

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В. ЛОМОНОСОВА
ФАКУЛЬТЕТЕ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ

ОТЧЕТ ПО ЗАДАНИЮ № 2
«Сборка многомодульной программы.
Обработка символьных данных»
Вариант 3 / 2 / 1

Выполнил:
студент 141 группы
Мустафин Ренат Рафаэлевич

Преподаватель:
Кузьменкова Евгения Анатольевна

Москва
2023

Содержание

Постановка задачи	3
Структура программы и спецификация функций	4
Сборка программы	7
Отладка программы, тестирование функций	8
Анализ допущенных ошибок	9
Литература	10

Постановка задачи

В данной лабораторной работе необходимо реализовать программу, которая будет проверять введенный текст (непустая последовательность) на соответствие заданному свойству, а также преобразовывать его в соответствии с заданными правилами.

Свойство текста:

Текст начинается латинской буквой и оканчивается латинской буквой.

Правила преобразования:

1. Заменить каждую ненулевую цифру на соответствующую ей по порядковому номеру строчную букву латинского алфавита ($1 \rightarrow a$, $2 \rightarrow b$ и т.д.).
2. Перенести в начало текста все входящие в него цифры с сохранением порядка их следования.

Программа должна быть реализована в виде многомодульной программы, где на языке Си реализованы функции ввода текста, проверки текста на соответствие заданному свойству, а также функция вывода текста, а на языке ассемблера NASM реализованы функции преобразования текста в соответствии с заданными правилами

Структура программы и спецификация функций

Программа состоит из следующих модулей:

1. **main.c** - основной модуль программы, в котором расположена функция `main`, а также функции ввода, проверки текста на соответствие заданному свойству, и вывода текста.
2. **transform.h** - заголовочный файл программы, в котором расположена спецификация функций преобразования текста.
3. **transform.asm** - модуль, с функциями преобразования текста в соответствии с заданными правилами.

Спецификация функций:

Главный файл `main.c`

```
#include "transform.h"
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define N 101

int main() {
    char str[N];

    printf("Enter a string (max 100 characters): ");
    scanf("%s", str);

    int len = strlen(str);

    // Check if first & last characters are letters
    if (isalpha(str[0]) && isalpha(str[len - 1])) {
        printf("Rule 1 usage\n");
        printf("Derived string:\n%s\n", rule1(str));
        return 0;
    } else {
        // Any other case
        printf("Rule 2 usage\n");
        printf("Derived string:\n%s\n", rule2(str));
        return 0;
    }

    return 0;
}
```

Заголовочный файл transform.h

```
/*
Header file for lib.asm file. It contains prototypes of functions that are
implemented in lib.asm file
*/

#ifndef __LIB_H__
#define __LIB_H__

// Function prototypes
char *rule1(char *);
char *rule2(char *);

#endif
```

Функция для преобразования по правилу 1

```
rule1:
    push ebp                ; Пролог
    mov ebp, esp           ; функции

    mov eax, dword[ebp + 8] ; Получаем адрес строки из стека
    xor ecx, ecx           ; Обнуляем счётчик
    mov dl, 'a'            ; Помещаем в регистр dl символ 'a'
    sub dl, '1'            ; Вычитаем из регистра dl символ '1'
    .cycle:
        cmp byte[ebp + ecx], 0 ; Сравниваем байт в памяти с 0
        je .out              ; Если 0, то выходим из цикла
        cmp byte[ebp + ecx], '1' ; Сравниваем байт в памяти с символом '1'
        jb .next             ; Если меньше '1', то переходим к следующему байту
        cmp byte[ebp + ecx], '9' ; Сравниваем байт в памяти с символом '9'
        ja .next             ; Если больше '9', то переходим к следующему байту
        add byte[ebp + ecx], dl ; Добавляем к байту в памяти значение dl
    .next:
        inc ecx              ; Увеличиваем счётчик
        jmp .cycle           ; Переход к началу цикла
    .out:
        mov esp, ebp        ; Эпилог
        pop ebp             ; функции
        ret
```

Функция для преобразования по правилу 2

```
rule2:
    push ebp                                ; Пролог
    mov ebp, esp                            ; функции

    mov eax, dword[ebp + 8]
    xor ecx, ecx                            ; счётчик

.cycle1:
    cmp byte[ebp + ecx], 0                 ; сравниваем байт в памяти с 0
    je .out1                              ; если 0, то выходим из цикла
    cmp byte[ebp + ecx], '0'              ; сравниваем байт в памяти с символом '0'
    jb .next1                             ; если меньше '0', то переходим к следующему байту
    cmp byte[ebp + ecx], '9'              ; сравниваем байт в памяти с символом '9'
    ja .next1                             ; если больше '9', то переходим к следующему байту
    movzx edx, byte[ebp + ecx]             ; переносим байт в регистр и расширяем его до dword
    push edx                              ; помещаем значение в стек

.next1:
    inc ecx                                ; увеличиваем счётчик
    jmp .cycle1                           ; повторяем цикл

.out1:
    xor ecx, ecx                          ; обнуляем счётчик

.cycle2:
    cmp byte[ebp + ecx], 0                 ; сравниваем байт в памяти с 0
    je .out2                              ; если 0, то выходим из цикла
    cmp byte[ebp + ecx], '0'              ; сравниваем байт в памяти с символом '0'
    jb .next2                             ; если меньше '0', то переходим к следующему байту
    cmp byte[ebp + ecx], '9'              ; сравниваем байт в памяти с символом '9'
    ja .next2                             ; если больше '9', то переходим к следующему байту
    jmp .countinc                         ; если в диапазоне, то переходим к следующему байту
    cmp byte[ebp + ecx], '9'              ; если не в диапазоне, то переходим к следующему байту
    movzx edx, byte[ebp + ecx]             ; переносим байт в регистр и расширяем его до dword
    push edx                              ; помещаем значение в стек

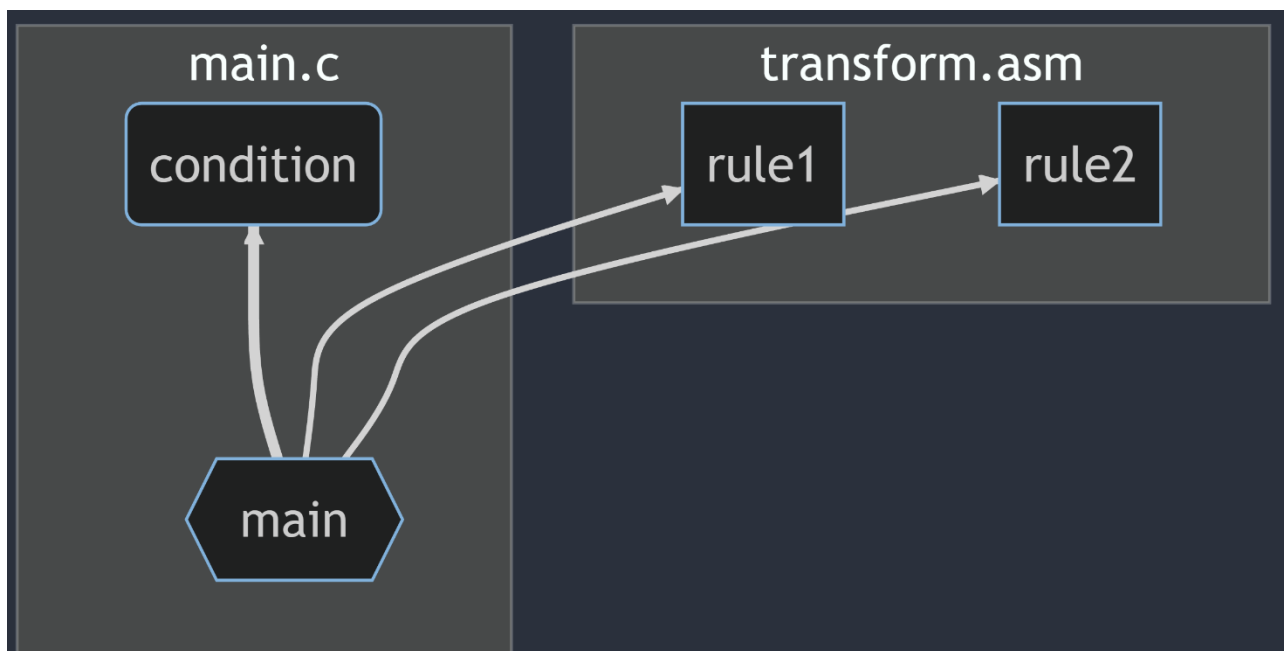
.countinc:
    inc ecx                                ; увеличиваем счётчик
    jmp .cycle2                           ; повторяем цикл

.out2:
    mov byte[ebp + ecx], 0                 ; помещаем в конец строки 0

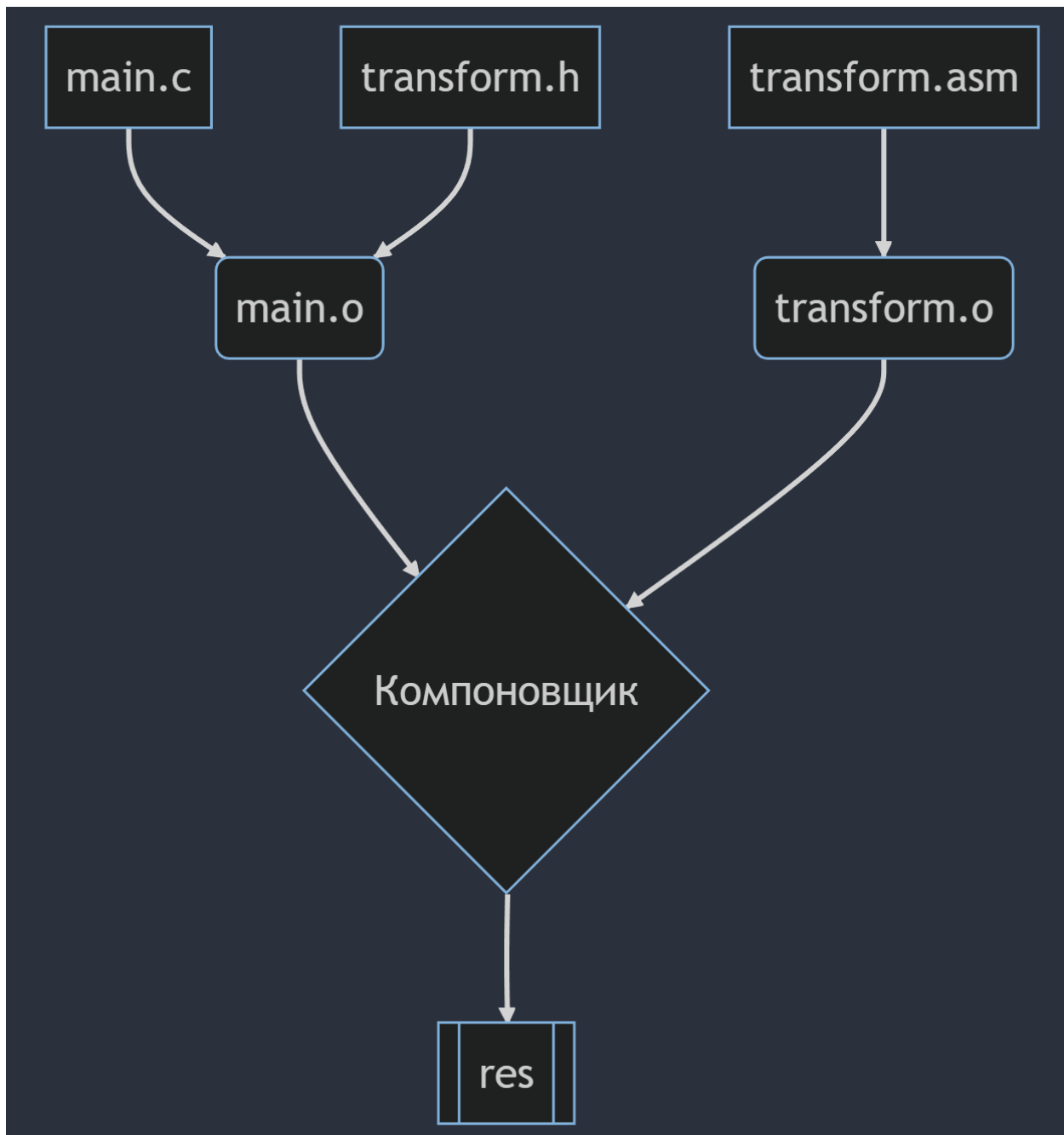
.ans:
    pop edx                               ; достаём значение из стека
    mov byte[ebp + ecx - 1], dl           ; помещаем значение в память
    loop .ans                            ; уменьшаем счётчик и переходим в начало цикла, пока счётчик не станет равен 0

    mov esp, ebp                          ; Эпилог
    pop ebp                              ; функции
    ret
```

Весь код задокументирован внутри файлов, что упрощает понимание действий



Сборка программы



Для удобства сборки программы был написан Makefile, который содержит следующие команды:

```
all: run

transform.o: transform.asm
    nasm -W+all -g -f elf32 -D UNIX transform.asm -o transform.o
main: main.c transform.h transform.o
    gcc -Wall -m32 main.c transform.o -o main
run: main
    ./main
```

Для сборки программы необходимо выполнить команду ``make`` в терминале. Для запуска программы необходимо выполнить команду ``make run`` в терминале.

Отладка программы, тестирование функций

Для тестирования программы были использованы следующие тесты:

Входная строка	Правило	Ожидаемый результат	Результат работы программы
1	2	1	1
;/	2	;/	;/
1a2b3c	2	123abc	123abc
aHGHJD1234GHFDz	1	aHGHJDabcdGHFDz	aHGHJDabcdGHFDz
HFHJ123IKH	1	HFHJabcIKH	HFHJabcIKH
akjsdlwa	1	akjsdlwa	akjsdlwa

Анализ допущенных ошибок

В ходе написания и тестирования программы не было допущено ошибок

Литература

1. Трифонов Н.П., Пильщиков В.Н. Задания практикума на ЭВМ (1 курс). Методическая разработка. – М.: ВМК МГУ, 2001.