**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.

*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 |  | 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 |  | *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.