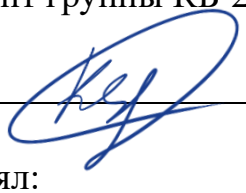
Федеральное государственное бюджетное образовательное учреждение высшего образования "Белгородский государственный технологический университет им. В.Г. Шухова"

Кафедра программного обеспечения вычислительной техники и автоматизированных систем.

Лабораторная работа № 4
Команды передачи управления.
Вариант 13

Выполнил:
Студент группы КБ-211



Коренев Д.Н.

Принял:

Осипов О.В.

Белгород 2023

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

Задание

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.

Вариант	Выражение	Размерность и тип переменных
13	$e = \begin{cases} \dfrac{(x+y+1)^2}{y^3-2}z^4, & y > 10 \text{ и } \lvert z\rvert > 5, \\ -z^2 + \dfrac{1}{x} + \dfrac{x}{y+10}, & z \in [-5;5] \text{ и } y > 10, \\ -5x^2 - 2(z+1)y, & y \le 10 \end{cases}$	x – беззнаковое 2-байтовое y – знаковое однобайтовое z – беззнаковое однобайтовое

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.

Вариант	Выражение	Размерность и тип переменных
13	$r = \sum_{i=0}^{n} k(y_i + x_i) + k^3 - y_i^2 + \dfrac{f(x_i, y_i)}{k},$ $f(x,y) = \sum_{j=0}^{x} \lvert yj - 1\rvert$	x – массив 1-байтовых беззнаковых чисел y – массив 2-байтовых знаковых чисел k – беззнаковая переменная размером 2 байта n – беззнаковая переменная размером 1 байт

```
.386
.model flat, stdcall
option casemap: none

include include\windows.inc
include include\kernel32.inc
include include\user32.inc
include include\msvcrt.inc

includelib user32.lib
includelib kernel32.lib
includelib msvcrt.lib
```

```
mem_dump32 MACRO mem_addr, rows
    LOCAL loop1, loop2
    LOCAL mem_dump_fmt, header_fmt, footer_fmt
    LOCAL r_edi, r_ebx, temp_mem_addr, stack_decr
    LOCAL holder_esi, holder_ebp, holder_ecx, holder_eax
    LOCAL reg_dump_fmt1, reg_dump_fmt2, reg_header
    .data
        reg_dump_fmt1 DB "| EAX=0x%08x  EBX=0x%08x  ECX=0x%08x  EDX=0x%08x |",
            13, 10, 0
        reg_dump_fmt2 DB "| ESI=0x%08x  EDI=0x%08x  EBP=0x%08x  ESP=0x%08x |",
            13, 10, 0
        reg_header DB "+==-REGISTERS-", 51 DUP ("="), "+", 13, 10, 0
        mem_dump_fmt DB "| %p", 9, " ", 16 DUP ("%02x "), "|", 13, 10, 0
        header_fmt DB "+==-ADDR-", 7 DUP ("="),
            "-00-01-02-03-04-05-06-07-08-09-0A-0B-0C-0D-0E-0F-+", 13, 10, 0
        footer_fmt DB "+", 41 DUP ("="), 45, 64, 75, 83, 69, 69, 78, 45, 61, 45,
            64, 75, 69, 82, 65, 83, 73, 46, 82, 85, 45, 61, 61, 43, 13, 10, 0
        temp_mem_addr DD ?
        r_edi DD ?
        r_ebx DD ?
        holder_esi DD ?
        holder_ebp DD ?
        holder_ecx DD ?
        holder_eax DD ?
        stack_decr DD 0
    .code
        ; Сохраняем регистры
        mov holder_esi, ESI
        mov holder_ebp, EBP
        mov holder_ecx, ECX
        mov holder_eax, EAX

        ; Выводим регистры
        invoke crt_printf, offset reg_header
        invoke crt_printf, offset reg_dump_fmt1, EAX, EBX, ECX, EDX
        invoke crt_printf, offset reg_dump_fmt2, ESI, EDI, EBP, ESP

        invoke crt_printf, offset header_fmt
        mov ESI, mem_addr
        mov temp_mem_addr, mem_addr
        mov r_edi, rows
        mov r_ebx, mem_addr ; EBX - addres for left column
        mov EBP, 15 ; offset
    loop2:
        mov ECX, 16
    loop1:
        mov EAX, 0 ; EAX - байт данных
        mov AL, [ESI+EBP]
        push EAX
        inc stack_decr
```

```
        dec EBP
        dec ECX
        jnz loop1
        add EBP, 32
        push r_ebx
        push offset mem_dump_fmt
        inc stack_decr
        inc stack_decr
        call crt_printf
        inc stack_decr
        add r_ebx, 16
        dec r_edi
        jnz loop2
        invoke crt_printf, offset footer_fmt

        ; Чистим стек
        mov EAX, stack_decr
        imul EAX, 4
        add ESP, EAX

        ; Восстанавливаем регистры
        mov ESI, holder_esi
        mov EBP, holder_ebp
        mov ECX, holder_ecx
        mov EAX, holder_eax
    ENDM

print_var MACRO var, var_size
    LOCAL loop1
    LOCAL stack_decr
    LOCAL holder_esi, holder_ebp, holder_ecx, holder_eax, holder_edi, holder_ebx
    LOCAL var_fmt
    .data
        var_fmt DB "0x", var_size - 1 DUP ("%02X_"), "%02X", 0
        holder_esi DD ?
        holder_ebp DD ?
        holder_ecx DD ?
        holder_eax DD ?
        holder_edi DD ?
        holder_ebx DD ?
        stack_decr DD 0
    .code
        mov holder_esi, ESI
        mov holder_ebp, EBP
        mov holder_ecx, ECX
        mov holder_eax, EAX
        mov holder_edi, EDI
        mov holder_ebx, EBX

        mov ESI, offset var
```

```
        mov EDI, 3
        mov EBX, offset var ; EBX - addres for left column
        mov EBP, 0 ; offset

        mov ECX, 0
    loop1:
        mov EAX, 0 ; EAX - data of byte
        mov AL, [ESI+EBP]
        push EAX
        inc stack_decr
        inc EBP
        inc ECX
        cmp ECX, var_size
        jne loop1
        push offset var_fmt
        inc stack_decr
        call crt_printf
        inc stack_decr
        add EBX, 16
        dec EDI

        ; Чистим стек
        mov EAX, stack_decr
        imul EAX, 4
        add ESP, EAX

        ; Восстанавливаем регистры
        mov ESI, holder_esi
        mov EBP, holder_ebp
        mov ECX, holder_ecx
        mov EAX, holder_eax
        mov EDI, holder_edi
        mov EBX, holder_ebx
    ENDM

abs_var MACRO var
    LOCAL abs_end
    .code
        cmp var, 0
        jge abs_end
        neg var
    abs_end:
    ENDM

nline MACRO
    LOCAL nline_fmt
    .data
        nline_fmt DB 13, 10, 0
    .code
        invoke crt_printf, offset nline_fmt
```

```
    ENDM

.data
    main_fmt DB "DBG: 0x%016X", 13, 10, 0
.code
solve_eq13 PROC x:WORD, y:BYTE, z:BYTE
    LOCAL abs_z:DWORD
        push EBX
        push ESI
        push EDI
        push EBP
        push ECX

        ; Проверяем (y > 10 and |z| > 5)
        cmp y, 10
        jng next_check1
        movsx EAX, z
        mov abs_z, EAX
        abs_var abs_z
        cmp abs_z, 5
        jg y_g_10_z_g_5
    next_check1:
        ; Проверяем (z >= -5 and z <= 5 and y > 10)
        cmp z, -5
        jnge next_check2
        cmp z, 5
        jnle next_check2
        cmp y, 10
        jg z_ge_m5_z_le_5_y_g_10
    next_check2:
        ; Проверяем (y <= 10)
        cmp y, 10
        jle y_le_10
    y_g_10_z_g_5:
        movzx EAX, x
        movsx EBX, y
        add EAX, EBX    ; EAX = x+y
        add EAX, 1      ; EAX = x+y+1
        imul EAX, EAX   ; EAX = (x+y+1)^2
        mov ESI, EAX    ; ESI = (x+y+1)^2 ; (1+14+1)^2=256

        movsx EBX, y
        movsx ECX, y
        imul EBX, EBX   ; EBX = y^2
        imul EBX, ECX   ; EBX = y^3
        sub EBX, 2      ; EBX = y^3 - 2 ; 14^3-2=2742

        cdq
        idiv EBX        ; EAX = (x+y+1)^2 / (y^3 - 2) ; 256//2742=0
```

```
        movsx EBX, z
        imul EBX, EBX   ; EBX = z^2
        imul EBX, EBX   ; EBX = z^4 ; 6^4=1296

        imul EAX, EBX   ; EAX = (x+y+1)^2 / (y^3 - 2) * z^4 ; 0*1296=0

        ; x dw 1
        ; y db 14
        ; z db 6
        ; 0x00_00_00_00
        jmp end_res

    z_ge_m5_z_le_5_y_g_10:
        movsx EAX, z
        imul EAX, EAX
        neg EAX
        mov EBX, EAX    ; EBX = -z^2

        movzx ECX, x
        mov EAX, 1
        cdq
        div ECX
        mov ECX, EAX    ; ECX = 1/x

        movzx EAX, x
        movsx EDX, y
        add ESI, 10     ; EDX = y+10
        cdq
        idiv ESI        ; EAX = x/(y+10)

        add EBX, ECX    ; EBX = -z^2 + 1/x
        add EBX, EAX    ; EBX = -z^2 + 1/x + x/(y+10)

        ; mov EBX, 69h
        mov EAX, EBX

        ; x dw 1
        ; y db 14
        ; z db 3
        ; 0xff_ff_ff_f8
        jmp end_res

    y_le_10:
        movzx EAX, x
        imul EAX, EAX
        imul EAX, 5
        neg EAX         ; -5x^2

        movsx EBX, z
        inc EBX         ; z+1
```

```
        imul EBX, 2     ; 2(z+1)

        movsx ECX, y
        imul EBX, ECX   ; 2(z+1)y
        sub EAX, EBX    ; -5x^2 - 2(z+1)y

        ; x dw 1
        ; y db 5
        ; z db -6
        ; 0x00_00_00_2d
        jmp end_res
    end_res:

        pop ECX
        pop EBP
        pop EDI
        pop ESI
        pop EBX
        ret             ; result in EAX
    solve_eq13 ENDP

abs PROC var:DWORD
    mov EAX, var
    cmp EAX, 0
    jge abs_end
    neg EAX
abs_end:
    ret
    abs ENDP

function1 PROC x:BYTE, y:WORD
    ; int sum = 0;
    ; for (j = 0; j < x; j++)
    ;   sum += abs((y*j)-1);
    LOCAL sum:DWORD, j:DWORD
    push ECX
    push EBX
    push ESI

    mov sum, 0
    mov j, 0


    movsx ESI, x
    sub ESI, 1
loop1:
    mov EAX, j
    xor EBX, EBX
    mov BX, y           ; y
```

```
    movsx EAX, BX       ; y
    imul EAX, j         ; y*j
    sub EAX, 1          ; y*j-1
    mov EBX, EAX        ; EBX = y*j-1
    invoke abs, EBX     ; EAX = abs(y*j-1)
    add sum, EAX

    xor EBX, EBX
    mov EBX, j
    inc j
    cmp EBX, ESI
    jl loop1

    mov EAX, sum

    pop ESI
    pop EBX
    pop ECX
    ret                 ; result in EAX
    function1 ENDP

solve_sig13 PROC x:DWORD, y:DWORD, k:WORD, n:BYTE
    ; int sum = 0;
    ; for (i = 0; i < n; i++){
    ;   sum += k*(y[i]+x[i])+k^3-y[i]^2+((f(x[i],y[i]))/(k));
    ; }
    LOCAL sum:DWORD, i:DWORD

    mov sum, 0
    mov i, 0
    xor ESI, ESI
    movsx ESI, byte ptr [n]
loop1:
    mov EAX, i
    imul EAX, 2
    add EAX, y
    movsx EBX, byte ptr [EAX] ; y[i]

    mov EAX, i
    imul EAX, 2
    add EAX, x
    movsx ECX, byte ptr [EAX] ; x[i]

    add EBX, ECX            ; y[i]+x[i]
    movsx EAX, k
    imul EBX, EAX             ; k*(y[i]+x[i])

    mov ECX, EAX
    imul ECX, EAX             ; k^2
    imul ECX, EAX             ; k^3
```

```
    add EBX, ECX            ; k*(y[i]+x[i])+k^3

    mov ECX, i
    imul ECX, 2
    add ECX, y
    movsx EDX, byte ptr [ECX] ; y[i]
    imul EDX, EDX             ; y[i]^2

    sub EBX, EDX            ; k*(y[i]+x[i])+k^3-y[i]^2
    ; -4824 here
    push EBX

    mov ECX, i
    imul ECX, 2
    add ECX, x
    movsx EDX, byte ptr [ECX] ; x[i]

    mov ECX, i
    imul ECX, 2
    add ECX, y
    movsx EAX, byte ptr [ECX] ; y[i]

    mov EBX, EDX
    invoke function1, BL, AX ; f(x[i],y[i])
    ; 72 here

    movzx ECX, k
    cdq
    idiv ECX                ; EAX = f(x[i],y[i])/k
    ; 18 here

    pop EBX
    add EBX, EAX            ; EBX = k*(y[i]+x[i])+k^3-y[i]^2+f(x[i],y[i])/k

    add sum, EBX

    inc i
    dec ESI
    jnz loop1

    mov EAX, sum

    ret                 ; result in EAX
    solve_sig13 ENDP

.data
    x dw 1,?            ; беззнаковое 2 байта
    y db 14,?,?,?       ; знаковое однобайтовое
    z db 6,?,?,?        ; знаковое однобайтовое
```

```
    result dd 0

    x2 db 2,3,4,5       ; 1б беззнаковые числа
    y2 dw 72,54,3,9     ; 2б знаковые числа
    k2 dw 4,?           ; 2б беззноковая переменная
    n2 db 4,?,?,?       ; 1б беззноковая переменная
                        ; количество элементов в массиве
    result2 dd 0

    ; 1
    ; 53
    ; 5
    ; 26
    ; =
    ; 85
.code

bubble_sort proc parr:DWORD,cnt:DWORD

    push ebx
    push esi

    sub cnt, 1                  ; set count to 1 less than member count

  lbl0:
    mov esi, parr               ; load array address into ESI
    xor edx, edx                ; zero the "changed" flag
    mov ebx, cnt                ; set the loop counter to member count

  lbl1:
    mov eax, [esi]              ; load first pair of array numbers into registers
    mov ecx, [esi+4]
    cmp eax, ecx                ; compare which is higher
    jle lbl2

    mov [esi], ecx              ; swap if 1st number is higher
    mov [esi+4], eax
    mov edx, 1                  ; set the changed flag if any swap is performed

  lbl2:
    add esi, 4                  ; step up one to compare next adjoining pair
    sub ebx, 1                  ; decrement the counter
    jnz lbl1

    test edx, edx               ; test if the changed flag is set.
    jz lbl3                     ; break out of the loop if no swap occurred

    sub cnt, 1                  ; decrement the upper bound for next pass
    jmp lbl0                    ; loop back for next pass
```

```
  lbl3:
    pop esi
    pop ebx

    ret

bubble_sort endp

print_array proc array_ptr:dword, array_size:dword, item_size:dword
    .data
        print_array_fmt DB "%d ",0
    .code
    push ebx
    push eax
    push ecx
    push edx

    mov ecx, array_ptr
    mov ebx, array_size


    prnt:

        mov eax, item_size
        cmp eax, 1
        je switch1
        cmp eax, 2
        je switch2
        cmp eax, 4
        je switch4

    switch1:
        mov eax, 0
        mov al, byte ptr [ecx]                  ; ecx = array[i]
        jmp switch_end
    switch2:
        mov eax, 0
        mov ax, word ptr [ecx]                  ; ecx = array[i]
        jmp switch_end
    switch4:
        mov eax, 0
        mov eax, dword ptr [ecx]                  ; ecx = array[i]
        jmp switch_end
    switch_end:

        push eax
        push ecx

        push eax
        push offset print_array_fmt
```

```
        call crt_printf                 ; Вывод результата на экран
        add esp, 2*4

        pop ecx
        pop eax

        add ecx, item_size                      ; ecx = array[i+1]
        dec ebx                         ; ebx = n-1
        cmp ebx, 0
        jne prnt                        ; если ebx не равно 0, то переходим к
следующему элементу массива

        pop edx
        pop ecx
        pop eax
        pop ebx
        RET 8
print_array endp

print_array_8byte PROC ptrr:DWORD, sze:DWORD
    .data
        print_array_8byte_fmt DB "%lld ",0
        val DQ ?
    .code
    pusha

    mov eax, ptrr
    mov esi, sze
loopm:

    mov ebx, eax
    push eax

    mov eax, [ebx]
    mov dword ptr val[0], eax
    add ebx, 4
    mov eax, [ebx]
    mov dword ptr val[4], eax
    ; invoke crt_printf, offset main_fmt, val[0]
    ; invoke crt_printf, offset main_fmt, val[4]
    invoke crt_printf, offset print_array_8byte_fmt, val
    pop eax
    add eax, 8
    dec esi
    jnz loopm

    popa
    ret 8
print_array_8byte ENDP
scan_array_8byte PROC ptrr:DWORD, sze:DWORD
```

```
    .data
        scan_array_8byte_fmt DB "%lld", 0
        scan_array_8byte_val DQ ?
    .code
    pusha

    mov eax, ptrr
    mov esi, sze
loopm:

    mov ebx, eax
    push eax

    invoke crt_scanf, offset scan_array_8byte_fmt, offset scan_array_8byte_val
    ; invoke crt_printf, offset main_fmt, scan_array_8byte_val
    ; var 4:4
    mov eax, dword ptr scan_array_8byte_val[0]
    mov [ebx], eax
    mov eax, dword ptr scan_array_8byte_val[4]
    mov [ebx+4], eax

    pop eax
    add eax, 8
    dec esi
    jnz loopm

    popa
    ret 8

scan_array_8byte ENDP

start:
    invoke solve_eq13, x, y, z
    mov result, EAX
    .data
        eq13_fmt1 DB "x=%hu, y=%hhd, z=%hhd, result=%d=", 0
    .code
    invoke crt_printf, offset eq13_fmt1, dword ptr x, dword ptr y, dword ptr z,
        dword ptr result
    print_var result, 4
    nline
    nline

    .data
        eq13_fmt2_1 DB "x={", 0
        eq13_fmt2_2 DB "},", 13, 10, "y={", 0
        eq13_fmt2_3 DB "},", 13, 10, "k=%hd, n=%d, result=%d=", 0
    .code
    invoke solve_sig13, offset x2, offset y2, k2, n2
    mov result2, EAX
```

```
    invoke crt_printf, offset eq13_fmt2_1
    invoke print_array, offset x2, 4, 1
    invoke crt_printf, offset eq13_fmt2_2
    invoke print_array, offset y2, 4, 1
    mov eax, 0
    mov al, n2
    mov ebx, 0
    mov bx, k2
    invoke crt_printf, offset eq13_fmt2_3, ebx, eax, result2
    print_var result2, 4 ; -1651
    nline
    nline

.data
    arr dq ?,?,?,4,5,6,7,8,9,10
    arr_size dd 10
.code
    invoke scan_array_8byte, offset arr, 3
    invoke print_array_8byte, offset arr, arr_size
    ; nline
    ; mem_dump32 offset x2, 4
    invoke ExitProcess, 0   ; Выход из программы
end start
```

Вывод: в ходе лабораторной работы мы изучили команды перехода для организации циклов и ветвлений, получили навыки создания процедур с аргументами.