Министерство науки и высшего образования РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт инженерной физики и радиоэлектроники

Радиоэлектронная техника информационных систем

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ**

Лабораторная работа №2

Изучение функций и процедур

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Преподаватель | |  |  | \_\_\_\_\_\_\_\_\_\_\_\_ | Т. Н. Батурин |
|  | |  |  | подпись, дата |  |
| Студент | РФ19-32Б | | | \_\_\_\_\_\_\_\_\_\_\_\_ | И. А. Негодеева | |
|  |  | | | подпись, дата |  | |

Красноярск 2022

**1 Цель работы**

Выполнить задание лабораторной №1 на языке Си, используя функции и указатели. Переписать программу, написанную в лабораторной работе №1 с помощью процедур языка ассемблера. Вызвать процедуры, написанные на ассемблере в коде на языке Си. Сравнить результат выполнения функций на языке Си и на языке ассемблера.

**2 Ход работы**

Дан одномерный массив, состоящий из чисел, написать программу для расчета среднего арифметического отрицательных элементов массива и поиска минимального элемента массива.

Код выполнения лабораторной работы представлен в приложениях приложение А – В.

**ПРИЛОЖЕНИЕ А**

**Файл main.c**

#include <stdio.h> /\* �-аголовочный файл объявляет несколько целочисленных типов и макросов \*/

#include <stdint.h> /\* �-аголовочный файл объявляет несколько целочисленных типов и макросов \*/

typedef struct{

int32\_t b;

int32\_t c;

}new\_t;

new\_t s;

// �нициализируем первичный массив с числами

int32\_t array\_original [] = {1, 1, 4, -2, -6, 0, 3, 2, 1, 3};

// �нициализируем указатель на результат (ответ). Начальное значение NULL

int32\_t\* array = NULL;

// Объявляем прототип функции, которая будет выполнять задание

int32\_t\* array\_treatment (int32\_t\* array\_ptr, size\_t array\_size);

// Объявляем прототип функции, которая будет вычислять сумму элементов массива

int32\_t array\_sum (int32\_t\* array\_ptr, size\_t array\_size);

new\_t les(int32\_t\* array\_ptr, size\_t array\_size);

// Объявляем прототип функции, которая будет производить

// расчет среднего арифметического отрицательных элементов

int32\_t avg\_calc(int32\_t\* array\_ptr, size\_t array\_size);

// Объявляем прототип функции, которая будет производить

// поиск минимального элемента массива

int32\_t find\_min(int32\_t\* array\_ptr, size\_t array\_size);

// Прототип функции варианта

new\_t var5(int32\_t\* array\_ptr, size\_t array\_size);

//Экспортируем функцию из main.s написанную на ассемблере.

/////вызвать наверное только var\_5 роднулечку

extern int32\_t var\_5 (int32\_t\* array\_ptr, size\_t array\_size, int32\_t\* new\_array\_ptr );

//extern int32\_t array\_sum\_asm (int32\_t\* array\_ptr, size\_t array\_size);

//extern int32\_t find\_min\_func (int32\_t\* array\_ptr, size\_t array\_size);

// Объявляем пустой массив для передачи его адреса в функцию array\_treatment\_asm

int32\_t new\_array\_ptr [sizeof(array\_original)/sizeof(int32\_t)]= {0};

int main (void)

{

s.b = 5;

s.c = 10;

// вывод через **return** 2 значения

/\*�'ызываем функцию array\_treatment.

Функция возвратит указатель на массив соответствующий заданию.

Размерность массива такая же как у оригинального массива.\*/

new\_t a;

a = les(array\_original, sizeof(array\_original)/sizeof(int32\_t));

/\*�'ызываем функцию array\_treatment\_asm, тело которой описано в файле main\_1.s. Так как, в языке ассемблера не функций malloc() и вообще нет механизма выделения данных из кучи, то мы заранее выделяем место в оперативной памяти для обработанного массива, поэтому нам необходимо передать адрес нового массива в функцию. Размерность массива такая же как у оригинального массива.\*/

var\_5(array\_original, sizeof(array\_original)/sizeof(int32\_t), new\_array\_ptr);

while (1)

{

}

**return** 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Функция для расчета среднего арифметического

// и поиска минимального элемента массива

new\_t les(int32\_t\* array\_ptr, size\_t array\_size)

{

new\_t v;

v = var5(array\_ptr, array\_size);

**return** v;

}

//\*\*\*\*\*\*\*Функция варианта 5\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

new\_t var5(int32\_t\* array\_ptr, size\_t array\_size)

{

new\_t b;

b.b = avg\_calc(array\_ptr, array\_size);

b.c = find\_min(array\_ptr, array\_size);

**return** b;

}

//\*\*\*\*\*\*\*\*Расчет среднего арифметического отрицательных элементов\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int32\_t avg\_calc(int32\_t\* array\_ptr, size\_t array\_size)

{

// masive - указатель на массив (адрес массива)

// masive \_size - размер массива

int i; // i - элемент массива

int sn, n, sredar;

sn = 0;

n = 0;

sredar = 0;

// Поиск отрицательного элемента массива

for (i = 0; i < array\_size; i++)

{

if (array\_ptr[i] < 0)

{

n = n + 1;

sn = sn + array\_ptr[i];

}

}

// Среднее арифметическое

sredar = sn / n;

**return** sredar;

}

//\*\*\*\*\*\*\*\*\*\*Поиск минимального элемента массива\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int32\_t find\_min(int32\_t\* array\_ptr, size\_t array\_size)

{

int i; // i - элемент массива

int nm;

nm = array\_ptr[0]; // Титул наименьшего = нулевому элементу массива

for (i = 0; i < array\_size; i++)

{

if (array\_ptr[i] < nm)

{

nm = array\_ptr[i];

}

}

**return** nm;

}

**ПРИЛОЖЕНИЕ Б**

**Файл main\_1.s**

AREA |.text|, CODE, READONLY

array\_sum\_asm PROC ; Начало функции array\_sum\_asm

EXPORT array\_sum\_asm

; Функция (процедура) реализует суммирование массива

; Функция принимает указатель на массив через регистр R0

; размер массива через регистр R1.

; Результат работы функции возвращается через регистр R0

; R0 = \*array

; R1 = size(array)

; return R0 = sum\_array

MOV R3, #0 ; R3 = "i" = 0 - элемент массива

MOV R4, #0 ; R4 = 0 - обнуление регистра R3

MOV R5, #0 ; R5 = "sn" = 0 - сумма отрицательных элементов

MOV R6, #0 ; R6 = "n" = 0 - кол-во отрицательных элементов

MOV R7, #4 ; Размер одного элемента массива = 4 байтам,

MUL R1, R7 ; Поэтому умножаем кол-во элементов массива на 4.

CVTR ; Метка CVTR

CMP R3, R1 ; Проверка "i" с "masive\_size"

BLT CAJ ; Если i < masive\_size, переход в "CAJ"

B SRZN ; �'езусловный переход в "SRZN" - выполняется только, когда не произошел переход в "CAJ"

CAJ ; Метка CAJ

; �-апись в регистр "R3" значения по адресу сохран�'нном в "R2"

LDRB R4, [R0, R3] ; R4 = mas[i]

SXTB R4, R4 ; Расширение знака

CMP R4, #0 ; Сравнить mas[i] с 0

BLT JNT ; Если mas[i] < 0, то переходим в JNT

B OUTXX ; �'езусловный переход в OUTXX

JNT ; Метка JNT

SXTB R5, R5 ; Расширение знака

ADD R5, R4 ; sn = sn + mas[i]

ADD R6, #1 ; n = n + 1

OUTXX ; Метка OUTXX

ADD R3, #1 ; i = i + 1

ADD R3, #4

B CVTR ; Переход в начало подпрограммы

; Среднее арифметическое

CMP R6, #0 ; Сравнение n с нулем

BNE SRZN ; Если не равно нулю, то переход в SRZN

B OUTEX ; �наче переход в OUTEX

SRZN ; Метка SRZN

SDIV R5, R5, R6 ; Среднее арифм.

; �-аписать значение результата в регистр "R0"

OUTEX ; Метка OUTEX

SXTB R0, R0 ; Расширение знака

MOV R0, R5 ; �-апись R5 в R0

BX LR ; �'озвращение по адресу, сохраненном в LR

ENDP ; Конец подпрограммы array\_sum\_asm

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

find\_min\_func PROC

EXPORT find\_min\_func

; Функция (процедура) реализует обработку массива в соответствии с заданием

; Функция принимает указатель на массив через регистр R0,

; размер массива через регистр R1,

; указатель на пустой массив через регистр R2

; R0 = \*array

; R1 = size(array)

; R2 = \*new\_array

MOV R3, #0 ; R3 = "i" = 0 - элемент массива

MOV R4, #0 ; R4 = 0 - обнуление регистра R3

MOV R5, #0 ; R5 = "nm" = 0 - наименьший элемент

MOV R6, #4 ; Размер одного элемента массива = 4 байтам,

MUL R1, R6 ; Поэтому умножаем кол-во элементов массива на 4.

KYTR ; Метка KYTR

CMP R3, R1 ; Проверка "i" c "masive\_size"

BLT CAT ; Если i < MAX\_size, переход в "CAT"

B OUTEXX ; �'езусловный переход в "OUTEXX" - выполняется только, когда не произошел переход в "CAT"

CAT ; Метка CAT

; �-апись в регистр "R3" значения по адресу сохран�'нном в "R2"

LDRB R4, [R0, R3] ; R4 = mas[i]

SXTB R4, R4 ; Расширение знака

CMP R4, #0 ; Сравнить mas[i] с 0

BLT LKJ ; Если mas[i] < 0, то переходим в LKJ

B OUTXXX ; �'езусловный переход в OUTXXX

LKJ ; Метка LKJ

CMP R4, R5 ; Сравнение mas[i] с nm

BLE SRAW ; Если mas[i] < nm, то переход в SRAW

B OUTXXX ; �наче переход в OUTXXX

SRAW ; Метка SRAW

MOV R5, R4 ; �-апись R5 = R4, присвоили R5 Титул элемента с минимальным значением

OUTXXX ; Метка OUTXX

ADD R3, #1 ; i = i + 1

ADD R3, #4

B KYTR ; Переход в начало подпрограммы

OUTEXX ; Метка OUTEXX

SXTB R0, R0

MOV R0, R5 ; �-апись R5 в R0

BX LR ; �'озвращение по адресу, сохраненном в LR

ENDP ; Конец ПП find\_min\_func

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

var\_5 PROC ; Начало функции array\_treatment\_asm

EXPORT var\_5

; Функция (процедура) реализует обработку массива в соответствии с заданием

; Функция принимает указатель на массив через регистр R0,

; размер массива через регистр R1,

; указатель на пустой массив через регистр R2

; R0 = \*array

; R1 = size(array)

; R2 = \*new\_array

**PUSH** {LR}

MOV R3, #0 ; R3 = "i" = 0 - элемент массива

; �'ызываем процедуру вычисления средн. арифм. отриц. эл. массива

**PUSH** {R0-R4, LR} ; �-аписываем все необходимые регистры в стек, необходимые для процедуры array\_treatment\_asm

; в регистрах R0 и R1 уже лежит необходимые данные для вычисления суммы

BL array\_sum\_asm

MOV R4, R0 ; �-апись суммы массива в регистр R4

STR R4, [R4] ; Соханение содержимого регистра R4 в памяти по адресу в регистре R4

**POP** {R0-R4, LR} ; возвращаем записанные ранее регистры из стека

; �'ызываем процедуру поиска минимального элемента массива

**PUSH** {R0-R7, LR} ; �-аписываем все необходимые регистры в стек, необходимые для процедуры array\_treatment\_asm

; в регистрах R0 и R1 уже лежит необходимые данные для вычисления суммы

BL find\_min\_func

MOV R7, R0 ; �-апись суммы массива в регистр R4

STR R7, [R7] ; Соханение содержимого регистра R4 в памяти по адресу в регистре R4

**POP** {R0-R7, LR} ; возвращаем записанные ранее регистры из стека

**POP** {LR}

BX LR

END ; Конец файла

**ПРИЛОЖЕНИЕ В**

**Файл startup.s**

Stack\_Size EQU 0x400

AREA STACK, NOINIT, READWRITE, ALIGN=3

Stack\_Mem SPACE Stack\_Size

\_\_initial\_sp

; <h> Heap Configuration

; <o> Heap Size (in Bytes) <0x0-0xFFFFFFFF:8>

; </h>

Heap\_Size EQU 0x200

AREA HEAP, NOINIT, READWRITE, ALIGN=3

\_\_heap\_base

Heap\_Mem SPACE Heap\_Size

\_\_heap\_limit

PRESERVE8

THUMB

; Vector Table Mapped to Address 0 at Reset

AREA RESET, DATA, READONLY

DCD \_\_initial\_sp ; Top of Stack

DCD Reset\_Handler ; Reset Handler

AREA |.text|, CODE, READONLY

; Reset handler

Reset\_Handler PROC

IMPORT \_\_main

LDR R0, =\_\_main

BX R0

; B main\_asm

ENDP

IMPORT \_\_use\_two\_region\_memory

EXPORT \_\_user\_initial\_stackheap

\_\_user\_initial\_stackheap

LDR R0, = Heap\_Mem

LDR R1, =(Stack\_Mem + Stack\_Size)

LDR R2, = (Heap\_Mem + Heap\_Size)

LDR R3, = Stack\_Mem

BX LR

ALIGN

END

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* (C) COPYRIGHT STMicroelectronics \*\*\*\*\*END OF FILE\*\*\*\*\*