**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО**

**ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»  
(БГТУ им. В.Г.Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №4

Дисциплина: Архитектура вычислительных систем

по теме Команды передачи управления

Выполнил: ст. группы ВТ-32  
 Воскобойников И. С.

Проверил: Осипов.О.В

**Белгород 2020**

**Цель работы**: изучение команд перехода для организации циклов и ветвлений, получение навыков создания процедур с аргументами

**Задания для выполнения к работе**

1. Написать программу для вычисления значения арифметического выражения, используя команды условного и безусловного перехода согласно варианту задания. Подобрать набор тестовых данных (не менее 3). При выполнении операций с числами, преобразовывать их к 4-байтовым числам со знаком. Результат вывести на экран.
2. Написать программу для вычисления значения арифметического выражения, содержащего функцию. Вычисление функции организовать в виде отдельной подпрограммы по всем правилам, описанным выше. Для обработки массивов использовать команды для работы с циклами и команды условного перехода. Подобрать набор тестовых данных (не менее 3). Результат вывести на экран.

**ЗАДАНИЕ 1**

|  |  |
| --- | --- |
|  | *x* – беззнаковое однобайтовое  *y*, *z, t* – знаковые  2-байтовые числа. |

.386 ; Тип процессора

.model flat, stdcall ; Модель памяти и стиль вызова подпрограмм

option casemap: none ; Чувствительность к регистру

; --- Подключение файлов с кодом, макросами, константами, прототипами функций и т.д.

include c:\masm32\include\windows.inc

include c:\masm32\include\kernel32.inc

include c:\masm32\include\user32.inc

include c:\masm32\include\msvcrt.inc

; --- Подключаемые библиотеки ---

includelib c:\masm32\lib\user32.lib

includelib c:\masm32\lib\kernel32.lib

includelib c:\masm32\lib\msvcrt.lib

; --- Сегмент данных ---

.data

x db ?

y dw ?

z dw ?

t dw ?

i\_f db "%d %d %d %d", 0

de db "x[0,15] ", 0

dej1 db "x<0 ", 0

dej2 db "x>15 ", 0

o\_f db "EAX = %d", 13, 10,

"EDX = %d", 13, 10,

"%s", 0

; --- Сегмент кода ---

.code

start:

push offset t ;осуществляем ввод

push offset z ;

push offset y ;

push offset x ;

push offset i\_f ;

call crt\_scanf ;

add esp, 5\*4 ;чистим стек после ввода

MOV AL, x ;помещаем х в eax для сравнения

MOVSX AX, AL ;и

MOVSX EAX, AX ;расширяем его до двойного слова

CMP x, 0

JL j1

CMP x, 15

JG j2

;x[0,15]

push offset de ;вывод того что мы попали в x[0,15]

MOVSX EBX, z ;записываем z в EBX попутно расширяя его

ADD EBX, EAX ;z+x

MOV ECX, EAX ;запоминаем расширенный х что б еще раз не расширать

CDQ ;расширяем х что бы поделить его он уже в EAX

IDIV EBX ;x/(z+x)

IMUL EAX, 12 ;12\*x/(z+x)

MOV EDI, EAX ;сохраняем целую часть

MOV ESI, EDX ;сохраняем остаток

MOV EAX, ECX ;x

IMUL EAX, EAX ;x^2

IMUL EAX, EAX ;x^4

MOVSX EBX, z ;записываем z в EBX попутно расширяя его

SUB EAX, EBX ;x^4-z

MOVSX EBX, y ;записываем y в EBX попутно расширяя его

SUB EBX, ECX ;y-x

CDQ ;расширяем х^4-z что бы поделить его он уже в EAX

IDIV EBX ;(x^4-z)/(y-x)

ADD EAX, EDI ;(x^4-z)/(y-x) + 12\*x/(z+x)

ADD EDX, ESI ;и складываем остатки

JMP out1 ;выход к выводу

j1: ;x<0

push offset dej1;вывод того что мы попали в x<0

MOVSX EBX, z ;записываем z в EBX попутно расширяя его

IMUL EBX, EBX ;возводим z в квадрат

MOVSX ECX, t ;записываем t в ECX попутно расширяя его

ADD EBX, ECX ;t+z^2

CDQ ;расширяем х что бы поделить его он уже в EAX

IDIV EBX ;x/(t+z^2)

JMP out1 ;выход к выводу

j2: ;x>15

push offset dej2;вывод того что мы попали в x>15

IMUL EAX, EAX ;x^2

ADD EAX, 23 ;x^2+23

MOVSX EBX, t ;записываем t в EBX попутно расширяя его

MOVSX ECX, y ;записываем y в ECX попутно расширяя его

CDQ ;расширяем х что бы поделить его он уже в EAX

IDIV EBX ;(x^2+23)/t

SUB EAX, ECX ;(x^2+23)/t - y

out1:

push EDX ;вывод

push EAX ;вывод

push offset o\_f ;вывод

call crt\_printf ;вывод

add esp, 4\*4 ;чистим стек после вывода

push NULL

call ExitProcess ; Выход из программы

end start

**ЗАДАНИЕ 2**

|  |  |
| --- | --- |
|  | *h* – однобайтовое знаковое  *x* – массив двухбайтовых знаковых чисел  *y* – массив однобайтовых беззнаковых чисел  *n* – беззнаковая переменная размером 1 байт |

.386 ; Тип процессора

.model flat, stdcall ; Модель памяти и стиль вызова подпрограмм

option casemap: none ; Чувствительность к регистру

; --- Подключение файлов с кодом, макросами, константами, прототипами функций и т.д.

include c:\masm32\include\windows.inc

include c:\masm32\include\kernel32.inc

include c:\masm32\include\user32.inc

include c:\masm32\include\msvcrt.inc

; --- Подключаемые библиотеки ---

includelib c:\masm32\lib\user32.lib

includelib c:\masm32\lib\kernel32.lib

includelib c:\masm32\lib\msvcrt.lib

; --- Сегмент данных ---

.data

n db 10

h db 9

k db 8

x db ?, ?, ?, ?, ?, ?, ?, ?, ?, ?

y db ?, ?, ?, ?, ?, ?, ?, ?, ?, ?

i\_f db "%hhd %hhd", 0

hk\_invite db "Input h k: ", 0

i\_invite db "Input array: ", 0

o\_print db "Your array: ", 0

i\_d db "%hhd",0

o\_d db "%hhd ",0

br db 13, 10, 0

w db "w = %d", 0

; --- Сегмент кода ---

.code

;процедура ввода массива void input\_array(short\* a, int n)

input\_arr proc

push EAX ;сохраняем значение EAX

push ECX ;сохраняем значение ECX

push EDX ;сохраняем значение EDX

push offset i\_invite

call crt\_printf

add esp, 4

MOV EAX, [ESP + 16] ;Достаем из памяти указатель на начало массива

MOV ECX, [ESP + 20] ;Достаем из памяти размер массива

i1:

push ECX ;сохраняем состояние регистра потому что

push EAX ;crt\_printf их затирает

push offset i\_d

call crt\_scanf ;ввод очередного элемента

add esp, 4 ;очистка стека после ввода

pop EAX ;восстанавливаем значение регистров

pop ECX ;

INC EAX ;переход к следующему элементу

LOOP i1 ;продолжение цикла

pop EDX ;EDX

pop ECX ;Достаем сохраненные регистры ECX

pop EAX ;EAX

ret 4\*2 ;выход

input\_arr endp

;процедура для вывода void output\_array(short\* a, int n)

output\_arr proc

push EAX ;сохраняем значение EAX

push ECX ;сохраняем значение ECX

push EDX ;сохраняем значение EDX

push offset o\_print

call crt\_printf

add esp, 4

MOV EAX, [ESP + 16] ;Достаем из памяти указатель на начало массива

MOV ECX, [ESP + 20] ;Достаем из памяти размер массива

o1:

push ECX ;сохраняем состояние регистра потому что

push EAX ;crt\_printf их затирает

MOV AL, [EAX] ;передаем то что нужно вывести в AL

MOVSX AX, AL ;расширяем AL до AX что бы использовать push

push AX ;push не любит 1(

push offset o\_d

call crt\_printf ;вывод очередного элемента

add esp, 6 ;очистка стека после ввода

pop EAX ;восстанавливаем значение регистров

pop ECX ;

INC EAX ;переход к следующему элементу

LOOP o1 ;продолжение цикла

push offset br

call crt\_printf

add esp, 4

pop EDX ;EDX

pop ECX ;Достаем сохраненные регистры ECX

pop EAX ;EAX

ret 4\*2 ;выход

output\_arr endp

;функция из лабораторной int f(int x, int y)

f proc

push ECX ;сохраняем значение ECX

MOV EAX, [ESP + 8] ;Достаем из памяти x

MOV ECX, [ESP + 12] ;Достаем из памяти y

CMP EAX, ECX ;x<=y

JLE jd ;

CMP ECX, 10 ;y<=10

JLE jd

;|x-y^2|

IMUL ECX, ECX ;y^2

SUB EAX, ECX ;|x-y^2|

CMP EAX, 0 ;если результат меньше нуля делаем модуль NEG

JL my\_modul ;прыгаем на модуль

JMP out\_f ;если модуль делать не надо сразу на выход

jd: ;метка для x>y и y>10 ((x+y)^2)

ADD EAX, ECX ;x+y

IMUL EAX, EAX ;(x+y)^2

JMP out\_f

my\_modul: ;метка для модуля

NEG EAX ;инвертируем знак

out\_f: ;метка для выхода

pop ECX ;Достаем сохраненные регистры ECX

ret 4\*2 ;выход

f endp

;функция возведения в целую степень int p(int x, int p)

p proc

push ECX ;сохраняем значение ECX

push EDX ;сохраняем значение EDX

MOV EDX, [ESP + 12] ;Достаем из памяти число возводимое в степень

MOV ECX, [ESP + 16] ;Достаем из памяти степень

MOV EAX, EDX ;переносим число в EAX

p1:

IMUL EAX, EDX ;накручиваем степень

LOOP p1

pop EDX ;EDX

pop ECX ;Достаем сохраненные регистры ECX

ret 4\*3 ;выход

p endp

start:

push offset hk\_invite; Вывод приглашения ввода

call crt\_printf

add esp, 4 ;чистим стек

push offset k ;осуществляем ввод

push offset h ;

push offset i\_f ;

call crt\_scanf ;

add esp, 3\*4 ;чистим стек после ввода

MOVSX CX, n ;перемещаем n в CX и расширяем его

MOVSX ECX, CX ;перемещаем n в ECX и расширяем его

push ECX ;передаем параметры для input\_arr в стек

push offset x ;

call input\_arr ;вызов ввода

push ECX ;ввод массива y

push offset y ;

call input\_arr ;

push ECX ;передаем параметры для output\_arr в стек

push offset x ;

call output\_arr ;вызов вывода

push ECX ;вывод массива y

push offset y ;

call output\_arr ;

MOV ESI, 0 ;индекс прохода по массиву

XOR EBP, EBP ;сюда будет накапливаться результат

main\_loop:

MOVSX BX, k ;переносим k в регистры и расширяем

MOVSX EBX, BX ;еще расширяем

MOVSX DX, x[ESI];переносим xi в регистры и расширяем

MOVSX EDX, DX ;еще расширяем

MOVSX DI, y[ESI];переносим yi в регистры и расширяем

MOVSX EDI, DI ;еще расширяем

push EBX

push EDX

call p ;EAX=xi^k

ADD EBP, EAX ;x^k

MOV EAX, EDX ;перемещаем x в EAX

ADD EAX, EDI ;xi+yi

IMUL EAX, EBX ;k\*(xi+yi)

MOVSX BX, h ;переносим h в регистры и расширяем

MOVSX EBX, BX ;еще расширяем

CDQ ;расширяем EAX до EDX:EAX для деления

IDIV EBX ;k\*(xi+yi)/h

SUB EBP, EAX ;xi^k -k\*(xi+yi)/h

MOV EAX, EDX ;EAX=xi

IMUL EAX, EBX ;xi\*h

CDQ ;расширяем EAX до EDX:EAX для деления

IDIV EDI ;xi\*h/yi

ADD EBP, EAX ;xi^k + xi\*h/yi - k\*(xi+yi)/h

push EDI

push EDX

call f

IMUL EAX, EAX ;f(x,y)^2

ADD EBP, EAX ;xi^k + xi\*h/yi - k\*(xi+yi)/h + f(x,y)^2

INC ESI ;переход к следующему индексу

LOOP main\_loop

push EBP ;вывод результата

push offset w

call crt\_printf

add esp, 8 ;чистим стек

push NULL

call ExitProcess ; Выход из программы