|  |
| --- |
| Пермский филиал федерального государственного автономного образовательного учреждения высшего образования  «Национальный исследовательский университет  «Высшая школа экономики»  *Факультет экономики, менеджмента и бизнес-информатики* |
|  |
| Буров Никита Андреевич  **«КОМАНДЫ АРИФМЕТИКО-ЛОГИЧЕСКОГО УСТРОЙСТВА»**  Отчет по лабораторной работе  студента образовательной программы «Программная инженерия»  по направлению подготовки *09.03.04 Программная инженерия*   |  |  | | --- | --- | |  | Руководитель:  Доцент  кафедры информационных технологий в бизнесе  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Л.Н. Лядова | |

Пермь, 2019

# Задача 1.

Уточненная постановка задачи: Разработайте процедуру поиска первого (наиболее близкого к левому верхнему углу) «седлового» элемента матрицы: элемента, являющегося минимумом в строке и максимумом в столбце. Результат – индексы (номер строки и номер столбца) найденного элемента. Если элемент не найден, то в качестве результата – нули.

Сложность решения заключается в том, что двумерные массивы представлены в памяти компьютера как одномерные, вследствие чего необходимо пересчитывать смещение относительно начала, чтобы искать минимумы в строке.

Весь алгоритм сводится к двум шагам: необходимо найти минимум в строке, а потом проверить, является ли найденный элемент максимумом в столбце. Эти действия необходимо повторить пока не найдем «седловой» элемент, либо пока не пройдем все элементы матрицы. Таким образом, с учетом особенностей строения памяти компьютера алгоритм будет выглядеть следующим образом:

1. Инициализировать переменные для поиска значения минимума (min), индекса минимума в столбце (colIndex) и в строке (lineIndex). Временные переменные для работы: minIndex.
2. Инициализация счетчика для прохода по каждой строке.
3. Вычисления смещения строки двумерного массива относительно начала.
4. Поиск минимума в строке.
5. Проверка, является ли найденный элемент максимум в строке.
6. Если находим, что элемент – «Седловой», то выходим из цикла и завершаем программу. Иначе – увеличиваем счетчик пройденных строк и начинаем с первого шага.

Все числа в массиве – целые числа со знаком в формате двойного слова (int), поэтому при адресации массива необходимо использовать масштаб.

Для организации циклов использована команда loop. Чтобы упростить программу вычисление индекса столбца по заданному индексу элементу в памяти была создана процедура, которая через регистр ebx получала индекс в памяти, а возвращала индекс столбца.

## Листинг программы для решения задачи 1.

// Самостоятельная работа 1.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#include "pch.h"

#include <iostream>

using namespace std;

/\*

1) решение (найти первый седловой элемент в матрице)

\* Для каждой строки:

Найти минимум

Сравнить его со всеми элементами столбца - если он больше всех в строке - вывод

иначе - ищем дальше

2) решение (найти по седловому элементу в каждой строке/столбце)

Найти минимум

Сравнить его со всеми элементами столбца - если он больше всех в строке - вывод в массив

иначе - ищем дальше

3) решение (найти все седловые элементы)

\*/

int main()

{

const int N = 3; //Количество строк

const int M = 4; //Количество столбцов

static int A[N][M] = { {5,6,4,5},{-2,5,3,7},{8,7,-2,6} };

static int max, min, minIndex = 0, lineIndex = -1, colIndex = -1;

static char \_offset = 0;

\_asm {

\_\_START:push eax;

push ebx;

push ecx;

push edx;

push edi;

xor eax, eax;

xor ebx, ebx;

xor ecx, ecx;

xor edx, edx;

xor edi, edi;

mov ecx, N;

outer\_lbl: mov ax, word ptr[N]; // Записываем счетчик в ax для вычисления смещения

sub ax, cx; // Получаем количество пройденных строк

mov ebx, M; // Записываем второй операнд для умножения

mul bx; // Получаем смещение в ax

mov di, ax; // Запоминаем смещение

mov minIndex, 0; // Обнуление индекса столбца минимального элемента

push ecx;

mov ecx, M; // Записываем счетчик для прохода по строке

mov edx, A[4 \* edi]; // Инициализируем минимум - всегда первый элемент строки

mov min, edx;

dec ecx;

jmp find\_min\_lbl;

outer\_loop\_lbl\_transit: xor edi, edi;

jmp outer\_lbl;

find\_min\_lbl: inc edi; // Переходим к следующему элементу строки

mov eax, A[4 \* edi]; // Перемещаем новый элемент в регистр для сравнения

cmp eax, min; // Сравниваем элементы

jg not\_min; // Если элемент не максимальный - переходим к соотв. метке

mov min, eax; // Иначе - перезапоминаем минимум

push ebx;

mov ebx, M;

sub ebx, ecx;

call get\_column\_index; // Вычисление индекса столбца минимального элемента

mov minIndex, ebx; // Запоминаем индекс столбца

pop ebx;

not\_min: loop find\_min\_lbl;

mov ecx, N; // Заново инициализируем счетчик для прохода по столбцу

mov edi, minIndex; // Инициализация регистра-индекса значением столбца, где расположен минимальный элемент

find\_max\_lbl: mov eax, A[4 \* edi]; // Записываем минимум в регистр для дальнейшего сравнения

cmp eax, min; // Сравнение

jg if\_more; // Если найден новый максимальный элемент

add edi, M; // Увеличиваем счетчик для перехода к след. элементу

loop find\_max\_lbl; // Если элемент не найден - продолжаем цикл

pop ecx;

mov eax, N; // Записываем счетчик в ax для вычисления смещения

sub eax, ecx; // Получаем количество пройденных строк

mov lineIndex, eax; // Запоминаем индекс элемента

mov eax, minIndex;

mov colIndex, eax;

jmp outer\_loop\_lbl;

if\_more: pop ecx;

jmp outer\_loop\_lbl;

outer\_loop\_lbl: dec ecx;

cmp ecx, 0;

jg outer\_loop\_lbl\_transit;

jmp \_end;

get\_column\_index: push eax; // Метод получения индекса столбца по индексу в одномерном массиве

push ecx;

push edx;

xor eax, eax;

xor ecx, ecx;

xor edx, edx;

mov ax, bx;

mov bx, word ptr[M];

div bx;

mov bx, dx;

pop eax;

pop ecx;

pop edx;

ret;

\_end: pop edi;

pop edx;

pop ecx;

pop ebx;

pop eax;

}

cout << lineIndex << endl;

cout << colIndex << endl;

}

# Задача 2

Постановка задачи: Разработайте процедуру перевода числа со знаком, записанного в символьной строке, во внутреннее представление в памяти компьютера в формате байта/слова/двойного слова.

На вход поступает число, представленной символьной строкой. В программе входная строка является массивом символов (тип char[]).

Основной идеей алгоритма является то, что можно выполнить перевод строки в число, зная ASCII коды цифр, после чего перевести и это число в двоичную системы счисления. После того, как будет получена двоичная запись числа, по ее длине можно будет судить, какой формат будет подходящим для вывода результата. Для упрощения используются только форматы байта и слова со знаком.

Алгоритм перевода число во внутреннее представление выглядит следующим образом:

1. Инициализация переменных.
2. Проверка на то, является ли число отрицательным.
3. Перевод значимой части числа в десятичную систему счисления (тип int)
4. Если число отрицательное – выполнение преобразования в дополнительный код. Инвертирование выполняется при помощи команды btc.
5. Перевод числа в двоичную систему счисления с одновременным вычислением длины итогового числа в двоичной СС.
6. Представление числа в формате байта со знаком или слова в зависимости от вычисленной длины.

## Листинг задачи 2

// Задача 2.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#include "pch.h"

#include <iostream>

#include <string>

using namespace std;

/\* Если будет время допиши дополнительный код для отрицательных чисел

\*/

const char MINUS = '-';

int main()

{

//Перевод в двоичную

setlocale(LC\_ALL, "RUS"); //Установить Русский язык в консоли

char stroka[] = "-900";

const int l = sizeof(stroka)-1; //длина строки

char symb[] = { '\0' };

static int number = 0;

static bool is\_negative = false;

static bool is\_word;

char resDW[16] = { '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0' };

char resW[8] = { '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0' };

char count = 0;

\_\_asm {

push eax;

push ebx;

push ecx;

push edx;

push edi;

xor eax, eax;

xor ebx, ebx;

xor ecx, ecx;

xor edx, edx;

xor edi, edi;

mov ecx, l; // Запоминаем длину строки

mov edi, 1; // Степень 10

//Отрицательное число или нет

mov al, MINUS;

cmp al, stroka[0];

jne TO\_10SS;

mov is\_negative, 1;

//Перевод в 10СС

TO\_10SS: mov al, stroka[ecx - 1]; // Получение последней цифры-символа строки

sub al, 48; // Получение последней цифры-цифры

mul edi; // Умножение на 10 в степени

add number, eax; // Добавление к числу в десятичной СС

mov eax, 10; // Увеличение степени 10

mul edi;

mov edi, eax;

xor eax, eax; // Обнуление цифры

dec ecx;

mov bl, is\_negative;

cmp ecx, ebx

jne TO\_10SS;

mov eax, number;

cmp eax, 127;

jg IS\_NEGATIVE;

mov is\_word, 1;

IS\_NEGATIVE:mov al, 1;

cmp al, is\_negative; // Проверка отрицательное число или нет

jne INIT; // Если число не отрицательное, то сразу же переходим к переписыванию бит

xor edi, edi;

mov ecx, 16;

LOOP\_IS\_NEGATIVE: btc number, edi; // Цикл перевода в отрицательное число в памяти

inc edi;

loop LOOP\_IS\_NEGATIVE;

inc number;

INIT: xor edi, edi;

mov edi, 16; // По умолчанию считаем, что у нас двойное слово. Максимум - 15 цифр

mov ebx, 2;

TO\_2SS:

xor edx, edx;

mov eax, number;

div ebx; // Разделили число на 2

mov resDW[edi - 1], dl; // Записали остаток

dec edi; // Сдвинули индекс

mov number, eax; // Запомнили частное от деления

inc count;

cmp eax, 0; // Пока в eax что-то есть - продолжаем делить

jne TO\_2SS;

mov al, is\_word;

cmp al,1;

jl \_end;

mov ecx, 8;

mov edi, 15

LOOP\_LBL\_WORD: mov al, resDW[edi];

mov resW[edi-8], al;

dec edi;

loop LOOP\_LBL\_WORD;

\_end: pop edi;

pop edx;

pop ecx;

pop ebx;

pop eax;

}

if (is\_word)

for (int i = 0; i < sizeof(resW); i++)

cout<<(int)resW[i];

else

for (int i = 0; i < sizeof(resDW); i++)

cout<<(int)resDW[i];

}

# Задача 3

Постановка задачи: разработайте процедуру перевода двоичного числа со знаком из внутреннего представления в памяти компьютера в заданном формате (байта/слова/двойного слова) в символьную запись (в строку символов).

Основной идеей алгоритма является то, что сначала выполняется перевод в десятичную систему счисления, а после этого уже – преобразование в строку с форматированием для вывода (запись пробелов).

Алгоритм перевода числа из двоичной записи выглядит следующим образом:

1. Проверка, является ли число отрицательным.
2. Перевод значащей части в десятичную систему счисления.
3. Для каждой цифры числа:
   1. Выполнения перевода значения в код ASCII.
   2. Если число будет на границе классов (единицы, тысячи, миллионы), то необходимо написать пробел.
   3. Записать цифру числа.
4. Если число отрицательное – необходимо в строковую запись добавить спереди минус.
5. Вывести результат на печать.

Изначальной результат перевода записывается в массив символов (char[]), в котором 14 членов (с поправкой на возможную запись минуса и пробелов).

## Листинг задачи 3

/ Задача 3.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#include "pch.h"

#include <iostream>

const char MINUS = '-';

using namespace std;

int main()

{

//Перевод в десятичную

setlocale(LC\_ALL, "RUS"); //Установить Русский язык в консоли

char stroka[] = "111101000010010000000";

int len = sizeof(stroka) - 1;

unsigned int number = 0;;

bool is\_negative = 0;

static char res[14];

static int rankCount = 0;

\_\_asm {

push eax;

push ebx;

push ecx;

push edx;

push edi;

xor eax, eax;

xor ebx, ebx;

xor ecx, ecx;

xor edx, edx;

xor edi, edi;

mov edi, 1; //Запоминаем текущую степень 2-ки

mov ecx, len; //Запоминаем длину строки

//Отрицательное число или нет

mov al, MINUS;

cmp al, stroka[0];

jne TO\_10SS;

mov is\_negative, 1;

TO\_10SS: mov al, stroka[ecx - 1];

sub al, 48; //Получение последней цифры-цифры

mul edi; //Умножение на 2 в степени

add number, eax;

mov eax, 2; //увеличение 2-ки

mul edi;

mov edi, eax;

xor eax, eax; //Обнуление цифры

dec ecx;

mov bl, is\_negative;

cmp ecx, ebx;

jne TO\_10SS;

mov ecx, 14;

mov edi, 10;

TO\_CHAR\_ARRAY: xor edx, edx;

mov eax, number;

div edi;

mov number, eax;

cmp ecx, 14;

je WRITE\_LBL;

WRITE\_SPACES: push edx; //Запись пробелов в итоговое число

push eax;

push ebx;

xor edx, edx;

xor eax, eax;

xor ebx, ebx;

cmp rankCount, 3;

jne CONT\_LBL;

mov res[ecx - 1], ' ';

dec ecx;

mov rankCount,0;

CONT\_LBL: pop ebx;

pop eax;

pop edx;

push eax;

WRITE\_LBL:add dl, 48;

mov res[ecx - 1], dl; //запись цифры в массив

pop eax;

inc rankCount;

dec ecx;

cmp eax, 0;

jne TO\_CHAR\_ARRAY;

mov al, is\_negative;

cmp al, 1;

jne \_END;

mov res[ecx - 1], '-';

\_END: pop edi;

pop edx;

pop ecx;

pop ebx;

pop eax;

}

for (int i = 0; i < sizeof(res); i++)

cout << res[i];

}

# Задача 4

Постановка задачи: Разработайте процедуру, реализующую *операцию сложения для целых чисел*, представленных в двоично-десятичном формате. Разрядность чисел определите самостоятельно (разрядность числа может задаваться пользователем).

Основной идеей алгоритма является поразрядное сложение при помощи операция сложения двоично-десятичных чисел. Сложение производится в два этапа: сначала складываются разряды, которые есть в обоих числах, после чего разряды, которые есть в одном числе, но нет в другом – просто сносятся с учетом перехода через десяток.

Итоговый алгоритм выглядит следующим образом:

1. Между двумя строковыми представлениями чисел вычисляется разница в их длине.
2. В цикле вычисляется сумма разрядов, которые есть в обоих числах, после чего выполняется их запись в массив для вывода результата.
3. После того, как закончилось сложение разрядов, которые есть в обоих числах в зависимости от разницы в длине двух строк и значения флага CF после выполнения последней операции сложения выполняются действия по одному из 4-х сценариев:
   1. CF = 0, разница между длинами исходных чисел равна нулю: в этом случае больше не выполняется никаких действий.
   2. CF = 1, разница равна 0. Дополнительно дописывается 1 в ответ.
   3. CF=0, разница больше 1. В этом случае из числа, длина которого больше в ответ сносятся старшие разряды.
   4. CF=1, разница больше 1. В данном случае выполняется снос чисел с поправкой на флаг CF, пока на итерациях будет происходить переход через десяток.
4. Переворот итогового массива результата и вывод на печать.

Для сложения чисел использован оператор add. Для коррекции результата – aaa. Так как флаг CF после операций нужно сохранять, то была заведена ещё одна переменная is\_CF.

## Листинг задачи 4

// Задача 2 (нормальная версия).cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#include "pch.h"

#include <string>

#include <iostream>

using namespace std;

//Сумма двоично-десятичных чисел

int main()

{

static string A, B, result, reverse\_result;

static int lenA, lenB, difference, count = 4;

static bool is\_negative, is\_CF;

setlocale(LC\_ALL, "RUS"); // Установить Русский язык в консоли

printf("Введите первое число:\n"); // Ввод чисел

getline(cin, A);

printf("Введите второе число: \n");

getline(cin, B);

if (A.size() < B.size()) // Длина числа А всегда больше длины числа В

swap(A, B);

lenA = A.size(); // Запоминаем длины чисел

lenB = B.size();

\_\_asm {

\_\_START:push eax;

push ebx;

push ecx;

push edx;

push edi;

xor eax, eax;

xor ebx, ebx;

xor ecx, ecx;

xor edx, edx;

xor edi, edi;

DIFFERENCE:mov eax, lenA;

mov ebx, lenB;

sub eax, ebx;

mov difference, eax;

xor eax, eax;

xor ebx, ebx;

lea esi, [A];

lea edi, [B];

add esi, 4;

add edi, 4;

clc;

mov ecx, lenB;

SUM\_BOTH: add ecx, difference;

dec ecx; // Для удобства адресации

mov al, [esi + ecx]; // Загрузили цифру первого числа

sub al, 48;

aaa;

add al, is\_CF;

mov is\_CF, 0;

sub ecx, difference;

mov bl, [edi + ecx]; // Загрузили цифру второго числа

sub bl, 48;

aaa;

adc al, bl;

aaa;

jnc OUTPUT;

mov is\_CF, 1;

OUTPUT: add al, 48;

mov ebx, count;

mov[result + ebx], al; // Запись результата

inc count;

inc ecx;

loop SUM\_BOTH;

xor eax, eax;

xor ebx, ebx;

mov ecx, difference;

add al, is\_CF;

cmp al, 0;

jne IS\_CF; // Осталась ли единица

cmp difference, 0; // Есть ли разница между числами в длине

je \_REVERSE\_INIT;

mov ecx, difference;

LOOP\_LBL:dec ecx;

mov al, [esi + ecx];

mov ebx, count;

mov[result + ebx], al; // Запись результата

inc count;

inc ecx;

loop LOOP\_LBL;

jmp \_REVERSE\_INIT;

IS\_CF: cmp difference, 0;

jne LOOP\_LBL\_DIFF;

mov ebx, count;

mov[result + ebx], 49;

inc count;

jmp \_REVERSE\_INIT;

mov ecx, difference;

LOOP\_LBL\_DIFF: mov al, 0;

dec ecx;

mov bl, [esi + ecx];

add al, bl;

add al, is\_CF;

aaa;

jnc NO\_OVERFLOW;

mov is\_CF, 1;

jmp \_OUTPUT\_2;

NO\_OVERFLOW: mov is\_CF, 0;

\_OUTPUT\_2:mov ebx, count;

add al, 48;

mov[result + ebx], al; //Запись результата

inc count;

inc ecx;

xor eax, eax;

loop LOOP\_LBL\_DIFF;

cmp is\_CF, 1;

jne \_REVERSE\_INIT;

mov ebx, count;

add al, 48;

mov[result + ebx], 49;

inc count;

\_REVERSE\_INIT:

xor eax, eax;

add eax, 4;

mov ecx, count;

LOOP\_LBL\_REVERSE: dec ecx;

mov bl, [result + ecx];

mov[reverse\_result + eax], bl;

inc eax;

inc ecx;

dec ecx;

cmp ecx, 4;

jne LOOP\_LBL\_REVERSE;

pop edi;

pop edx;

pop ecx;

pop ebx;

pop eax;

}

printf(reverse\_result.c\_str());

}