

Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра ЕОМ



Звіт

до лабораторної роботи №1

з дисципліни: “Системне програмування”

Варіант №18

Виконав:

ст.гр. КІ-34

Романів В. А.

Прийняв:

Мархивка В. С.

Львів 2022

*Тема:* змішане програмування на мовах С та асемблер.

*Мета:* оволодіти навиками створення програм, частини яких написані різними мовами програмування. Засвоїти правила взаємодії між програмними модулями різних мов програмування.

### Завдання

Створити перший програмний проект для 32-х/64-х розрядної платформ, який реалізує схему викликів C-ASM-C та здійснює обчислення заданого виразу, згідно варіанту. Проект повинний складатися з двох модулів, передача параметрів між якими здійснюється через стек. Константа передається через спільну пам'ять, як глобальні дані. Множення/ділення рекомендовано робити через команди зсувів.

18.	$X = 4 * (B_2 - C_1) + D_2 / 4 + K$	56987018
-----	-------------------------------------	----------

Код програми x32:

Main.cpp

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <string.h>
#include <cstdlib>
#include <iostream>

extern "C" int calc(signed short, signed char, signed short);

extern "C" int K = 0x56987018;

// Y = 4 * (B2 - C1) + D2/4 + K

int main()
{
    char arr[256];

    signed long        checker;

    signed short        b;
    signed char          c;
    signed short        d;

    //Enter B value
    printf("B: ");
    scanf("%s", arr);

    //Check if input is correct
    if ((strlen(arr) > 6 && arr[0] == '-') ||
        (strlen(arr) > 5 && arr[0] != '-')) {
        printf("Two bytes overflow");
        return -1;
    }

    //Check if B has the correct size
    checker = atoi(arr);
```

```

    if (checker > 32767 || checker < -32767) {
        printf("Two bytes overflow");
        return -1;
    }

    //initialise b
    b = (short)checker;

    //Enter C value
    printf("C: ");
    scanf("%s", arr);

    //Check if input is correct
    if ((strlen(arr) > 4 && arr[0] == '-') ||
        (strlen(arr) > 3 && arr[0] != '-')) {
        printf("One byte overflow");
        return -1;
    }

    //Check if C has the correct size
    checker = atoi(arr);
    if (checker > 127 || checker < -127) {
        printf("One byte overflow");
        return -1;
    }
    c = (char)checker;

    //Enter D value
    printf("D: ");
    scanf("%s", arr);

    //Check if input is correct
    if ((strlen(arr) > 6 && arr[0] == '-') ||
        (strlen(arr) > 5 && arr[0] != '-')) {
        printf("Two bytes overflow");
        return -1;
    }

    //Check if D has the correct size
    checker = atoi(arr);
    if (checker > 32767 || checker < -32767) {
        printf("Two bytes overflow");
        return -1;
    }
    d = (short)checker;

    printf("\n4 * (B - C) + D/4 + K =      %d\n", (signed int)(4 * (b - c) + (d / 4) +
K));
    signed int res = calc(b, c, d);
    printf("\nResult of procedure calc is: %d\n", res);

    std::cout << "\n\n\n Finish" << std::endl;
    return 0;
}

```

## Calc.asm під x-32

```

.686
.model flat, c

printf proto c : vararg
EXTERN K : DWORD

.data
msg db 'Output from asm module is:  %d', 0

```

```

.code
calc PROC
;Save the previous base pointer (ebp)
;and set EBP to point at that position on
;the stack (right below the return address).
push    ebp
mov     ebp, esp

;Enter the first operand from the function parameters
mov     ax, word ptr [ebp + 8]

;Extend ax to eax
cwde

;Save data from eax to ebx
mov     ebx, eax

;Enter the second operand from the function parameters
mov     al, byte ptr [ebp + 12]

;firstly extend al to ax then ax to eax
cbw
cwde

;Perform the subtraction operation
;(B - C)
sub     ebx, eax
mov     eax, ebx

;Perform the mult
;4*(B-C)
shl     eax, 2

;Save result to stack
push    eax

;Reset the eax register
xor     eax, eax

;Enter the third operand from the function parameters
mov     ax, word ptr [ebp + 16]

;Save result of data from eax plus 3 in edx register
lea     edx, [eax + 3]

;check if the third operand is negative
test    ax, ax
js      next_position
;if not just perform D/4 then go to another operations
sar     ax, 2
jmp     end_point

;if this is a negative operand
;move to eax data from edx register 52line on code
;and then D/4
next_position:
mov     eax, edx
sar     ax, 2
cwde

end_point:
;Save result in edx register (D/4 result)
mov     edx, eax

;Take data from stack to eax register
pop     eax

```

```

;Perform 4 * (B - C)+ D/4
;and then perform 4 * (B - C)+ D/4 + K
add     eax, edx
add     eax, K

;Take previous base pointer (ebp)
pop     ebp

;Save the eax data to the stack
push    eax

;Print message to the screen
invoke  printf, offset msg, eax

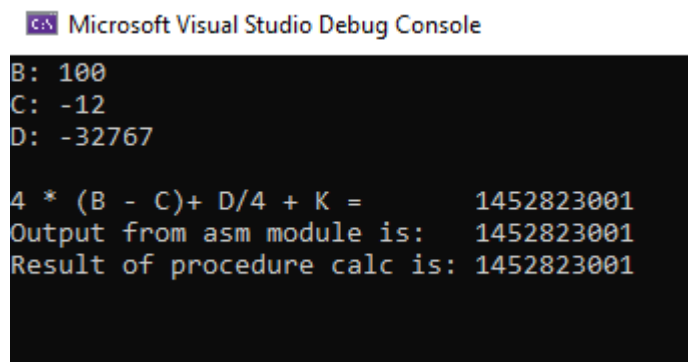
;Take the data from the stack and return it
pop     eax
ret

calc ENDP

END

```

## Результат роботи X-32:



```

Microsoft Visual Studio Debug Console

B: 100
C: -12
D: -32767

4 * (B - C)+ D/4 + K =      1452823001
Output from asm module is:  1452823001
Result of procedure calc is: 1452823001

```

## Main.cpp x64

```

#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <string.h>
#include <cstdlib>
#include <iostream>

extern "C" int calc(signed short, signed char, signed short);

extern "C" int K = 0x56987018;

int main()
{
    char arr[256];

    signed long checker;

    signed short b;
    signed char c;
    signed short d;

    printf("B: ");
    scanf("%s", arr);

    if ((strlen(arr) > 6 && arr[0] == '-') ||
        (strlen(arr) > 5 && arr[0] != '-')) {
        printf("Two bytes overflow");
        return -1;
    }
}

```

```

    }

    checker = atoi(arr);
    if (checker > 32767 || checker < -32767) {
        printf("Two bytes overflow");
        return -1;
    }
    b = (short)checker;

    printf("C: ");
    scanf("%s", arr);

    if ((strlen(arr) > 4 && arr[0] == '-') ||
        (strlen(arr) > 3 && arr[0] != '-')) {
        printf("One byte overflow");
        return -1;
    }

    checker = atoi(arr);
    if (checker > 127 || checker < -127) {
        printf("One byte overflow");
        return -1;
    }
    c = (char)checker;

    printf("D: ");
    scanf("%s", arr);

    if ((strlen(arr) > 6 && arr[0] == '-') ||
        (strlen(arr) > 5 && arr[0] != '-')) {
        printf("Two bytes overflow");
        return -1;
    }

    checker = atoi(arr);
    if (checker > 32767 || checker < -32767) {
        printf("Two bytes overflow");
        return -1;
    }
    d = (short)checker;

    printf("\n4 * (B - C) + D/4 + K =      %d\n", (signed int)(4 * (b - c) + (d / 4) +
K));
    signed int res = calc(b, c, d);
    printf("\nResult of procedure calc is: %d\n", res);

    std::cout << "\n\n\n Finish" << std::endl;
    return 0;
}

```

## Calc.asm x64

```
printf proto c : vararg
```

```
EXTERN K : DWORD
```

```
.data
```

```
msg db 'Output from asm module is: %d', 0
```

```
.code
```

```
calc PROC
```

```
    ;allocate space for local variables  
    sub     rsp,    20h
```

```
    ;reset rax
```

```
    xor     rax, rax
```

```
    ;move the first operand from fuction parametrs
```

```
    ;to ax
```

```
    mov     ax,     cx
```

```
    ;extend ax to eax
```

```
    cwde
```

```
    ;save the eax data to the ebx register
```

```
    mov     ebx, eax
```

```
    ;move the second operand from fuction parametrs
```

```
    ;to al
```

```
    mov     al,     dl
```

```
    ;extend al to ax then ax to eax
```

```
    cbw
```

```
    cwde
```

```
    ;Perfort B-C
```

```
    sub     ebx, eax
```

```
    mov     eax, ebx
```

```
    ;Perfort 4*(B-C)
```

```
    shl     eax, 2
```

```
    ;Save result to the r12d register
```

```
    mov     r12d, eax
```

```
    ;reset eax
```

```
    xor     eax, eax
```

```
    ;enter the third operand from fuction parametrs
```

```
    ;to al
```

```
    mov     ax,     r8w
```

```
    ;Save result of data from eax plus 3 in edx register
```

```
    lea     edx, [eax + 3]
```

```
    ;check if the third operand is negative
```

```
    test    ax, ax
```

```
    js      next_position
```

```
    ;if not just perform D/4 then go to another operations
```

```
    sar     ax, 2
```

```
    cwde
```

```
    jmp     end_point
```

```
next_position:
```

```
    ;if this is a negative operand
```

```
    ;move to eax data from edx register 41line on code
```

```
    ;and then D/4
```

```
    mov     eax, edx
```

```
    sar     ax, 2
```

```
    cwde
```

```
end_point:
```

```
    ;Save result in edx resister (D/4 result)
```

```
    mov     edx, eax
```

```
    ;Take the first step result to the eax register(32line)
```

```
    mov     eax, r12d
```

```
    ;Perform 4 * (B - C)+ D/4
```

```
    ;and then perform 4 * (B - C)+ D/4 + K
```

```

add     eax, edx
add     eax, K

;Save result to the r12d register
mov     r12d, eax
;Print message
lea     rcx, msg
mov     edx, eax
call    printf
;Return result value
mov     eax, r12d
;Return the stack pointer to the same position
;free space of local variables
add     rsp, 20h
;return
ret

```

```

calc ENDP
END

```

### Результат x64

```

Microsoft Visual Studio Debug Console
B: 1000
C: -100
D: -32767

4 * (B - C) + D/4 + K =      1452826953
Output from asm module is: 1452826953
Result of procedure calc is: 1452826953

```

### Відповіді на контрольні запитання

1. Змішане програмування – це реалізація поставленого завдання двома, або більше мовами програмування. В даному випадку це мови програмування C/C++ та асемблер.
2. На платформі x86 всі аргументи функції передаються через стек як 32 бітні дані. Значення, що повертається, також розширюється до 32 біт і повертається в регістрі EAX або у регістрі ST(0), якщо результат дійсне число. Параметри розміщуються в стеку справа наліво. Компілятор створює код прологу та епілогу для збереження та відновлення регістрів ESI, EDI, EBX та EBP, якщо вони використовуються у функції.
3. Перші чотири параметри передаються через регістри, а параметри, які залишилися через стек в порядку справа на ліво. Цілочисельні параметри в перших чотирьох позиція передаються в порядку зліва направо у регістри RCX, RDX, R8, R9. Результат повертається з допомогою регістра RAX.
4. Повернення результату – цілочисельний результат повертається за допомогою регістру RAX. Числа з плаваючою комою повертаються за допомогою регістру XMM0.(x64)  
Значення, що повертається, також розширюється до 32 біт і повертається в регістрі EAX або у регістрі ST(0), якщо результат дійсне число.(x32)
5. Стек повинна очищати викликаюча функція. (cdecl)
6. Викликана функція
7. На платформі x64 є тільки одна угода про виклики (модифікований варіант угоди \_fastcall), яка використовується по замовчуванню. Функція, яка буде викликати інші функції, обов'язково повинна в області стеку зарезервувати



місце під перші чотири параметри функції (32 байти), яка буде викликатися. Це місце використовується у функціях для копіювання значень параметрів з регістрів, що забезпечує звертання до параметрів «по-старому» ([RBP + xx]). Очищення стеку лежить на викликаючій функції.

8. 1) Ім'я процедури (функції) повинна бути задана в директиві public:

public ім'я процедури (функції)

2) Якщо процедура (функція) використовує дані, пам'ять під які виділена в іншій програмі, то в цій програмі використовується директива extrn, визначення 1, визначення 2, ...

Загальний вид визначення для даних, що передаються:

ім'я – тип: кількість, де

ім'я - ім'я даного, пам'ять під яке виділена в іншому модулі;

тип – тип даного, використовується для визначення довжини даного в байтах, для задання використовуються ключові слова: byte, word, dword, qword, tbyte або ім'я структури для 1-, 2-, 4-, 8-, 10 – байтових даних та даних визначених користувачем. Якщо в якості зовсім іншого імені використовується ім'я, визначене в директиві equ, його тип – abs;

кількість – задає кількість елементів даного типу, використовується, якщо в якості зовнішнього передається масив, дозволяє застосовувати в процедурі (функції) операції SIZE, LENGTH.

Таким чином, імена використаних, але не визначених даних повинні визначатися директивою extrn, а імена визначених даних, які можуть використовуватися іншими модулями, визначаються директивою public. Щоб забезпечити можливість однаково задавати дані в різних модулях, використовується директива global. Загальний вигляд директиви:

global визначення 1, визначення 2, ...

Визначення задаються так само, як для директиви extrn.

Щоб визначити функцію global, транслятор “дивиться” чи виділена пам'ять для даного в цьому модулі. Якщо виділена, директива еквівалентна директиві local, в протилежному випадку – extrn. Використання global дає можливість зробити загальним визначення для декількох модулів. Загальна частина може бути поміщена в файл, який підключається до модулів за допомогою директиви include.

Загальний вигляд директиви:

`include ім'я файла.`

Для файла може бути задане його повне ім'я.

Зовнішня процедура (функція) може використовуватися іншими програмами, про які розробник заздалегідь може не знати, ця програма не повинна “псувати” зміст ресурсів загального користування.

***Висновок:*** оволодів навиками створення програм, частини яких написані різними мовами програмування. Засвоїв правила взаємодії між програмними модулями різних мов програмування.