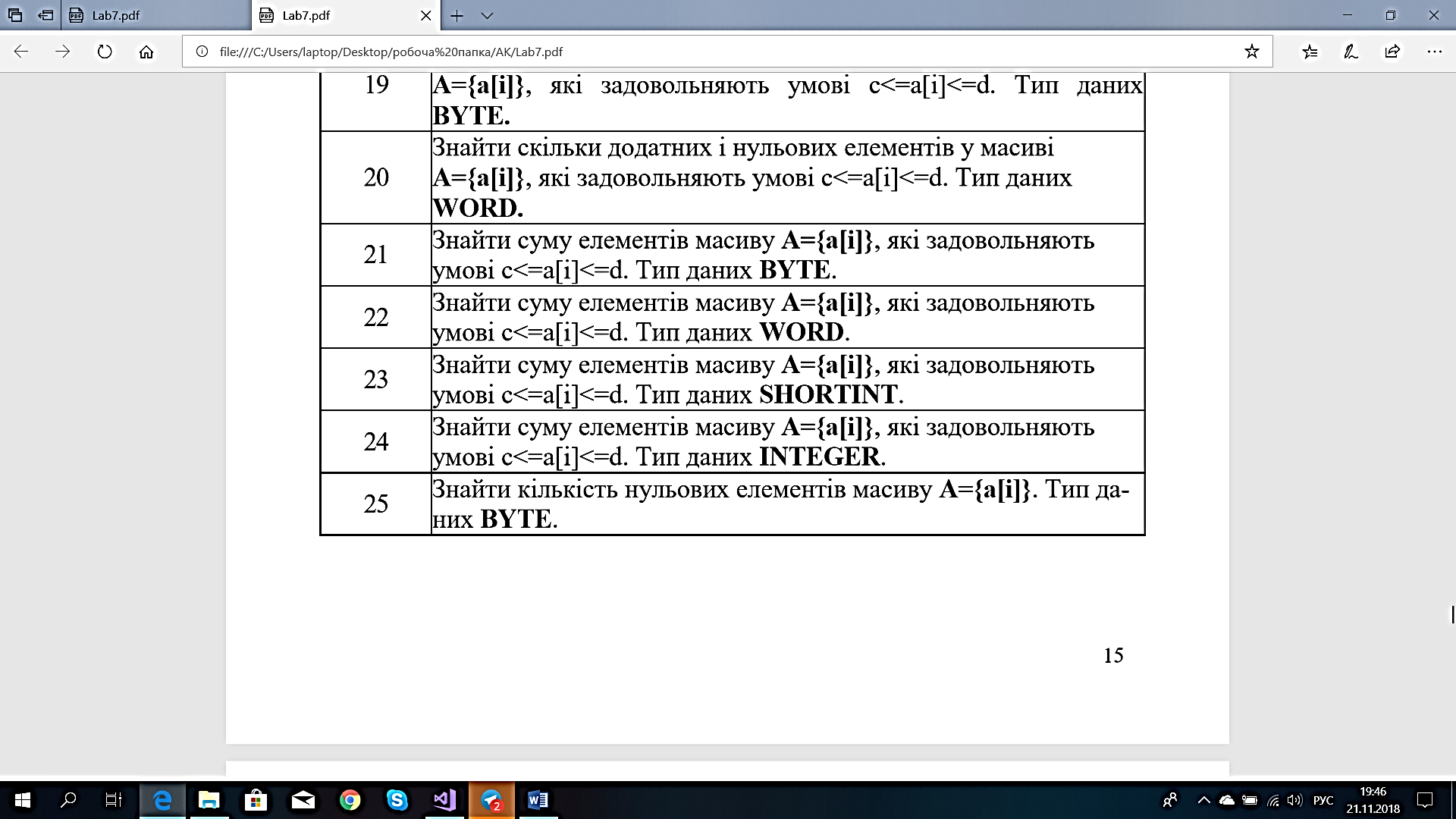
**ЛАБОРАТОРНА РОБОТА № 7**

**ОРГАНІЗАЦІЯ ЦИКЛІВ І РОБОТА З ЦІЛОЧИСЛЕНИМИ МАСИВАМИ**

***Мета:*** ознайомитися з основними командами мови Assembler для організації циклів і роботи з цілочисельними масивами; набути практичних навичок в написанні програм з використанням циклів та масивів на мові Assembler.

**Хід роботи:**

**Завдання 1**: Написати програму для обробки одномірного масиву для початкових даних в знаковому форматі. Виконати покрокове виконання асемблерного коду та навести значення регістрів при їх виконанні. Відмітити нормальні та аномальні результати, зробити аналіз результатів.



Лістинг програми:

#include "pch.h"

#include <iostream>

#include <stdlib.h>

#include <time.h>

#define N 10

int main()

{

short int a[N];

short int c = 0, d = 0;

printf("c<=a[i]<=d\n\n");

do {

printf("Enter the values of the range [32768...32767]:\n");

printf("c = "); scanf\_s("%d", &c);

printf("d = "); scanf\_s("%d", &d);

if (c >= d)

{

printf("c can not be greater or equal d! Enter values again.\n\n");

}

} while (c >= d);

int n = N;

short int res = 0;

for (int i = 0; i < N; i++)

{

a[i] = rand() % 10;

printf("A[%d] = %d\n", i, a[i]);

(a[i] >= c && a[i] <= d) ? res += a[i] : 0;

}

\_\_asm {

mov ax, c // <ax> = c

mov bx, d // <bx> = d

mov ecx, n // <ecx> = n

mov dx, 0 // <dx> = 0

dec ecx // зменшуємо значення в регістрі <ecx> на 1

cycle: // цикл cycle

shl ecx, 1 // зсув вліво на 1 розряд

mov si, a[ecx] // <si> = a[ecx]

cmp si, 0 // порівнюємо значення регістра <si> з 0

jl exit1 // якщо менше - перейти до циклу exit1

cmp si, ax // порівнюємо значення регістра <si> з значенням в регістрі <ax>

jl exit1 // якщо менше - перейти до циклу exit1

cmp si, bx // порівнюємо значення регістра <si> з значенням в регістрі <bx>

jg exit1 // якщо більше - перейти до циклу exit1

add dx, si // додаємо <dx> та <si>

exit1: // цикл exit1

shr ecx, 1 // зсув вправо на 1 розряд

dec ecx // зменшуємо значення в регістрі <ecx> на 1

cmp ecx, 0 // порівнюємо значення регістра <ecx> з 0

jnl cycle // поки не менше - перейти до циклу cycle

mov res, dx // res = <dx>

}

if (res > 32767 || res < -32768)

{

printf("Overflow!\n");

}

else

{

printf("Summa = %d\n", res);

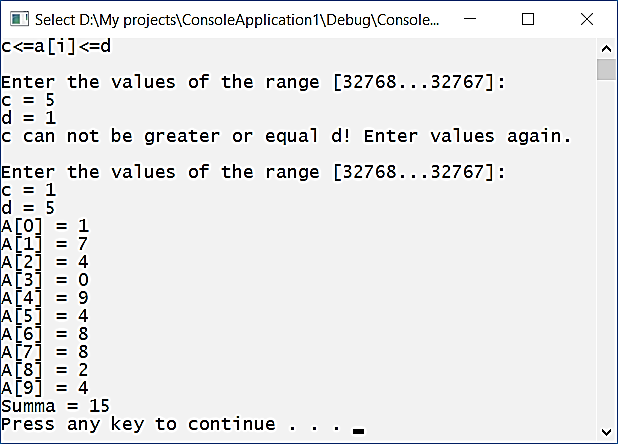
}

system("pause");

return 0;

}

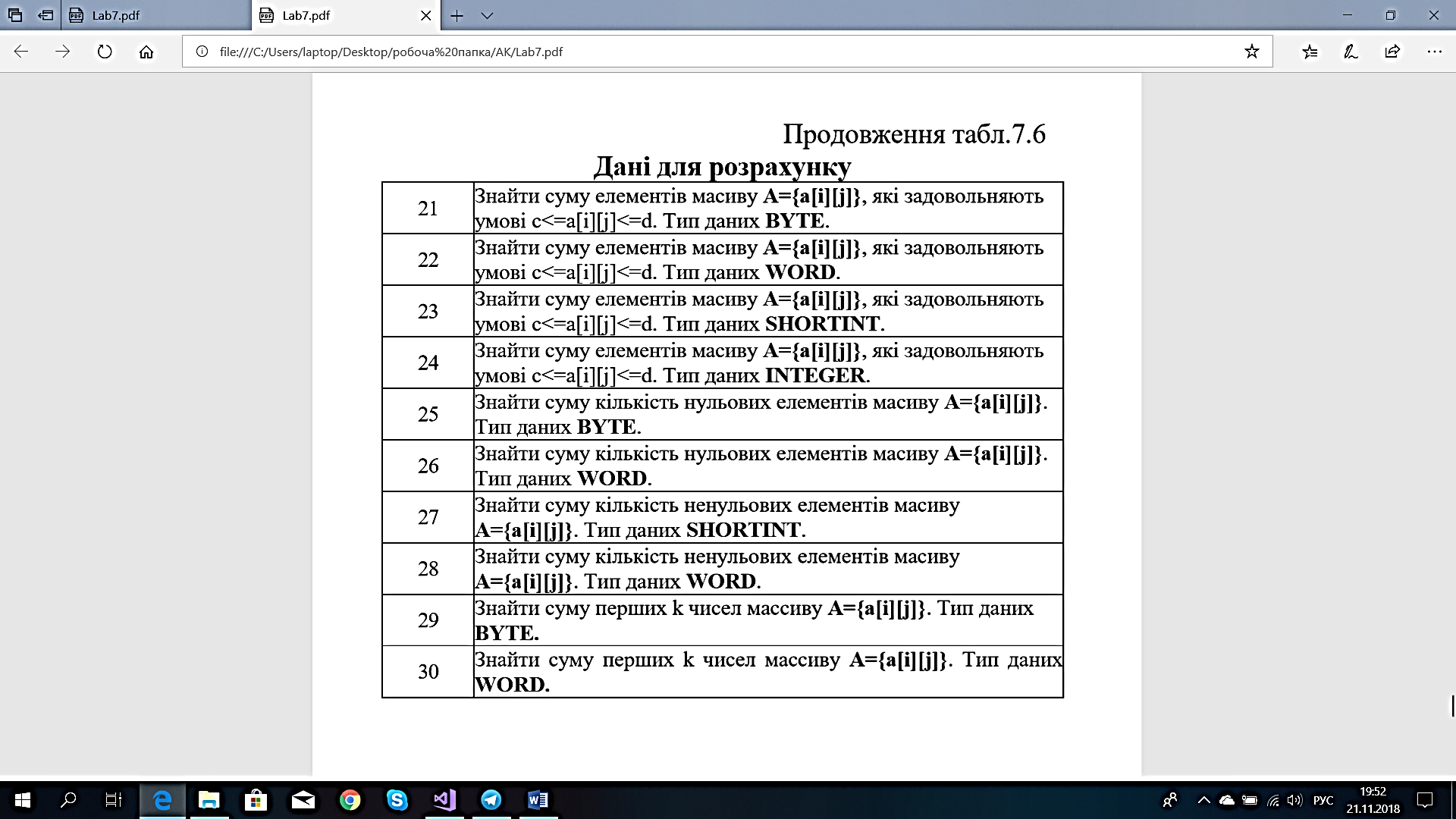
Результат виконання програми:



**Значення регістрів при покроковому виконанні**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Команда** | **Значення регістра** | | | | **EFLAGS/FLAGS (CF, OF)** |
| **ax** | **bx** | **eax** | **dx** |
| mov ax, c | 1 | н/в | н/в | н/в | **-** |
| mov bx, d | н/в | 5 | н/в | н/в | **-** |
| mov ecx, n | н/в | н/в | 10 | н/в | **-** |
| mov dx, 0 | н/в | н/в | н/в | 0 | **-** |
| dec ecx | н/в | н/в | 9 | н/в | **-** |
| cycle: | Мітка циклу cycle(код буде виконуватись в цій мітці, якщо умова – істина) | | | | |
| shl ecx, 1 | Зсув біта операнда вліво на 1 розряд | | | | |
| mov si, a[ecx] | Заносимо в регістр <si> елемент масиву з відповідним індексом | | | | |
| cmp si, 0 | Крок масиву порівнюється з 0 | | | | |
| jl exit1 | Якщо менше – перейти до мітки циклу exit1 | | | | |
| cmp si, ax | Порівнюємо елемент масиву з нижньою границею – c | | | | |
| jl exit1 | Якщо менше – перейти до мітки циклу exit1 | | | | |
| cmp si, bx | Порівнюємо елемент масиву з верхньою границею – d | | | | |
| jg exit1 | Якщо більше – перейти до мітки циклу exit1 | | | | |
| add dx, si | Додаємо значення регістрів <dx> та <si> | | | | |
| exit1: | Мітка циклу exit1(код буде виконуватись в цій мітці, якщо умова – істина) | | | | |
| shr ecx, 1 | Зсув біта операнда вправо на 1 розряд | | | | |
| dec ecx | Зменшуємо значення в регістрі <ecx> на 1 | | | | |
| cmp ecx, 0 | Порівнюємо значення регістра <ecx> з 0 | | | | |
| jnl cycle | Перейти на мітку циклу, якщо умова є неві | | | | |
| mov res, dx | н/в | н/в | н/в | 15 | **-** |

**Завдання 2**: Написати програму для обробки двовимірного масиву для початкових даних в знаковому форматі. Виконати покрокове виконання асемблерного коду та навести значення регістрів при їх виконанні. Відмітити нормальні та аномальні результати, зробити аналіз результатів.



Лістинг програми:

#include "pch.h"

#include "pch.h"

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <time.h>

#define N 5

int main()

{

short int a[N][N];

short int c = 0, d = 0;

do {

printf("c<=a[i]<=d\n\n");

printf("Enter the values c and d:\n");

printf("c = "); scanf\_s("%d", &c);

printf("d = "); scanf\_s("%d", &d);

printf("\n");

if (c >= d)

{

printf("c can not be greater or equal d! Enter values again.\n");

}

}

while (c >= d);

int n = N \* N; // загальна кількість елементів масиву

short int res = 0;

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

a[i][j] = rand() % 10;

printf("%d ", a[i][j]);

(a[i][j] >= c && a[i][j] <= d) ? res += a[i][j] : 0;

} printf("\n");

}

\_\_asm

{

mov bx, c // <bx> = c

mov dx, d // <dx> = d

mov ecx, n // <ecx> = n

lea si, a // завантаження зміщення масиву в регістр SI(замінює shr ecx, 1 та shl ecx, 1)

cycle: // цикл cycle

lodsw // завантажуємо слово з пам'яті, на який вказує регістр si

cmp ax, bx // порівнюємо з нижньою границею - c

jl next // якщо менше - перейти до циклу next

cmp ax, dx // порівнюємо з верхньою границею - d

jg next // якщо більше - перейти до циклу next

add dx, si // додаємо <dx> та <si>

next: // цикл next

dec ecx // зменшуємо к-ть чисел на 1

cmp ecx, 0 // порівнюємо к-ть чисел з 0

jnl cycle // якщо не менше - перейти до циклу cycle

}

if (res > 32767 || res < -32768)

{

printf("Overflow!\n");

}

else {

printf("\n");

printf("Summa = %d\n", res);

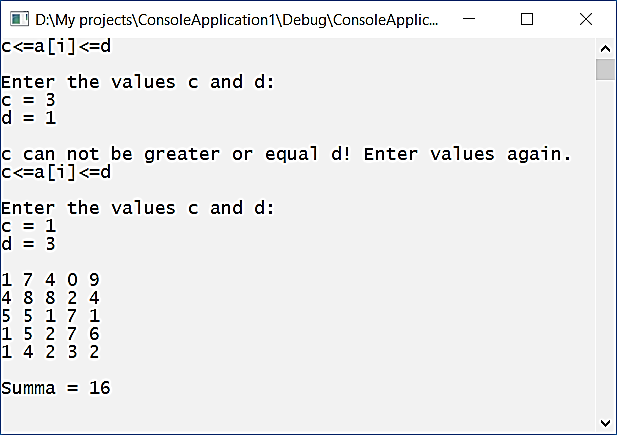
}

\_getch();

return 0;

}

Результат виконання програми:



Результат виконання програми для обробки двовимірного масиву

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Команда** | **Значення регістра** | | | **EFLAGS/FLAGS (CF, OF)** |
| **bx** | **dx** | **ecx** |
| mov bx, c | 1 | н/в | н/в | **-** |
| mov dx, d | н/в | 3 | н/в | **-** |
| mov ecx, n | н/в | н/в | 25 | **-** |
| lea si, a | Беремо в регістр <si> адресу першого елемента масиву | | | |
| cycle: | Мітка циклу cycle(код буде виконуватись в цій мітці, якщо умова – істина) | | | |
| lodsw | Завантажуємо слово з пам'яті, на який вказує регістр si | | | |
| cmp ax, bx | Порівнюємо елемент масиву з нижньою границею – c | | | |
| jl next | Якщо менше – перейти до мітки циклу next | | | |
| cmp ax, dx | Порівнюємо елемент масиву з верхньою границею – d | | | |
| jg next | Якщо більше – перейти до мітки циклу next | | | |
| add dx, si | Додаємо значення регістрів <dx> та <si> | | | |
| next: | Мітка циклу next(код буде виконуватись в цій мітці, якщо умова – істина) | | | |
| dec ecx | Зменшуємо значення в регістрі <ecx> на 1 | | | |
| cmp ecx, 0 | Порівнюємо значення регістра <ecx> з 0 | | | |
| jnl cycle | Поки індекс елементу масиву не менше 0 – перейти до циклу cycle | | | |

***Висновок:*** в ході виконання лабораторної роботи ознайомлено з основними командами мови Assembler для організації циклів і роботи з цілочисельними масивами; набутo практичних навичок в написанні програм з використанням циклів та масивів на мові Assembler.