элементов вектора равно 2, то отсортировать элементы вектора по убыванию.

23 Если в векторе есть 3 элемента, принадлежащие интервалу $[3, 7]$, то отсортировать элементы вектора по возрастанию.

24 Если в векторе нет элементов, принадлежащих интервалу $[4, 8]$, то отсортировать элементы вектора по убыванию.

25 Если все четные элементы вектора стоят на нечетных местах, то отсортировать элементы вектора по возрастанию.

26 Если все нечетные элементы вектора стоят на четных местах, то отсортировать элементы вектора по убыванию.

27 Если все положительные элементы вектора четные числа, то отсортировать элементы вектора по возрастанию.

28 Если все отрицательные элементы вектора нечетные числа, то отсортировать элементы вектора по убыванию.

29 Если максимальный элемент вектора четное число, то отсортировать элементы вектора по возрастанию.

30 Если минимальный элемент вектора нечетное число, то отсортировать элементы вектора по убыванию.

Задание 2. Написать программу обработки квадратной матрицы A размерности $(n \times n)$. Размерность и элементы матрицы вводятся с клавиатуры, результаты выводятся на экран.

Варианты

1 Построить вектор B размерности n , где b_i равно количеству положительных элементов i -й строки матрицы A .

2 Найти максимальный элемент матрицы и поменять его местами с элементом $A(1,1)$.

3 Изменить все элементы матрицы на противоположные по знаку, если они меньше 5, и оставить без изменений в противном случае.

4 Получить новую матрицу путем умножения всех элементов первоначальной на наименьший положительный элемент матрицы.

5 Построить вектор B , где $b_i = -10$, если в строке с номером i матрицы A число положительных элементов не меньше числа отрицательных элементов, и $b_i = -1$ – в противном случае.

6 Построить вектор B размерности n , присвоив i -му элементу вектора B значение 1, если в строке с номером i матрицы A сумма всех элементов положительна; -1 , если сумма всех элементов отрицательна; 0, если сумма всех элементов равно 0.

7 Получить новую матрицу путем деления всех элементов на наибольший элемент матрицы.

8 Переменной t присвоить значение, равное сумме элементов строки, в которой находится максимальный элемент матрицы.

9 Переменной t присвоить значение, равное сумме элементов столбца, в котором находится минимальный элемент матрицы.

10 Найти скалярное произведение строки и столбца, на пересечении которых находится наименьший элемент матрицы.

11 В каждом столбце матрицы оставить без изменения минимальный и максимальный элементы. Остальные заменить единицами.

12 В каждой строке матрицы оставить без изменения минимальный и максимальный элементы. Остальные заменить нулями.

13 В каждой строке матрицы поменять местами минимальный и максимальный элементы строки.

14 В каждом столбце матрицы поменять местами минимальный и максимальный элементы столбца.

15 Найти сумму положительных элементов, расположенных выше главной диагонали.

16 Найти число четных положительных элементов, расположенных ниже главной диагонали.

17 Найти максимальный элемент матрицы на главной диагонали.

18 Найти минимальный элемент матрицы на побочной диагонали.

19 Поменять местами минимальный и максимальный элементы матрицы.

20 Поменять местами максимальный и минимальный элементы на главной диагонали.

21 Поменять местами максимальный и минимальный элементы на побочной диагонали.

22 Найти номер строки с максимальной суммой элементов.

23 Построить вектор B , i -й элемент которого равен количеству четных элементов i -й строки матрицы.

24 Построить вектор B , i -й элемент которого равен количеству нечетных элементов i -го столбца матрицы.

25 Построить вектор B , i -й элемент которого равен максимальному элементу i -й строки матрицы.

26 Построить вектор B , i -й элемент которого равен минимальному элементу i -го столбца матрицы.

27 Построить вектор B , i -й элемент которого равен сумме максимального и минимального элементов i -й строки матрицы.

28 Найти сумму четных элементов выше главной диагонали.

29 Найти количество отрицательных элементов матрицы на побочной диагонали.

30 Найти произведение элементов, стоящих на главной диагонали.

Литература

- 1 Гук, М. Процессоры Pentium III, Athlon и другие / М. Гук, В. Юров. – СПб. : Питер, 2000. – 379 с.
- 2 Зубков, С. В. Ассемблер для DOS, Windows и UNIX / С. В. Зубков. – М. : ДМК Пресс, 2000. – 534 с.
- 3 Программирование на языке ассемблера для персональных ЭВМ : учебное пособие / А. Ф. Каморников [и др.]. – Гомель : ГГУ, 1995. – 95 с.
- 4 Пустоваров, В. И. Ассемблер : программирование и анализ корректности машинных программ / В. И. Пустоваров. – К. : Издательская группа ВНУ, 2000. – 480 с.
- 5 Сван, Т. Освоение Turbo Assembler / Т. Сван. – Киев : Диалектика, 1996. – 540 с.
- 6 Юров, В. Assembler / В. Юров. – СПб. : Питер, 2001. – 624 с.
- 7 Юров, В. Assembler : практикум / В. Юров. – СПб. : Питер, 2002. – 400 с.
- 8 Юров, В. Assembler : специальный справочник / В. Юров. – СПб. : Питер, 2000. – 496 с.

Производственно-практическое издание

Ружицкая Елена Адольфовна,
Осипова Ольга Геннадьевна,
Ковалёва Виктория Александровна

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ASSEMBLER

**Программирование ветвящихся
и циклических вычислительных процессов,
обработка массивов**

Практическое пособие

Редактор *В. И. Шкредова*
Корректор *В. В. Калугина*

Подписано в печать 19.01.2016. Формат 60x84 1/16.
Бумага офсетная. Ризография. Усл. печ. л. 2,8.
Уч-изд. л. 3,1. Тираж 25 экз. Заказ 31.

Издатель и полиграфическое исполнение:
учреждение образования
«Гомельский государственный университет
имени Франциска Скорины».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 1/87 от 18.11.2013.
Специальное разрешение (лицензия) № 02330 / 450 от 18.12.2013.
Ул. Советская, 104, 246019, Гомель.