МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

# Кафедра электронных вычислительных машин

Отчет по предмету

Программирование на языке ассемблера

Лабораторная работа №3

**«Целочисленные арифметические операции. Обработка массивов числовых данных»**

|  |  |
| --- | --- |
| **Выполнил:**  Студент группы 230501  Кочеров Р.С. | **Проверил:**  Туровец Н.О. |

Минск 2023

**Цель работы**: Ознакомиться с арифметическими операциями над целочисленными данными, обработкой массивов чисел, ознакомиться с правилами оформления ассемблерных процедур.

**Теоретические сведения:**

1. Арифметические операции над целыми числами.

Арифметические операции над целыми числами в двоичной арифметике выполняются с помощью следующих команд:

-- команды сложения (ADD, ADC):

*ADD приемник, источник*

-- команды вычитания (SUB, SBB):

*SUB приемник, источник*

-- команды умножения (MUL, IMUL):

*MUL источник*

-- команды деления (DIV, IDIV):

*DIV источник*

1. Логические побитовые операции над целыми числами.

Логические побитовые операции над целыми числами выполняются с помощью следующих команд:

-- логическое И (AND):

*AND приемник, источник*

-- логическое ИЛИ (OR):

*OR приемник, источник*

-- логическое исключающее ИЛИ (XOR):

*XOR приемник, источник*

-- инверсия (NOT):

*NOT приемник*

1. Процедуры.

Процедура в ассемблере – это аналог функции C, процедур и функций PASCAL и т.п. Ассемблер не накладывает на процедуры никаких ограничений – на любой адрес программы можно передать управление командой CALL, и оно вернется к вызвавшей процедуре, как только встретится команда RET.

Описание операндов PROC:

-- метка – название процедуры.

-- тип может принимать значения NEAR и FAR, и если он указан, все команды RET в теле процедуры будут заменены соответственно на RETN и RETF. По умолчанию подразумевается, что процедура имеет тип NEAR в моделях памяти TINY, SMALL и COMPACT.

-- язык действует аналогично такому же операнду директивы .MODEL, определяя взаимодействие процедуры с языками высокого уровня. В некоторых ассемблерах директива PROC позволяет также считать параметры, передаваемые вызывающей программой. В этом случае указание языка необходимо, так как различные языки высокого уровня используют разные способы передачи параметров.

-- USES регистры – список регистров, значения которых изменяет процедура. Ассемблер помещает в начало процедуры набор команд PUSH, а перед командой RET – набор команд POP, так что значения перечисленных регистров будут восстановлены.

**Текст программы:**

.model small

.stack 100h

.data

strError db 'ERROR$'

greeting db 'Input amount of numbers:$'

strInput db 'Input a number from -32768 to 32767:$'

strRepeat db 'Repeat input!!!$'

result db 'Result:$'

enter db 10, 13, '$'

massiv dw 30 dup (?)

i db 7,8 dup(?)

povtorenie1 db ?

povtorenie2 db ?

lenght dw ?

znak db '-','$'

number dw ?

.code

input proc near

mov ah, 0ah

mov dx, offset i

int 21h

ret

input endp

perevod proc near

xor ax, ax

xor dx, dx

lodsb

push ax

cmp ax, '-'

jne cycle

dec cx

lodsb

cycle:

cmp ax, '0'

jb error

cmp ax, '9'

ja error

sub ax,'0'

push ax

mov ax, massiv[di]

jo error2

mul bx

jo error2

mov massiv[di], ax

jo error2

pop ax

add massiv[di], ax

jo error

lodsb

loop cycle

pop ax

cmp ax, '-'

je minus

jmp endPerevod

error:

pop ax

cmp ax, '-'

je minus

error1:

mov ah, 09h

lea dx, enter

int 21h

mov ah, 09h

lea dx, strError

int 21h

mov dx, 1

jmp endPerevod

error2:

pop ax

pop ax

mov ah, 09h

lea dx, enter

int 21h

mov ah, 09h

lea dx, strError

int 21h

mov dx, 1

jmp endPerevod

minus:

not massiv[di]

add massiv[di], 1

cmp massiv[di], -32768

js error1

jmp endPerevod

endPerevod:

ret

perevod endp

seach proc near

mov povtorenie1, 0

mov cx, lenght

go:

xor ax, ax

mov ax, massiv[si]

repne scasw

jne my\_ret

inc povtorenie1

mov dl, povtorenie1

jcxz my\_ret

jmp go

my\_ret:

inc si

inc si

ret

seach endp

DecToASCII proc near

@b:

div bx

add dx, '0'

push dx

inc cx

xor dx, dx

cmp ax, 0

jne @b

@a:

pop bx

mov i[si+2], bl

inc si

loop @a

mov i[si+2], '$'

ret

DecToASCII endp

main:

mov ax,@data

mov ds, ax

mov es, ax

start:

mov ah, 09h

lea dx, enter

int 21h

mov massiv[di], 0

mov di, 0

mov ah, 09h

lea dx, greeting

int 21h

call input

lea si, i+2

mov bx, 10

xor cx, cx

mov cl, i[1]

call perevod

add cx, massiv[di]

js start

xor cx, cx

mov ax, massiv[di]

mov massiv[di], 0

mov lenght, ax

mov bx, 30

cmp bx, lenght

jl start

mov bx, 0

cmp bx, lenght

je start

mov cx, lenght

mov di, 0

mov povtorenie2, 0

jmp main1

povtor:

mov ah, 09h

lea dx, enter

int 21h

mov ah, 09h

lea dx, strRepeat

int 21h

dec di

dec di

mov massiv[di], 0

main1:

push cx

mov ah, 09h

lea dx, enter

int 21h

mov ah, 09h

lea dx, strInput

int 21h

call input

lea si, i+2

mov bx, 10

mov cl, i[1]

call perevod

pop cx

inc di

inc di

cmp dx, 1

je povtor

loop main1

mov cx, lenght

mov povtorenie1, 0

xor si, si

seach1:

push cx

mov di, offset massiv

call seach

mov bl,povtorenie2

mov bh,povtorenie1

cmp bh, bl

jg swap

pop cx

loop seach1

jcxz exit

swap:

mov bl, bh

mov povtorenie2, bl

pop cx

mov number, ax

loop seach1

jcxz exit

exit:

mov ah, 09h

lea dx, enter

int 21h

mov ah, 09h

lea dx, result

int 21h

xor ax,ax

add ax, number

js step2

jns printf

step2:

not ax

add ax, 1

xor cx, cx

mov cx, ax

mov ah, 09h

lea dx, znak

int 21h

mov ax, cx

printf:

mov bx, 10

xor si, si

xor dx, dx

xor cx, cx

call DecToASCII

mov ah, 09h

lea dx, i+2

int 21h

mov ah, 4ch

int 21h

end main

**Тест программы:**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание**