Министерство науки и высшего образования Российской Федерации

Санкт-Петербургский политехнический университет Петра Великого

Высшая школа программной инженерии

КУРСОВАЯ РАБОТА

Программирование на языке Ассемблера

по дисциплине «Архитектура компьютера»

Выполнил _____ К. П. Баранов

студент гр. в5130904/20021

Руководитель _____ С. А. Молодяков

профессор, д.т.н

«__» _____ 2023 г.

Санкт-Петербург

2023

Содержание

Задание на курсовую работу

Разработать игру: кто придумает больше слов из символов заданной строки.

Проверять:

— на допустимость по символам;

— по словарю, есть ли такие слова.

Предусмотреть возможность дополнения словаря и игнорирование высоты букв (А = а).

Блок-схемы алгоритмов задач и их описания

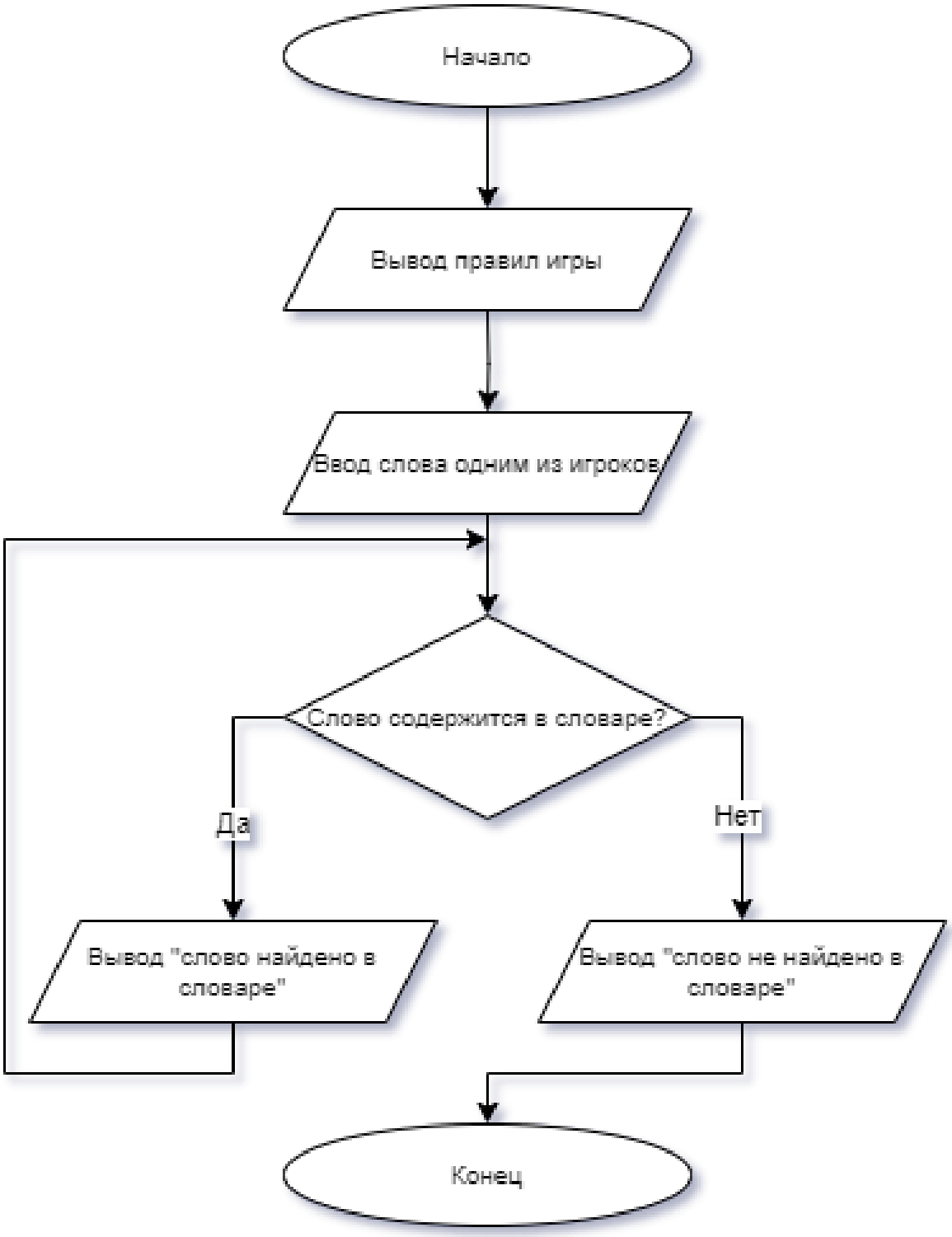


Рисунок 1 – Блок-схема алгоритма игры в слова.

Представленная блок-схема демонстрирует логику игры в слова. На начальном этапе выполнения программы выводится на экран список правил игры и исходную строку, из букв которой игроки должны составлять слова. Два игрока поочередно вводят свои слова в терминал.

Согласно техническому заданию к курсовой работе, слова, вводимые игроками, должны быть составлены из символов исходной строки и представлять собой реальные слова, а не произвольные комбинации букв. Для обеспечения этой проверки используется предварительно заданный словарь, содержащий все допустимые слова, которые можно составить из букв исходной строки. Такой подход снимает необходимость проверки пользовательского ввода на соответствие символам исходной строки.

Далее выполняются алгоритмы, которые сканируют словарь на полное посимвольное совпадение с каждым введённым игроком словом. Если введённое слово найдено в словаре, на экране терминала появляется сообщение о том, что слово успешно обнаружено в словаре, и игра продолжается. Следующий игрок приглашается ввести новое слово.

В случае, если введённое слово отсутствует в словаре, выводится информационное сообщение о завершении игры. Это сигнализирует о завершении работы всего приложения, и программа завершает выполнение.

Таким образом, логика программы опирается на предварительно созданный словарь для проверки допустимости слов, что обеспечивает более эффективный и надёжный процесс игры.

Таблица используемых функций операционной системы

В представленном исходном коде игры в слова **не прямо** используются системные вызовы для взаимодействия с операционной системой.

Ответственность за их использование перенесена на внешние функции из библиотеки Си (*scanf, printf, strtok, strcmp*). Эти функции предоставляются стандартной библиотекой языка С и являются высокоуровневыми интерфейсами для ввода-вывода и обработки строк.

Непосредственно в коде игры не происходит прямого вызова системных вызовов для ввода-вывода. Однако ввод/вывод осуществляется с помощью описанных разработчиком макросов *Cprintf* и *Cscanf*, которые в свою очередь вызывают соответствующие внешние функции из библиотеки языка С, использующие, в том числе, такие системные вызовы операционной системы как «*SYS_WRITE*» с кодом «0» и «*SYS_READ*» с кодом «1».

Завершение работы процесса (программы) осуществляется автоматически с помощью применения всё той же библиотеки С:

```
; Someone accidently or by purpose entered the wrong word.
game_over:
  Cprintf incorrect_msg, string_format
  xor     eax, eax ; Set zero (EXIT_SUCCESS).
  leave
  ret
```

Инструкция «*xor eax, eax*» устанавливает значение регистра EAX в ноль, что является кодом успешного завершения программы (*EXIT_SUCCESS*).

Инструкция «*leave*» в ассемблере x86-64 используется для уничтожения текущего стекового фрейма (stack frame) в процессе завершения работы подпрограммы. Она эффективно выполняет две операции: восстанавливает значение указателя стека (RSP) и базового указателя кадра (RBP) до состояния, которое было на момент входа в текущую подпрограмму. Использование инструкции «leave» удобно, поскольку она автоматически выполняет необходимые операции для завершения подпрограммы и восстановления предыдущего состояния стека.

Это делает код более компактным и понятным, освобождая программиста от ручного управления указателем кадра и стеком при выходе из функций.

Инструкция «*ret*» применяется для возврата из функции. В данном случае, так как присутствует только одна функция «*main*», эта инструкция фактически завершает выполнение программы.

Таким образом, завершение программы происходит неявно с использованием «*SYS_EXIT*» (код системного вызова 60), и код завершения устанавливается в ноль, что указывает на успешное завершение программы.

Таким образом, можно описать следующую таблицу используемых системных функций:

Название функции	Краткое описание	Код команды
SYS_WRITE	Запись	0
SYS_READ	Чтение	1
SYS_EXIT	Завершение программы	60

Листинг программы с комментариями и макросами

Листинг файла «*macro.nasm*».

```
; Constants:
%define BUFFER_SIZE  100 ; Size of the user input buffer.

; Macro that wraps clib scanf.
; Input: RSI - format, RDX - address of string.
%macro Cscanf 2
    lea  rdi, [%1]
    lea  rsi, [%2]
    xor  rax, rax
    call scanf
%endmacro

; Macro that wraps clib printf().
; Input: RSI - address of string, RDX - format.
%macro Cprintf 2
    lea  rdi, [%1]
    lea  rsi, [%2]
    xor  rax, rax
    call printf
%endmacro

; Macro that wraps clib strtok().
; Input: RDI - 1st arg of strktok(), RSI - 2nd arg of strtok().
; Output: Result of strtok() call in RAX.
%macro Cstrtok 2
    lea  rdi, [%1] ; Input string.
    lea  rsi, [%2] ; Delimiter.
    xor  rax, rax  ; Clear RAX before calling, because it would contain the
result.
    call strtok    ; char* token = strtok(dictionary, delimiter);
%endmacro

; Macro that wraps clib strtok() but 1st arg is always NULL.
; Input: RSI - 2nd arg of strtok().
; Output: Result of strtok() call in RAX.
%macro Cstrtok 1
    xor  rdi, rdi  ; Because the first param to strtok should be NULL == 0.
    lea  rsi, [%1] ; The second param is the same delimiter.
    xor  rax, rax  ; Clear RAX before calling, because it would contain the
result.
    call strtok    ; token = strtok(NULL, delimiters);
%endmacro
```

Листинг файла «*coursework.nasm*».

```
; 6. Develop a word game: who can come up with more words from the characters of
a given string.
;    Check for:
;      - validity based on characters;
;      - existence in the dictionary.
;    Allow for dictionary expansion and ignore case sensitivity (A = a).

%include 'macro.nasm'  ; Include external macro and constants definitions.

section .rodata
    ; Game rules string.
    rules db `Игра в слова на английском языке!\n`
          db `Правила: игроки по очереди вводят по одному слову.\n`
          db `Слово должно состоять из символов заданной строки, а также
принадлежать словарю.\n`
          db `Это означает, что нельзя писать несуществующие слова, либо же
слова, содержащие другие символы.\n`
          db `Игра оканчивается как только один из игроков введёт некорректное
слово.\n`
          db `Слова можно писать в любом регистре, высота букв
игнорируется.\n\n`, 0
    rules_len equ $ - rules ; Length of the rules string.

    source_string_prompt     db  'Исходная строка, из букв которой вам необходимо
составлять слова: ', 0 ; Prompt for source string.
    source_string_prompt_len equ $ -
source_string_prompt                                        ; Prompt for
source string length.
    source_string            db  `ehllo\n`,
0                                                   ; Source string for
word formation.
    source_string_len        equ $ -
source_string                                               ; Source
string length.

    delimiter db " ", 0 ; Delimiter is space symbol.

    incorrect_msg db `Слово не найдено в словаре! Игра окончена, вы проиграли
:)\n`, 0 ; Incorrect message string
    correct_msg   db `Слово найдено в словаре! Теперь очередь следующего
игрока!\n`, 0 ; Correct message (found) string.

    string_format db "%s", 0 ; Format string for Cprintf.

    ; hello - привет, приветствие.
    ; hell - ад, место в религиозных представлениях, предназначенное для
наказания грешников.
    ; hole - отверстие, яма, дыра.
```

```
    ; hoe - мотыга, садовый инструмент для рыхления почвы.
    ; lo - сокращение от "low" (низкий), также используется как устное выражение
в значении "привет" или "пока".
    ; ole - устаревшая форма слова "oil" (масло) или в испанском языке может быть
использовано как восклицание поддержки.
    ; ell - небольшая комната или помещение.
    dictionary     db  'hello hell hole hoe lo ole ell', 0 ; Space-separated
dictionary words.
    dictionary_len equ $ - dictionary                      ; Length of the
dictionary string.

section .bss
    user_input      resb BUFFER_SIZE ; Buffer to hold user input.
    dictionary_copy resb BUFFER_SIZE ; Copy of the dictionary for tokenization.

section .text
    global  main
    extern  scanf, printf, strtok, strcmp ; Extern C functions from C library.
    default rel  ; RIP-relative addressing for [name].

main:
    ; Create a stack-frame, realigning the stack to 16-byte alignment before
calls.
    push rbp
    mov  rbp, rsp

    ; Write rules and source string to stdout.
    Cprintf rules,                string_format
    Cprintf source_string_prompt, string_format
    Cprintf source_string,        string_format

    game_loop:
        ; Read user input.
        Cscanf string_format, user_input

        ; Copy the dictionary to a writable area for tokenization.
        lea rdi, [dictionary_copy]
        lea rsi, [dictionary]
        mov rcx, dictionary_len
        rep  movsb

        ; Tokenize the dictionary string by delimiter (space).
        ;   char* token = strtok(dictionary, delimiter);
        Cstrtok dictionary_copy, delimiter
        ; Result in RAX.

        ; Compare the user input with each token in the dictionary.
        ; while(token != NULL)
        ;     if(strcmp(token, input) == 0)
        ;         printf("FOUND!\n");
```

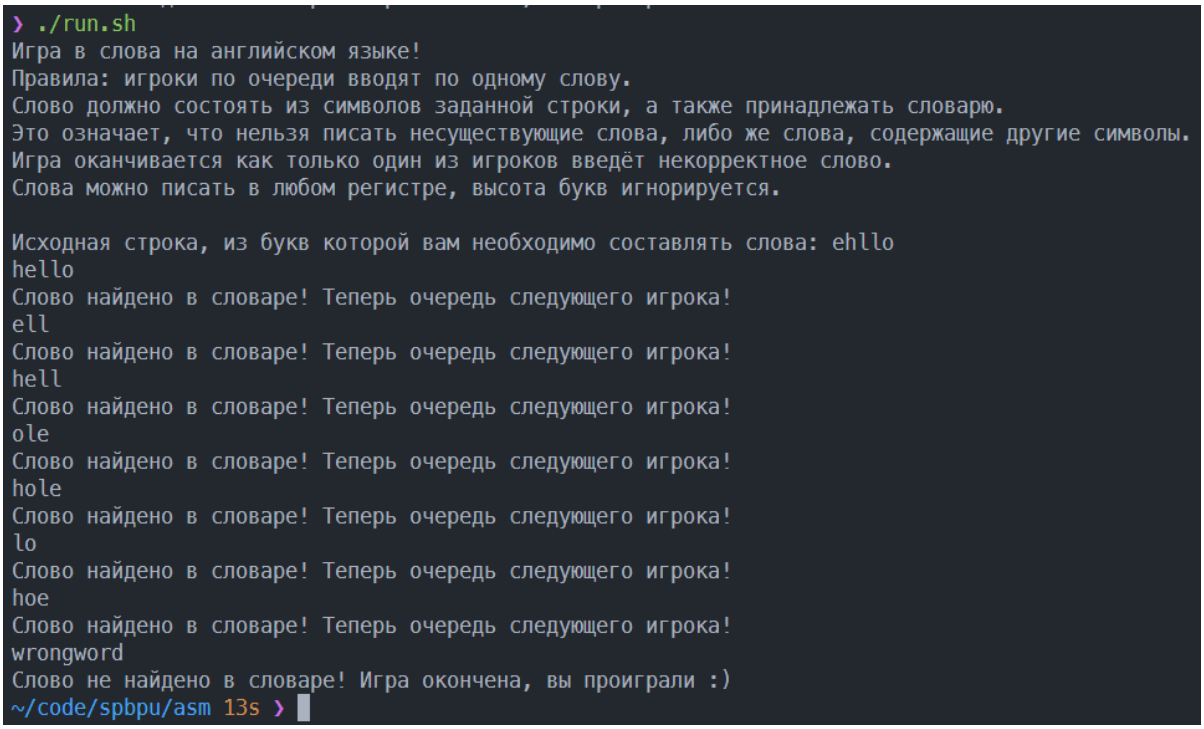
```
        ;     token = strtok(NULL, delimiters);
        ; }
        compare_loop:
            test rax, rax  ; Set ZF to 1 if RAX == 0 (token == NULL).
            jz   game_over ; Game ends if the token is NULL (end of dictionary).

            ; Now compare the current token in dictionary with user input.
            lea  rdi, [rax]
            lea  rsi, [user_input]
            call strcmp            ; strcmp(token, input)

            ; If strcmp returned 0, the word is found. Print a message and
continue the game loop.
            test rax, rax
            jz   word_found ; Word found if strcmp(token, input) == 0.

            ; If not, continue tokenizing.
            Cstrtok delimiter    ; Move to the next token = strtok(NULL,
delimiters).
            ; Result (next token, e.g., the next word from the dictionary) in
RAX.
            jmp     compare_loop

            word_found:
                Cprintf correct_msg, string_format
                jmp     game_loop                  ; Continue the game loop.

            ; Someone accidently or by purpose entered the wrong word.
            game_over:
                Cprintf incorrect_msg, string_format
                xor     eax,           eax           ; Set zero (EXIT_SUCCESS).
                leave
                ret
```

Листинг файла «*run.sh*».

```
#!/usr/bin/env bash

# Set up build folder
BUILD_FOLDER="build"

# Create build folder
mkdir -p "$BUILD_FOLDER" || {
    echo "Error: Unable to create build folder '$BUILD_FOLDER'"
    exit 1
}

# Function to check for errors
check_error() {
```

```
    local code=$?
    local error_message=$1

    if [ $code -ne 0 ]; then
        echo "Error: $error_message. Exit code = $code."
        exit $code
    fi
}

# Compile assembly
nasm -g -F dwarf -f elf64 -o "$BUILD_FOLDER/coursework.o" coursework.nasm
check_error "Compilation failed"

# Outdated, because now I use GCC for C library functions:
# Link
# ld.gold -static -s -o "$BUILD_FOLDER/coursework" "$BUILD_FOLDER/coursework.o" -
no-pie
# check_error "Linking failed"

# Use GCC
gcc -g -m64 -o "$BUILD_FOLDER/coursework" "$BUILD_FOLDER/coursework.o" -no-pie
check_error "Linking failed"

# Run the executable
EXECUTABLE_PATH="$BUILD_FOLDER/coursework"
if [ -x "$EXECUTABLE_PATH" ]; then
    "$EXECUTABLE_PATH"
    # Optional: debug
    # gdb --args "$BUILD_FOLDER/coursework"
    check_error "Execution failed"
else
    echo "Error: Executable '$EXECUTABLE_PATH' not found or not executable"
    exit 1
fi
```

Листинг (скриншоты) результатов выполнения заданий



Список использованной литературы

1. **Stack Overflow** (https://stackoverflow.com): Этот ресурс является крупнейшим сообществом разработчиков, где можно найти ответы на различные вопросы по программированию на разных языках, включая ассемблер. Вопросы и ответы на Stack Overflow часто предоставляют практические решения для сложных задач.
2. **NASM - The Netwide Assembler Documentation** (https://www.opennet.ru/docs/RUS/nasm/nasm_ru4.html): Официальная документация по Netwide Assembler (NASM) предоставляет исчерпывающий и авторитетный источник информации о синтаксисе и особенностях языка ассемблера NASM.
3. **NASM Tutorial** (https://metanit.com/assembler/nasm): Руководство по NASM на Metanit предоставляет понятные и наглядные пошаговые инструкции для освоения ассемблера с использованием NASM.
4. **NASM x64 C Calling Convention** (https://soliduscode.com/nasm-x64-c-calling-convention): Этот источник объясняет конвенции вызова для функций на ассемблере x64, используемые при взаимодействии с функциями, написанными на языке C.
5. **NASM Community Forum** (https://forum.nasm.us): Форум сообщества NASM, где разработчики могут задавать вопросы, делиться опытом и получать поддержку от опытных членов сообщества.
6. **C++ Reference** (https://en.cppreference.com/w/c/string): Справочник по функциям обработки строк в языке программирования C.
7. **draw.io** (https://app.diagrams.net): Этот инструмент для создания диаграмм является удобным ресурсом для визуализации структуры программы и логики игры, в частности, с его помощью была создана блок-схема в данном отчёте.