МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт (факультет) Институт информационных технологий
Кафедра математического и программного обеспечения ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине Программирование на ассемблере

на тему Программирование на языке низкого уровня

Выполнил студент группы 1ПИб-02-3оп-22
группа
направления подготовки (специальности)
09.03.04 Программная инженерия
(искусственный интеллект)
шифр, наименование
Фролов Никита Игоревич
фамилия, имя, отчество

Руководитель
Виноградова Людмила Николаевна
фамилия, имя, отчество
старший преподаватель
должность

Дата предоставления работы
«15» января 2024 г.

Заключение о допуске к защите

Оценка ,
количество баллов

Подпись преподавателя,

Череповец, 2024
год

Аннотация

Курсовая работа посвящена теме «Программирование на языке низкого уровня».

Выполнил: Фролов Никита Игоревич, студент группы 1ПИб-02-3оп-22 Института информационных технологий Череповецкого государственного университета.

В работе исследуются основные возможности языка ассемблера микропроцессора Intel 8086, в том числе принципы создания файлов и работы с ними на данном языке.

В работе описывается создание программы, которая заменяет в тексте входного файла строчные буквы, заключённые в скобки, на прописные, и записывает результат обработки в новый текстовый файл.

Курсовая работа состоит из 40 страниц: введения, 7 глав, заключения, списка литературных источников и приложения; содержит 4 источника, 4 таблицы и 3 приложения.

Оглавление

ВВЕДЕНИЕ

Язык ассемблера – это язык программирования низкого уровня, язык процессора в понятном для человека виде. В данной курсовой работе рассматривается язык ассемблера для микропроцессора Intel 8086, который был выпущен компанией Intel в 1978 году [4].

Всего в процессоре Intel 8086 имеется 14 16-разрядных регистров [2][4]:

– 8 регистров общего назначения (AX, BX, CX, DX) – их особенностью является то, что они допускают адресацию целых регистров, а также отдельно их младшей половины (AL, BL, CL, DL) и старшей половины (AH, BH, CH, DH). Это позволило сохранить обратную совместимость с 8-битными программами;

– 4 регистра-указателя (SI, DI, BP, SP);

– 4 сегментных регистра (CS, SS, DS, ES);

– программный счётчик или указатель команды (IP);

– регистр флагов FLAGS с 9 флагами.

Система команд процессора Intel 8086 состоит из более 100 команд, на основе комбинаций которых генерируется 3800 их вариаций [4]. Программы для компьютера состоят из команд микропроцессора, представляющих собой машинные коды. Для выполнения программы она должна быть помещена в память компьютера. Специальный регистр микропроцессора содержит всегда адрес следующей исполняемой команды. В памяти компьютера содержатся и данные, которые могут использоваться исполняемой программой [2].

ОСНОВНАЯ ЧАСТЬ

1. Изучение и описание предметной области

Для взаимодействия с текстовыми файлами на языке ассемблера используется набор функций DOS, а именно:

2) **Функция 3CH** – функция создания нового файла или перезаписи старого файла. Для выполнения функции необходимо занести:

– в регистр CX – необходимый атрибут файла: 00h – нормальный файл, 01h – только для чтения, 02h – скрытый, 04h – системный, 08h – метка диска, 10h – директория, 20h – архивный файл;

– в регистровую пару DS:DX – адрес ASCIIZ-строки [1].

Файл может иметь несколько атрибутов – их можно комбинировать с помощью операции «ИЛИ». При перезаписи существующего файла его атрибуты не изменяются.

В случае успешного выполнения функции создается новый элемент оглавления и файл с заданным атрибутом, а в регистре AX возвращается файловый описатель созданного файла. Файловым описателем может быть одно из следующих значений:

– 0 – STDIN (стандартный ввод);

– 1 – STDOUT (стандартный вывод);

– 2 – STDERR (стандартный вывод ошибок);

– 3 – AUX (последовательный порт);

– 4 – PRN (параллельный порт) [1].

Если файл уже существует, то его длина устанавливается равной 0 для перезаписи. При попытке перезаписать существующий файл с атрибутом «только чтение» функция установит флаг CF и занесет в регистр AX код ошибки 05H (доступ запрещен) [3].

3) **Функция 3DH** – функция открытия существующего файла. Обязательно вызывается перед функцией чтения файла. Эта функция проверяет правильность

имени файла и его наличие на диске. Для выполнения функции необходимо занести:

– в регистр AL код доступа: 0 – открыть файл только для чтения, 1 – открыть файл только для записи, 2 – открыть файл для чтения и записи;

– в регистровую пару DS:DX – адрес ASCIIZ-строки [1].

Если файл с указанным именем не существует, то функция 3DH вернет код ошибки 02H (файл не найден) или 03H (путь не найден) в регистре AX [3].

4) **Функция 3FH** – функция чтения данных из открытого файла. Для выполнения функции необходимо занести:

– в регистр BX – файловый номер;

– в регистр CX – число байт, которое необходимо прочитать;

– в регистровую пару DS:DX – адрес области памяти для записи прочитанной информации [1].

Перед чтением файл должен быть открыт функцией 3DH. При успешном выполнении функции в регистр AX заносится число реально прочитанных байт. Если число реально прочитанных байт меньше, чем было задано, то это означает, что был достигнут конец файла [3].

5) **Функция 3EH** – функция закрытия ранее открытого чтения или для записи файла. Если закрывается файл, открытый для записи, функция 3EH записывает оставшиеся данные из файлового буфера на диск, корректирует оглавление и таблицу FAT. Для выполнения функции необходимо занести:

– в регистр BX – файловый номер [1].

В случае ошибки в регистре AX устанавливается код 06H (неправильный файловый номер) [3].

6) **Функция 40H** – функция записи данных в открытый файл. Для выполнения функции необходимо занести:

– в регистр BX – файловый номер;

– в регистр CX – число байт, которое необходимо записать;

– в регистровую пару DS:DX – адрес области памяти с данными для записи [1].

Перед записью файл должен быть открыт для записи или создан функцией 3CH. При успешном выполнении функции в регистр AX заносится число реально записанных байт [1]. Если число реально записанных байт меньше, чем было задано, то это означает, что диск переполнен. В случае ошибки в регистре AX устанавливается код 05H (нет доступа) или 06H (неправильный файловый номер) [3].

7) **Функция 42H** – функция управления файловым указателем[1]. При открытии файла значение файлового указателя устанавливается в 0 и увеличивается при последующих операциях чтения/записи. Для выполнения функции необходимо занести:

– в регистр AL – один из кодов, которые определяют точку отсчета смещения: 0 – смещение от начала файла, 1 – смещение от текущего значения файлового указателя, которое может быть в любом месте, включая начало файла; 2 – смещение от конца файла;

– в регистр BX файловый номер;

– в регистровую пару CX:DX – требуемое значение файлового указателя [1].

В случае ошибки в регистре AX устанавливается код 01H (ошибка кода отсчета) или 06H (неверный файловый номер) [3].

Вместе с этими функциями используется INT 21H – прерывание в ОС MS-DOS, перехватывается программами для реализации их безопасного вызова. Прерывание – это временная остановка какого-либо процесса. Команда INT представляет собой 2-байтовую инструкцию, где первый байт – это код операции, а второй байт – номер типа прерывания. Команда INT реализует программные прерывания – прерывания, выполняющиеся, когда приходит время выполнить соответствующую команду. После их выполнения программа продолжает работать с команды, стоящей за командой вызова прерывания. Также существуют аппаратные прерывания, вызываемые любым периферийным устройством, например, нажатие клавиши, переход принтера в состояние

[1] Файловый указатель – это 32-битное число, определяющее текущую позицию чтения или записи для файла.

готовности, наступление некоторого события в микропроцессоре (деление на нуль, переполнение и т.п.) и т.д. [2]

2. Постановка задачи

Разрабатываемый программный продукт должен:

– получать на вход текстовый файл в кодировке ANSI для обработки;

– обрабатывать текст входного файла, заменяя строчные буквы английского и русского алфавитов, заключённые в скобки (круглые, фигурные, квадратные, угловые), на прописные;

– создавать новый файл и записывать в него результаты обработки текста исходного файла;

– выводить сообщения об ошибках или успешном выполнении программы на экран;

– работать на персональном компьютере под управлением ОС Microsoft Windows через эмулятор микропроцессора Intel 8086 – EMU8086.

3. Выбор структур данных

В таблице 1 представлены структуры данных, используемые в коде программы.

Таблица 1.

Структуры данных программы

Обозначение	Наименование	Тип данных
1	2	3
Начальное сообщение программы	START_MSG	Байт
Сообщение о попытке открытия файла input.txt программой	OPEN_MSG	Байт

Продолжение табл. 1

1	2	3
Сообщение о попытке чтения файла input.txt программой	READ_MSG	Байт
Сообщение об успешном выполнении обработки файла input.txt	DONE_MSG	Байт
Сообщение об ошибке открытия файла программой	ERR_OPEN_MSG	Байт
Сообщение об ошибке чтения файла программой	ERR_READ_MSG	Байт
Сообщение об ошибке закрытия файла программой	ERR_CLOSE_MSG	Байт
Сообщение об ошибке создания нового файла программой	ERR_CREATE_MSG	Байт
Путь к входному текстовому файлу input.txt	INPUT_F	Байт
Путь к выходному текстовому файлу output.txt, создаваемому программой	OUTPUT_F	Байт
Буфер для сохранения символов из текста входного файла	BUF	Байт
Описатель входного файла input.txt	HANDLE_I	Слово
Описатель выходного файла output.txt	HANDLE_O	Слово
Размер исходного файла input.txt	I_SIZE	Слово
Флаг «найдена открывающаяся скобка»	IS_BRACKET	Байт
Флаг «найден символ экранизации «\»	IS_ESCAPE_CHAR	Байт

4. Логическое проектирование

Программа выполняет следующий алгоритм для обработки файла:

1) Программа открывает исходный файл, либо выводит сообщение о невозможности открыть файл;

2) Программа высчитывает размер исходного файла;

3) Программа создаёт выходной файл output.txt, либо выводит сообщение о невозможности создать файл;

4) Программа читает исходный файл, либо выводит сообщение о невозможности прочитать файл;

5) Программа обрабатывает текст исходного файла посимвольно, а именно:

– проверяет, не достигнут ли конец файла;

– проверяет, не является ли прочитанный символ косой чертой «\»; если условие выполняется, то программа либо записывает в файл символ (если нужно экранировать сам символ «\»), либо поднимает флаг IS_ESCAPE_CHAR и переходит к следующему символу;

– проверяет, не является ли прочитанный символ открывающейся скобкой; если условие выполняется, то программа изменяет каждый последующую букву на прописную, пока не дойдёт до закрывающейся скобки, записывает в выходной файл обработанную букву и переходит к следующему символу;

– если не выполнены предыдущие условия, то программа записывает символ в выходной файл output.txt и переходит к следующему символу.

6) При достижении конца исходного файла программа закрывает оба файла, либо выводит сообщение о невозможности закрыть файлы;

7) Программа выводит сообщение об успешном выполнении.

5. Физическое проектирование

В таблице 2 описаны модули, представленные в коде программы.

Таблица 2.

Модули программы

Имя модуля	Заголовок процедуры или функции	Формальные параметры	Выполняемое действие
1	2	3	4
START	START	data START_MSG	Начало программы, вывод

Продолжение табл. 2

1	2	3	4
			начального сообщения
OPEN_INPUT	OPEN_INPUT	OPEN_MSG INPUT_F HANDLE_I	Открывает входной файл
GET_INPUT_SIZE	GET_INPUT_SIZE	HANDLE_I I_SIZE	Вычисляет размер входного файла
CREATE_FILE	CREATE_FILE	OUTPUT_F HANDLE_O	Создаёт выходной текстовый файл
READ_INPUT	READ_INPUT	HANDLE_I I_SIZE stack	Читает выходной файл
PROCESS_TEXT	PROCESS_TEXT	HANDLE_I SI IS_BRACKET	Обрабатывает текст входного файла
FOUND_OPEN_PAR	FOUND_OPEN_PAR	IS_BRACKET	Поднимает флаг IS_BRACKET при нахождении открывающейся скобки
FOUND_CLOSE_PAR	FOUND_CLOSE_PAR	IS_BRACKET	Опускает флаг IS_BRACKET при нахождении закрывающейся скобки
FOUND_ESCAPE	FOUND_ESCAPE	IS_ESCAPE	Проверяет флаг «найден символ экранизации «\»

Продолжение табл. 2

1	2	3	4
NEXT_CHAR	NEXT_CHAR	SI	Переход к следующему символу
LOWER_TO_UPPER	LOWER_TO_UPPER	-	Перевод строчной буквы в прописную
LOWER_TO_UPPER_YO	LOWER_TO_UPPER_YO	-	Перевод строчной ё в прописную Ё
WRITE_TO_FILE	WRITE_TO_FILE	BUF HANDLE_O	Запись в выходной файл
EOF	EOF	CLOSE_ERROR DONE_MSG	Закрытие файлов
OPEN_ERROR	OPEN_ERROR	ERR_OPEN_MSG	Вывод сообщения об ошибке открытия файла
READ_ERROR	READ_ERROR	ERR_READ_MSG	Вывод сообщения об ошибке прочтения файла
CREATE_ERROR	CREATE_ERROR	ERR_CREATE_MSG	Вывод сообщения об ошибке создания файла
CLOSE_ERROR	CLOSE_ERROR	ERR_CLOSE_MSG	Вывод сообщения об ошибке закрытия файла
FINISH	FINISH	-	Выход из программы

6. Кодирование

Программа применят для обработки текста файла алгоритм, описанный в п.5:

1) Программа выводит приветственное сообщение:

```
mov AX, data                ; загружаем переменные
mov DS, AX                  ; в сегмент DS
mov AH, 9h                  ; функция вывода строки целиком на экран
mov DX, offset START_MSG    ; DX := адрес сообщения для вывода
относительно начала сегмента
int 21h                     ; прерывание
```

2) Программа открывает исходный файл с помощью функции 3Dh, либо выводит сообщение о невозможности открыть файл:

```
OPEN_INPUT:
   mov AH, 9h                ; функция вывода на экран
   mov DX, offset OPEN_MSG   ; DX := адрес сообщения для вывода
относительно начала сегмента
   int 21h                   ; прерывание
   mov AL, 00h               ; открыть файл для чтения
   mov AH, 3Dh               ; функция открытия файла
   lea DX, INPUT_F           ; DX := адрес файла input.txt
   int 21h                   ; прерывание
   jc OPEN_ERROR             ; если CF = 1, то перейди к OPEN_ERROR
   mov HANDLE_I, AX          ; HANDLE_I := файловый указатель input.txt
OPEN_ERROR:
   mov AH, 09h                          ; функция вывода на экран
   mov DX, offset ERR_OPEN_MSG          ; DX := адрес сообщения ошибки
для вывода относительно начала сегмента
   int 21h                              ; прерывание
   jmp FINISH                           ; переход к FINISH
```

3) Программа высчитывает размер исходного файла с помощью функции 42h и записывает его в переменную I_SIZE:

```
GET_INPUT_SIZE:
   mov AH, 42h
```

```
    mov AL, 2                   ; 2 – указатель на конец файла
    mov BX, HANDLE_I            ; BX := описатель файла input.txt
    xor CX, CX                  ; CX:DX := смещение в файле в байтах
    mov DX, CX
    int 21h
    mov I_SIZE, AX
    mov AH, 42h                 ; устанавливаем указатель обратно в начало
    mov AL, 0                   ; 0 – указатель в начало файла
    mov BX, HANDLE_I
    xor CX, CX
    mov DX, CX
    int 21h
```

4) Программа создаёт выходной файл output.txt с помощью функции 3Ch, либо выводит сообщение о невозможности создать файл:

```
CREATE_FILE:
    mov AH, 3Ch             ; функция создания файла
    mov CX, 00h             ; создать файл .TXT
    lea DX, OUTPUT_F        ; DX := адрес output.txt
    int 21h
    jc CREATE_ERROR         ; если CF = 1, то перейди к CREATE_ERROR
    mov HANDLE_O, AX        ; HANDLE_O := описатель файла output.txt
    ; 3Ch закрывает файл автоматически, нужно открыть его заново!
    mov AH, 3Dh             ; функция открытия файла
    mov AL, 1               ; режим записи в файл
    lea DX, OUTPUT_F        ; DX := адрес output.txt
    int 21h
    jc OPEN_ERROR           ; если CF = 1, то перейди к OPEN_ERROR
    mov CX, I_SIZE          ; CX := размер файла input.txt
```

5) Программа читает исходный файл с помощью функции 3Fh, либо выводит сообщение о невозможности прочитать файл:

```
READ_INPUT:
    mov AH, 3Fh             ; функция чтения файла
    mov BX, HANDLE_I        ; BX := описатель файла input.txt
    mov CX, I_SIZE          ; CX := число байт для чтения (зд. размер input.txt)
    lea DX, stack           ; DX := адрес стека, куда поместить текст файла
```

```
int 21h
jc READ_ERROR          ; если IF CF = 1, то перейди к READ_ERROR
lea SI, [stack]        ; использовать адрес в stack для пересылки
соответствующего байта в регистр SI
READ_ERROR:
mov AH, 09h                          ; функция вывода на экран строки
mov DX, offset ERR_READ_MSG  ; DX := адрес сообщения ошибки для
вывода относительно начала сегмента
int 21h
jmp FINISH                           ; переход к FINISH
```

6) Программа обрабатывает текст исходного файла посимвольно, а именно:

a) программа читает один символ из файла:

```
PROCESS_TEXT:
  mov AH, 3Fh          ; функция чтения из файла
  mov BX, HANDLE_I ; BX := описатель файла input.txt
  mov CX, 1                    ; CX := число байт для чтения (здесь 1 байт)
  int 21h
  mov AH, [SI]         ; AH := символ из текста файла
```

b) программа проверяет, не достигнут ли конец файла:

```
  cmp AH, 0       ; если после прочтения функция поместила в регистр
  je EOF          ; AH := 0 – дошли до конца файла – переход к EOF
```

c) программа проверяет, не является ли прочитанный символ служебным «\» (для экранирования скобок, в случае, когда скобки и наклонная черта должны быть восприняты программой как символы текста):

```
cmp AH, '\'
je FOUND_ESCAPE      ; если символ оказался «\» – то осуществляем
переход к метке FOUND_ESCAPE
```

d) программа проверяет, не является ли прочитанный символ открывающейся скобкой и наличие символа экранизации «\»; если условие выполняется, то происходит переход к метке

FOUND_OPEN_PAR если закрывающейся – то к метке FOUND_CLOSE_PAR:

```
 ; если скобка открывающаяся – переходим к метке FOUND_OPEN_PAR
   cmp AH, '('
   je FOUND_OPEN_PAR
   cmp AH, '['
   je FOUND_OPEN_PAR
   cmp AH, '{'
   je FOUND_OPEN_PAR
   cmp AH, '<'
   je FOUND_OPEN_PAR
 ; если скобка закрывающаяся – переходим к метке FOUND_CLOSE_PAR
   cmp AH, ')'
   je FOUND_CLOSE_PAR
   cmp AH, ']'
   je FOUND_CLOSE_PAR
   cmp AH, '}'
   je FOUND_CLOSE_PAR
   cmp AH, '>'
   je FOUND_CLOSE_PAR

 FOUND_OPEN_PAR:
   cmp IS_ESCAPE_CHAR, 1 ; если флаг «найден символ экранизации «\»
поднят,
   je WRITE_TO_FILE      ; переходим к записи в файл
   mov IS_BRACKET, 1     ; если флаг «найдена скобка» поднят,
   jmp NEXT_CHAR         ; берём следующий символ

FOUND_CLOSE_PAR:
   cmp IS_ESCAPE_CHAR, 1 ; если флаг «найден символ экранизации «\»
поднят,
   je WRITE_TO_FILE      ; переходим к записи в файл
   mov IS_BRACKET, 0     ; если флаг «найдена скобка» опущен,
   jmp NEXT_CHAR         ; берём следующий символ

FOUND_ESCAPE:
   cmp IS_ESCAPE_CHAR, 1 ; если флаг символа экранизации «\» поднят,
   je WRITE_TO_FILE      ; переходим к записи в файл (случай «\\»)
```

```
    mov IS_ESCAPE_CHAR, 1 ; поднимаем флаг «найден символ
экранизации «\» (в случае, если до этого он был опущен)
    jmp NEXT_CHAR         ; берём следующий символ

NEXT_CHAR:
    inc SI                ; увеличиваем SI на 1, чтобы сдвинуть
указатель на следующий символ из текста
    jmp PROCESS_TEXT      ; продолжаем обработку
```

e) если не выполнены предыдущие условия, то программа проверяет, поднят ли флаг «найдена скобка» IS_BRACKET и, либо переходит к метке PROCESS_PAR, либо записывает символ в выходной файл после перехода к метке WRITE_TO_FILE:

```
    cmp IS_BRACKET, 1
    je PROCESS_PAR        ; если флаг поднят – переход к метке
PROCESS_PAR, обработка скобок
    jmp WRITE_TO_FILE     ; если никакие предыдущие условия не
выполнены, то символ записывается в файл
```

f) после перехода к метке PROCESS_PAR программа проверяет, является ли буква строчной или прописной и какому алфавиту принадлежит; если строчной – то заменяет её на прописную, иначе записывает её в файл:

```
    cmp AH, 'Z'                ; если код символа меньше или равен
«Z» – это прописная английская буква (A-Z, коды 65-90)
    jbe WRITE_TO_FILE          ; сразу записываем в файл
    cmp AH, 'z'                ; если код символа меньше или равен
«z» – это прописная английская буква (a-z, коды 97-122)
    jbe LOWER_TO_UPPER         ; переводим регистр
    cmp AH, 168                ; если код символа = коду «Ё» (168),
    je WRITE_TO_FILE           ; сразу записываем в файл
    cmp AH, 184                ; если код символа = коду «ё» (184),
    je LOWER_TO_UPPER_YO       ; переводим регистр отдельно
    cmp AH, 223
    jbe WRITE_TO_FILE          ; если код символа меньше или равен
«Я» – это прописная русская буква (А-Я, коды 192-223)
```

```
    jae LOWER_TO_UPPER          ; если код символа меньше или равен «я» – это строчная русская буква (а-я, коды 224-255)

LOWER_TO_UPPER:
    sub AH, 32                  ; в 1251 расстояние между прописными и строчными = 32 (кроме ё и Ё)
    mov [SI], AH                ; возвращаем обработанный символ обратно
    jmp WRITE_TO_FILE           ; записываем символ в файл

    LOWER_TO_UPPER_YO:
    sub AH, 16                  ; в 1251 расстояние между ё и Ё = 16
    mov [SI], AH                ; возвращаем обработанный символ обратно
    jmp WRITE_TO_FILE           ; записываем символ в файл
```

g) программа переходит к метке WRITE_TO_FILE и записывает в выходной файл обработанную строчную или прописную букву:

```
    WRITE_TO_FILE:
        mov BUF, AH             ; копируем символ из регистра AH в переменную BUF
        mov AX, 0               ; стираем предыдущее значение в AX
        mov AH, 40h             ; функция записи в файл
        mov BX, HANDLE_O        ; BX := описатель файла output.txt
        mov CX, 1               ; CX := сколько записать байт (здесь 1)
        lea DX, BUF             ; DX := адрес переменной BUF с символом для записи
        int 21h
        cmp CX, 1               ; если CX = 0 или больше 1, возможно, диск переполнен или ничего не было записано
        jne CREATE_ERROR        ; выводим ошибку
        mov IS_ESCAPE_CHAR, 0   ; если предыдущее условие не выполнено, то опускаем флаг «найден символ экранизации «\»
        jmp NEXT_CHAR           ; получаем следующий символ
```

h) если был осуществлён переход к метке FOUND_CLOSE_PAR, то флаг опускается и программа берёт следующий символ для обработки:

```
FOUND_CLOSE_PAR:
    cmp IS_ESCAPE_CHAR, 1 ; если флаг символа экранизации «\» поднят,
```

```
        je WRITE_TO_FILE        ; переходим к записи в файл
        mov IS_BRACKET, 0       ; если флаг «найдена скобка» опущен,
        jmp NEXT_CHAR           ; берём следующий символ
```

7) При достижении конца исходного файла программа закрывает оба файла с помощью функции 3Eh, выводит сообщение об окончании обработки файла и завершает работу, либо выводит сообщение о невозможности закрыть файлы:

```
EOF:
    mov AH, 3Eh             ; функция закрытия файла
    int 21h
    cmp AX, 00h             ; если в AX не 0 – файл не закрылся
    jne CLOSE_ERROR         ; вывод сообщения об ошибке закрытия файла
    mov AH, 3Eh
    int 21h
    jne CLOSE_ERROR

    mov AH, 9h              ; вывод сообщения на экран
    mov DX, offset DONE_MSG
    int 21h
    jmp FINISH

FINISH:
   mov AX, 4c00h            ; завершение программы
   int 21h

CLOSE_ERROR:
    mov AH, 09h
    mov DX, offset ERR_CLOSE_MSG
    int 21h
    jmp FINISH
```

7. Тестирование

Наборы тестовых данных представлены в табл. 3.

Таблица 3

Тестовые данные

№	Исходные данные	Тестируемый модуль или подпрограмма	Ожидаемый результат
1	2	3	4
1	Lorem (ipsum) dolor sit amet, (consec)tetur adipiscing elit. Phasellus com(modo) aliquam nulla, in aliquam lacus gravida sit amet. (Suspendisse) faucibus massa id metus pharetra (faucibus). Fusce at dui massa. Nullam blandit, ipsum nec molestie (aliquet), magna lectus euismod metus, quis bibendum arcu ante et (eros).	PROGRAM.ASM	Lorem IPSUM dolor sit amet, CONSECtetur adipiscing elit. Phasellus comMODO aliquam nulla, in aliquam lacus gravida sit amet. SUSPENDISSE faucibus massa id metus pharetra FAUCIBUS. Fusce at dui massa. Nullam blandit, ipsum nec molestie ALIQUET, magna lectus euismod metus, quis bibendum arcu ante et EROS.
2	— Я скажу наоборот, — начал (Сергей Иванович...) Но тут Левину опять показалось, что (они,) подойдя к самому главному, (опять) отходят, и решился предложить профессору (вопрос.)	PROGRAM.ASM	— Я скажу наоборот, — начал СЕРГЕЙ ИВАНОВИЧ... Но тут Левину опять показалось, что ОНИ, подойдя к самому главному, ОПЯТЬ отходят, и решился предложить профессору ВОПРОС.
3	— Я скажу наоборот, — начал (Сергей Иванович...) !@123%^### Lorem <ipsum) dolor sit amet, {consec)tetur adipiscing elit. Phasellus com\(modo\] aliquam nulla, in aliquam lacus gravida sit amet. <в этих скобках буквы поменяются на прописные,> \[а в этих нет, и скобки останутся\]! 1\\2, 1\\4 - это дроби	PROGRAM.ASM	— Я скажу наоборот, — начал СЕРГЕЙ ИВАНОВИЧ... !@123%^### Lorem IPSUM dolor sit amet, CONSECtetur adipiscing elit. Phasellus com(modo] aliquam nulla, in aliquam lacus gravida sit amet. В ЭТИХ СКОБКАХ БУКВЫ ПОМЕНЯЮТСЯ НА ПРОПИСНЫЕ, [а в этих нет, и скобки останутся]! 1\2, 1\4 - это дроби

В табл. 4 представлены результаты тестирования программы.

Таблица 4.

Результаты тестирования

Дата и время тестирования	Тестируемый модуль или подпрограмма	Кто проводил тестирование	Описание теста	Результаты тестирования
1	2	3	4	5
17.12.2023	PROGRAM.ASM	Фролов Н.И.	Чтение файла input.txt, набор данных №1	Программа не нашла файл для чтения.
21.12.2023	PROGRAM.ASM	Фролов Н.И.	Чтение файла input.txt, набор данных №1	Успех.
21.12.2023	PROGRAM.ASM	Фролов Н.И.	Чтение размера файла input.txt, набор данных №1	Успех.
21.12.2023	PROGRAM.ASM	Фролов Н.И.	Замена строчных латинских символов в скобках из файла input.txt на прописные, набор данных №1	Программа не заменила символы, заключённые в скобки.

Продолжение табл. 4

1	2	3	4	5
21.12.2023	PROGRAM.ASM	Фролов Н.И.	Замена строчной латиницы в скобках из файла input.txt на прописные, набор данных №1	Успех.
22.12.2023	PROGRAM.ASM	Фролов Н.И.	Замена строчной кириллицы в скобках из файла input.txt на прописные, набор данных №2	Программа не заменила символы, заключённые в скобки.
22.12.2023	PROGRAM.ASM	Фролов Н.И.	Замена строчной кириллицы в скобках из файла input.txt на прописные, набор данных №2	Программа не заменила символы, заключённые в скобки.
22.12.2023	PROGRAM.ASM	Фролов Н.И.	Замена строчной кириллицы и латиницы в скобках	Программа не заменила символы кириллицы,

Продолжение табл. 4

1	2	3	4	5
			из файла input.txt на прописные, набор данных №3	заключённые в скобки.
22.12.2023	PROGRAM.ASM	Фролов Н.И.	Замена строчной кириллицы и латиницы в скобках из файла input.txt на прописные, набор данных №3	Успех.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была разработана программа, которая работает с текстовым файлом в кодировке ANSI, и заменяет в тексте этого файла строчные буквы, заключённые в скобки, на прописные, а после записывает результат обработки в новый текстовый файл.

В ходе разработки программы были освоены базовые возможности языка ассемблер для микропроцессора Intel 8086, в том числе для работы с текстовыми файлами, и приобретены навыки работы с эмулятором EMU8086.

СПИСОК ЛИТЕРАТУРЫ

1. *Низкоуровневое программирование, INT 21h: Сервис DOS* — biosprog.narod.ru — [Электронный ресурс] — https://biosprog.narod.ru/real/dos/int21/index.htm — дата обращения: 20.12.2023;

2. *Пирогов В.Ю.* ASSEMBLER. Учебный курс. / Пирогов В.Ю. — М.: Издатель Молгачева С.В., Издательство Нолидж, 2001. — 848 с., ил.;

3. Виноградова, Л. Н. Системное программирование: учебное пособие – Череповец: ФГБОУ ВПО ЧГУ, 2016. – 210 с.

4. *Intel 8086* — ru.wikipedia.org — [Электронный ресурс] — https://ru.wikipedia.org/wiki/Intel_8086 — дата обращения: 10.12.2023.

ПРИЛОЖЕНИЕ 1

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий
наименование института (факультета)

Математическое и программное обеспечение ЭВМ
наименование кафедры

Программирование на ассемблере
наименование дисциплины в соответствии с учебным планом

УТВЕРЖДАЮ
Зав. кафедрой МПО ЭВМ,
д. т. н., профессор _____ Ершов Е. В.
«___» _____ 2023 г.

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ НИЗКОГО УРОВНЯ

Техническое задание на курсовую работу
Листов 8

Руководитель Виноградова Л.Н.
ФИО преподавателя

Исполнитель
студент 1ПИб-02-3оп-22
группа

Фролов Н.И.
Фамилия, имя, отчество

2023 год

Введение

Курсовая работа направлена на разработку программы на языке программирования низкого уровня.

1. Основания для разработки

Основанием для разработки является задание на курсовую работу по дисциплине «Программирование на ассемблере», выданное на кафедре МПО ЭВМ ИИТ ЧГУ.

Дата утверждения: 09 ноября 2023 года.

Наименование темы разработки: «Программирование на языке низкого уровня».

2. Назначение разработки

Основной задачей курсовой работы является создание программы, заменяющей в тексте исходного файла буквы слов, стоящих в скобках, прописными.

3. Требования к программе

3.1. Требования к функциональным характеристикам

Программа должна обрабатывать исходный текстовый файл с использованием функций DOS и формировать новый файл с результатами обработки исходного файла. Имя исходного и обработанного файлов задать в программе в виде ASCIIZ-строк.

При обработке файла используется следующую последовательность вызова функций DOS:

1) Открытие исходного файла (функция 3DH);

2) Создание нового файла, куда будут помещаться результаты обработки исходного файла (функция 3CH);
3) Чтение исходного файла (функция 3FH);
4) Обработка прочитанных данных;
5) Запись обработанных данных в созданный файл (функция 40H);
6) Закрытие файла с результатами обработки (функция 3EH);
7) Закрытие исходного файла (функция 3EH).

3.2. Требования к надежности

Для чтения данных из исходного файла используется буфер, определенный в программе как массив байтов или слов. Рекомендуемый размер буфера – 512 байт (256 слов).

Размер исходного файла для обработки выбрать меньшим или равным размеру буфера. Текст в файле не должен быть менее 5 строк, в строке – не менее 10 слов, язык текста – английский и русский. Файл должен быть сохранён в кодировке ANSI.

3.3. Условия эксплуатации

Требования не предъявляются.

3.4. Требования к составу и параметрам технических средств

Минимальные системные требования:

- Процессор: не менее 1 ГГц или SoC;
- ОЗУ: 1 ГБ для 32-разрядной системы или 2 ГБ для 64-разрядной системы;
- Место на жестком диске: 16 ГБ для 32-разрядной ОС или 20 ГБ для 64-разрядной ОС;
- Видеоадаптер: DirectX 9 или более поздняя версия с драйвером WDDM 1.0;

- Разрешение экрана не менее 800 x 600;
- Клавиатура, мышь.

3.5. Требования к информационной и программной совместимости

Для обеспечения совместимости требуется эмулятор EMU8086.

3.6. Требования к маркировке и упаковке

Не предъявляются.

3.7. Требования к транспортированию и хранению

Резервная копия разработки хранится на физическом носителе и в облачном хранилище для предотвращения потери файлов. Физический доступ осуществляется с помощью CD-диска и USB-накопителя, удаленный доступ – с помощью общедоступных сервисов облачного хранения.

3.8. Специальные требования

Не предъявляются.

4. Требования к программной документации

Предъявляются требования к содержанию расчетно-пояснительной записки, а также требования к оформлению документа.

4.1. Содержание расчётно-пояснительной записки

1. Титульный лист
2. Аннотация
3. Оглавление
4. Введение
5. Основная часть
   - 5.1. Изучение и описание предметной области
   - 5.2. Постановка задачи
   - 5.3. Выбор структур данных
   - 5.4. Логическое проектирование
   - 5.5. Физическое проектирование
   - 5.6. Кодирование
   - 5.7. Тестирование
6. Заключение
7. Список литературы
8. Приложения
   - 8.1. Техническое задание
   - 8.2. Руководство пользователя
   - 8.3. Текст программы

4.2. Требования к оформлению

Требования определяются государственным стандартом ГОСТ (табл. П1.1).

Таблица П1.1

Требования к оформлению

Документ	Печать на отдельных листах формата А4 (210х297 мм); оборотная сторона не заполняется; листы нумеруются. Печать возможна ч/б. Файлы предъявляются на компакт-диске: РПЗ с ТЗ; программный код. Листы и диск в конверте вложены в пластиковую папку скоросшивателя.
Страница	Ориентация – книжная; отдельные страницы, при необходимости, альбомная. Поля: верхнее, нижнее – по 2 см, левое – 3 см, правое – 1 см.
Абзацы	Межстрочный интервал – 1,5, перед и после абзаца – 0.
Шрифты	Кегль – 14. В таблицах шрифт 12. Шрифт листинга – 10 (возможно в 2 колонки).
Рисунки	Подписывается под ним по центру: Рис.Х. Название В приложениях: Рис.П1.3. Название
Таблицы	Подписывается: над таблицей, выравнивание по правому: «Таблица Х». В следующей строке по центру Название Надписи в «шапке» (имена столбцов, полей) – по центру. В теле таблицы (записи) текстовые значения – выровнены по левому краю, числа, даты – по правому.

5. Технико-экономические показатели

Требования не предъявляются.

6. Стадии и этапы разработки

Стадии и этапы разработки представлены в табл. П1.2.

Таблица П1.2

Стадии и этапы разработки

Наименование этапа разработки	Сроки разработки	Результат выполнения	Отметка о выполнении
1	2	3	4
Получение задания	09.11.2023	Утверждена тема работы	
Постановка задачи	15.11.2023	Определены условия для выполнения задания	
Оформление технического задания	15.11.2023	Оформлено техническое задание	
Изучение и описание предметной области	17.11.2023	Получены необходимые теоретические знания для разработки программы	
Выбор структур данных	21.11.2023	Выбраны структуры данных для программы	
Логическое проектирование	27.11.2023	Разработан алгоритм, решающий поставленную задачу	
Физическое проектирование	27.11.2023	Разработана модульная структура программы	
Кодирование	10.12.2023	Создана программа на языке низкого уровня	
Тестирование	20.12.2023	Конечный вариант программы проверен на работоспособность	

Продолжение табл. П1.2

1	2	3	4
Оформление сопроводительной документации	13.01.2024	Оформлена сопроводительная документация	

7. Порядок контроля и приемки

Порядок контроля и приема представлены в табл. П1.3.

Таблица П1.3

Порядок контроля и приема

Наименование контрольного этапа выполнения курсовой работы	Сроки контроля	Результат выполнения	Отметка о приемке результата контрольного этапа
Выбор темы	09.11.2023	Утверждена тема	
Сдача технического задания	16.11.2023	Согласовано техническое задание	
Сдача расчетно-пояснительной записки	15.01.2024	Согласована расчетно-пояснительная записка	
Сдача курсовой работы	15.01.2024	Получена оценка за выполненную работу	

ПРИЛОЖЕНИЕ 2

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

1. Общие сведения о программе

Данная программа предназначена для замены строчных букв латиницы и кириллицы в текстовом файле, заключённых в скобки. Программа работает с файлами в кодировке ANSI. Результат обработки программа сохраняет в новый текстовый файл с именем «output.txt».

2. Описание установки

Программный продукт требует установки дополнительного ПО, а именно эмулятора микропроцессора Intel 8086 под названием EMU8086. Файл «PROGRAM.ASM» копируется с носителя в папку «C:\EMU8086\MyBuild».

3. Описание запуска в эмуляторе EMU8086 версии 4.08 и ниже

Шаг 1. Запустите эмулятор EMU8086. В появившемся окне приветствия нажмите кнопку «Recent files» (рис. П2.1) → «Other files» из выпадающего списка. В появившемся окне откройте папку «C:\EMU8086\MyBuild» и выберите файл программы «PROGRAM.ASM».

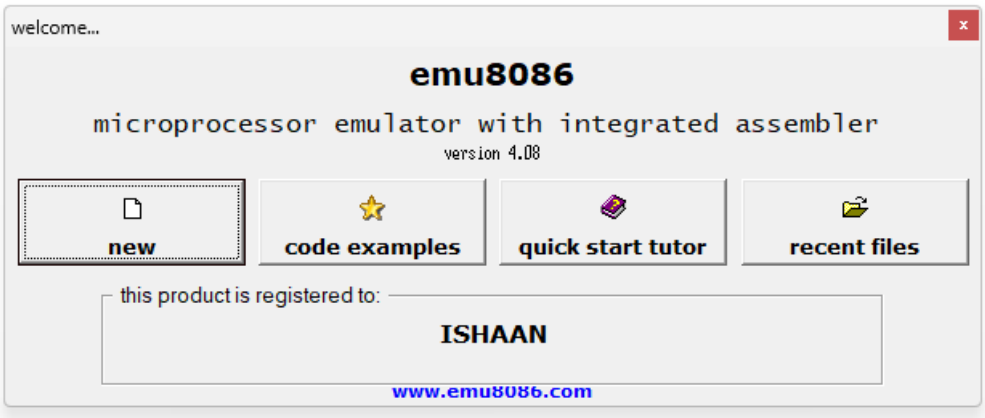

Рис. П2.1. Окно приветствия EMU8086, кнопка «Recent files» - крайняя справа

Шаг 2. Далее откроется окно редактора, на верхней панели которого расположена кнопка «Emulate» (рис. П2.2).

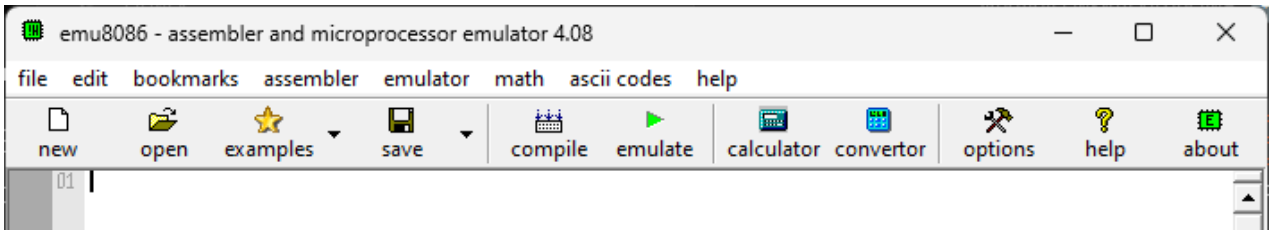

Рис. П2.2. Верхняя панель окна редактора кода EMU8086, кнопка «Emulate» находится по середине

Шаг 3. После нажатия кнопки «Emulate» появятся два новых окна – окно эмулятора и окно с исходным кодом программы. В окне эмулятора отображаются регистры и находятся кнопки управления программой. Нажмите кнопку «Run» для запуска (рис. П2.3).

Рис. П2.3. Панель окна эмулятора EMU8086, кнопка «Run» - крайняя справа до ползунка

Шаг 4. Результат обработки файла будет сохранён в новый текстовый файл с именем «output.txt» в той же папке.

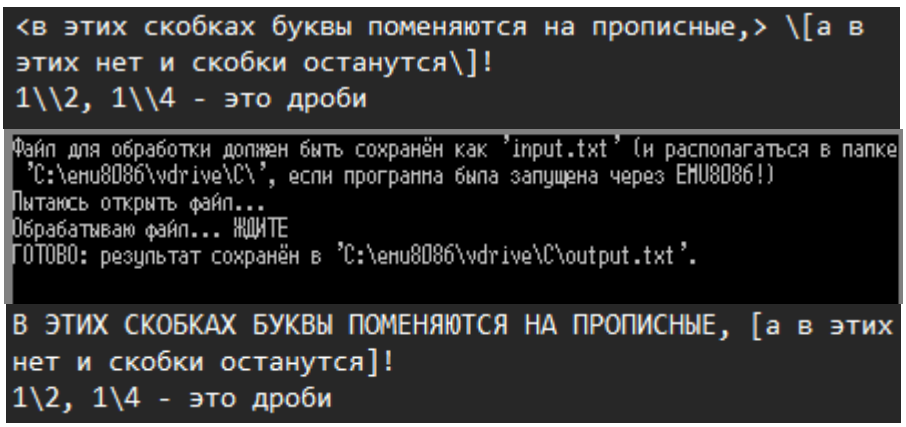

Рис. П2. 4. Пример работы программы (сверху вниз): текст входного файла, экран эмулятора, текст выходного файла

4. Инструкции по работе

– Перед запуском программы файл для обработки необходимо скопировать в папку «C:\EMU8086\vdrive\C\» и переименовать в «input.txt».

– Для правильной работы файл должен быть в кодировке ANSI:

1.1) откройте «Блокнот», нажмите на кнопку «Файл» на верхней панели приложения и выберите команду «Сохранить как» (рис. П2.4);

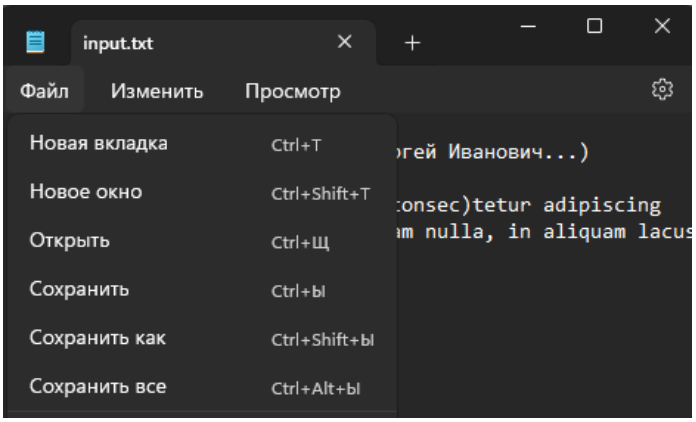

Рис. П2.5. Открытое меню «Файл» в Блокноте Windows 11

2.1) в появившемся окне найдите поле «Кодировка» и выберите «ANSI» из выпадающего списка, кликнув на поле (рис. П2.5);

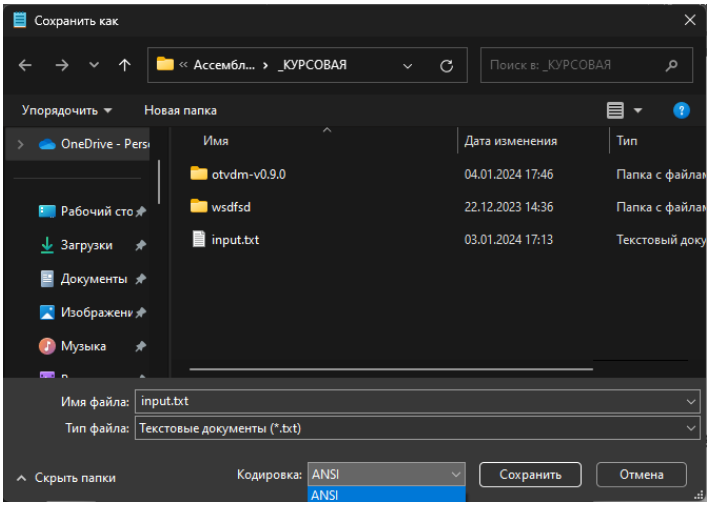

Рис. П2.6. Окно «Сохранить как» в Windows 11,
выбор кодировки расположен в нижней части окна

3.1) откройте нужную папку и сохраните файл.

– Если в тексте имеются скобки, которые должны быть проигнорированы программой, то перед ними следует поставить знак «\». Если в тексте также имеется данный символ, то перед ним следует также поставить знак «\».

– Все сообщения программы выводятся на экран эмулятора. В случае ошибки эмулятор прерывает работу программы. Чтобы вывести ошибку на экран, нужно снова нажать кнопку «Run»: программа продолжит работу, сообщение появится на экране эмулятора на русском языке.

ПРИЛОЖЕНИЕ 3

ТЕКСТ ПРОГРАММЫ

```
.8086
code SEGMENT
    ASSUME CS:code, DS:data, SS:stack
    START:
    mov AX, data
    mov DS, AX
    mov AH, 9h
    mov DX, offset START_MSG
    int 21h

   ; Открытие файла:
    OPEN_INPUT:
    mov AH, 9h
    mov DX, offset OPEN_MSG
    int 21h

    mov AL, 00h ; в режим чтения
    mov AH, 3Dh ; функция открытия файла
    lea DX, INPUT_F ; DX := адрес INPUTF
    int 21h ; execute
    jc OPEN_ERROR ; ошибка если CF == 1
    mov HANDLE_I, AX


   ; Получение размера файла:
    GET_INPUT_SIZE:
    mov AH, 42h
    mov AL, 2 ; режим 2 – указатель на конец файла
    mov BX, HANDLE_I
    xor CX, CX
    mov DX, CX
    int 21h
    mov I_SIZE, AX

    mov AH, 42h ; возвращаем указатель
    mov AL, 0      ; 0 – на начало файла
    mov BX, HANDLE_I
    xor CX, CX
    mov DX, CX
    int 21h

   ; Создание выходного файла:
    CREATE_FILE:
    mov AH, 3Ch ; создание файла
    mov CX, 00h  ; режим - созадние TXT файла
    lea DX, OUTPUT_F
    int 21h
    jc CREATE_ERROR
    mov HANDLE_O, AX

   ; 3Ch закрывает файл, для записи его нужно открыть заново
    mov AH, 3Dh
    mov AL, 1
    lea DX, OUTPUT_F
    int 21h
    jc OPEN_ERROR
    mov CX, I_SIZE


    ; Читаем исходный файл
    mov AH, 9h
    mov DX, offset READ_MSG
    int 21h

    READ_INPUT:
    mov AH, 3Fh            ; фунуция чтения
    mov BX, HANDLE_I ; копируем описатель файла в BX
    mov CX, I_SIZE ; CX := число байт для чтения
    lea DX, stack ; DX := адрес стека, куда сохраним текст файла
    int 21h                ; выполнить
    jc READ_ERROR ;  ошибка IF CF == 1

    lea SI, [stack]
    ; Обработка:
    PROCESS_TEXT:
    mov AH, 3Fh
    mov BX, HANDLE_I
    mov CX, 1
    int 21h

  ; Берём символ
    mov AH, [SI]
    cmp AH, 0
    je EOF          ; если достигли конца файла
    cmp AH, '\'
    je FOUND_ESCAPE
    cmp AH, '('
    je FOUND_OPEN_PAR ; если символ – открывающаяся скобка, переходим в этой метке
    cmp AH, '['
    je FOUND_OPEN_PAR
    cmp AH, '{'
    je FOUND_OPEN_PAR
    cmp AH, '<'
    je FOUND_OPEN_PAR

    cmp AH, ')'
    je FOUND_CLOSE_PAR ; если символ – закрывающаяся скобка, переходим в этой метке
    cmp AH, ']'
    je FOUND_CLOSE_PAR
    cmp AH, '}'
    je FOUND_CLOSE_PAR
```

```
102     cmp AH, '>'
103     je FOUND_CLOSE_PAR
104
105     cmp IS_BRACKET, 1
106     je PROCESS_PAR    ; если флаг поднят,
    то обрабатываем символ
107
108     jmp WRITE_TO_FILE ; иначе
    записываем его в файл
109
110     PROCESS_PAR:
111     cmp AH, 'Z'
112     jbe WRITE_TO_FILE ; если AH <= кода
    символа Z, то записываем в файл, буква
    уже прописная
113     cmp AH, 'z'
114     jbe LOWER_TO_UPPER ; IF AH > кода Z
    && AH <= кода z, то это английская
    строчная, нужно заменить
115     cmp AH, 168; отдельно проверяем букву
    Ё прописную
116     je WRITE_TO_FILE
117     cmp AH, 184; если AH = ё, то меняем
    отдельно, вычитая 16
118     je LOWER_TO_UPPER_YO
119     cmp AH, 223
120     jbe WRITE_TO_FILE ; IF AX <= кода
    символа Я, то записываем в файл, буква
    уже прописная
121     jae LOWER_TO_UPPER ; если AX < Я,
    то это русская строчная, нужно заменить
122     LOWER_TO_UPPER:
123   ; для a-z, а-я
124     sub AH, 32; расстояние - 32
125     mov [SI], AH
126     jmp WRITE_TO_FILE ; запись в файл
127
128     LOWER_TO_UPPER_YO:
129 ; для ё
130     sub AH, 16 ; в ansi расстояние между ё и
    Ё - 16
131     mov [SI], AH
132     jmp WRITE_TO_FILE ; записываем
    символ в файл
133
134   ; дошли до открывающейся скобки
135     FOUND_OPEN_PAR:
136     cmp IS_ESCAPE_CHAR, 1 ; прошлый
    символ был “\” ?
137     je WRITE_TO_FILE ; тогда пишем
    текущий в файл
138
139     mov IS_BRACKET, 1
140     jmp NEXT_CHAR
141
142   ; дошли до закрывающейся скобки
143     FOUND_CLOSE_PAR:
144     cmp IS_ESCAPE_CHAR, 1
145     je WRITE_TO_FILE
146
147     mov IS_BRACKET, 0
148     jmp NEXT_CHAR
149     ; встретился “\”
150     FOUND_ESCAPE:
151     cmp IS_ESCAPE_CHAR, 1 ; если
    предыдущий символ тоже  “\”
152     je WRITE_TO_FILE   ; то пиши “\” в
    файл
153 mov IS_ESCAPE_CHAR, 1   ; иначе
    поднимай флаг, встречен единичный «\»
154   jmp NEXT_CHAR ; перейди к
    следующему символу
155
156     NEXT_CHAR:
157     inc SI ; получаем следующий символ
158     jmp PROCESS_TEXT
159
160     WRITE_TO_FILE:
161     mov BUF, AH
162     mov AX, 0
163     mov AH, 40h
164     mov BX, HANDLE_O
165     mov CX, 1
166     lea DX, BUF
167     int 21h
168
169     cmp CX, 1
170     jne CREATE_ERROR
171
172     mov IS_ESCAPE_CHAR, 0
173
174     jmp NEXT_CHAR  ; переход к след
    символу
175
176     ; достигнут конец файла, закрываем оба
    файла
177     EOF:
178     mov AH, 3Eh
179     int 21h
180     cmp AX, 00h
181     jne CLOSE_ERROR
182     mov AH, 3Eh
183     int 21h
184     jne CLOSE_ERROR
185     mov AH, 9h
186     mov DX, offset DONE_MSG
187     int 21h
188     jmp FINISH
189
190 ;закрытие программы
191     FINISH:
192     mov AX, 4c00h
193     int 21h
194
195 ; сообщение об ошибке открытия файла
196     OPEN_ERROR:
197     mov AH, 09h
198     mov DX, offset ERR_OPEN_MSG
199     int 21h
200     jmp FINISH
201
202 ; сообщение об ошибке чтения файла
203     READ_ERROR:
204     mov AH, 09h
```

```
    mov DX, offset ERR_READ_MSG
    int 21h
    jmp FINISH

; сообщение об ошибке закрытия файла
    CLOSE_ERROR:
    mov AH, 09h
    mov DX, offset ERR_CLOSE_MSG
    int 21h
    jmp FINISH

; сообщение об ошибке создания файла
    CREATE_ERROR:
    mov AH, 09h
    mov DX, offset ERR_CREATE_MSG
    int 21h
    jmp FINISH

; ПЕРЕМЕННЫЕ
data SEGMENT
; «Файл для обработки должен быть сохранён как 'input.txt' (и располагаться в папке 'C:\emu8086\vdrive\C\', если программа была запущена через EMU8086!)»
START_MSG db 0094h, 00A0h, 00A9h, 00ABh, 020h, 00A4h, 00ABh, 00EFh, 020h, 00AEh, 00A1h, 00E0h, 00A0h, 00A1h, 00AEh, 00E2h, 00AAh, 00A8h, 020h, 00A4h, 00AEh, 00ABh, 00A6h, 00A5h, 00ADh, 020h, 00A1h, 00EBh, 00E2h, 00ECh, 020h, 00E1h, 00AEh, 00E5h, 00E0h, 00A0h, 00ADh, 00F1h, 00ADh, 020h, 00AAh, 00A0h, 00AAh, 020h, 027h, 069h, 06Eh, 070h, 075h, 074h, 02Eh, 074h, 078h, 074h, 027h, 020h, 028h, 00A8h, 020h, 00E0h, 00A0h, 00E1h, 00AFh, 00AEh, 00ABh, 00A0h, 00A3h, 00A0h, 00E2h, 00ECh, 00E1h, 00EFh, 020h, 00A2h, 020h, 00AFh, 00A0h, 00AFh, 00AAh, 00A5h, 020h, 027h, 043h, 03Ah, 05Ch, 065h, 06Dh, 075h, 038h, 030h, 038h, 036h, 05Ch, 076h, 064h, 072h, 069h, 076h, 065h, 05Ch, 043h, 05Ch, 027h, 02Ch, 020h, 00A5h, 00E1h, 00ABh, 00A8h, 020h, 00AFh, 00E0h, 00AEh, 00A3h, 00E0h, 00A0h, 00ACh, 00ACh, 00A0h, 020h, 00A1h, 00EBh, 00ABh, 00A0h, 020h, 00A7h, 00A0h, 00AFh, 00E3h, 00E9h, 00A5h, 00ADh, 00A0h, 020h, 00E7h, 00A5h, 00E0h, 00A5h, 00A7h, 0020h, 0045h, 004Dh, 0055h, 0038h, 0030h, 0038h, 0036h, 0021h, 0029h, "$"
; «Пытаюсь открыть файл...»
OPEN_MSG db 10, 13, 008Fh, 00EBh, 00E2h, 00A0h, 00EEh, 00E1h, 00ECh, 0020h, 00AEh, 00E2h, 00AAh, 00E0h, 00EBh, 00E2h, 00ECh, 0020h, 00E4h, 00A0h, 00A9h, 00ABh, 002Eh, 002Eh, 002Eh, "$"
; «Обрабатываю файл... ЖДИТЕ»
READ_MSG db 10, 13, 008Eh, 00A1h, 00E0h, 00A0h, 00A1h, 00A0h, 00E2h, 00EBh, 00A2h, 00A0h, 00EEh, 0020h, 00E4h, 00A0h, 00A9h, 00ABh, 002Eh, 002Eh, 002Eh, 0020h, 0086h, 0084h, 0088h, 0092h, 0085h, "$"
; «ГОТОВО: результат сохранён в 'C:\emu8086\vdrive\C\output.txt'.»
DONE_MSG db 10, 13, 0083h, 008Eh, 0092h, 008Eh, 0082h, 008Eh, 003Ah, 0020h, 00E0h, 00A5h, 00A7h, 00E3h, 00ABh, 00ECh, 00E2h, 00A0h, 00E2h, 0020h, 00E1h, 00AEh, 00E5h, 00E0h, 00A0h, 00ADh, 00F1h, 00ADh, 0020h, 00A2h, " 'C:\emu8086\vdrive\C\output.txt'.$"
; «ОШИБКА: не удалось открыть файл 'input.txt'!!»
ERR_OPEN_MSG db 10, 13, 008Eh, 0098h, 0088h, 0081h, 008Ah, 0080h, 003Ah, 0020h, 00ADh, 00A5h, 0020h, 00E3h, 00A4h, 00A0h, 00ABh, 00AEh, 00E1h, 00ECh, 0020h, 00AEh, 00E2h, 00AAh, 00E0h, 00EBh, 00E2h, 00ECh, 0020h, 00E4h, 00A0h, 00A9h, 00ABh, " 'input.txt'!!$"
; «ОШИБКА: не удалось прочитать файл 'input.txt'!!»
ERR_READ_MSG db 10, 13, 008Eh, 0098h, 0088h, 0081h, 008Ah, 0080h, 003Ah, 0020h, 00ADh, 00A5h, 0020h, 00E3h, 00A4h, 00A0h, 00ABh, 00AEh, 00E1h, 00ECh, 0020h, 00AFh, 00E0h, 00AEh, 00E7h, 00A8h, 00E2h, 00A0h, 00E2h, 00ECh, 0020h, 00E4h, 00A0h, 00A9h, 00ABh, " 'input.txt'!!$"
; «ОШИБКА: не удалось закрыть файл!!»
ERR_CLOSE_MSG db 10, 13, 008Eh, 0098h, 0088h, 0081h, 008Ah, 0080h, 003Ah, 0020h, 00ADh, 00A5h, 0020h, 00E3h, 00A4h, 00A0h, 00ABh, 00AEh, 00E1h, 00ECh, 0020h, 00A7h, 00A0h, 00AAh, 00E0h, 00EBh, 00E2h, 00ECh, 0020h, 00E4h, 00A0h, 00A9h, 00ABh, " !!$"
; «ОШИБКА: не удалось создать файл 'input.txt'!!»
ERR_CREATE_MSG db 10, 13, 008Eh, 0098h, 0088h, 0081h, 008Ah, 0080h, 003Ah, 0020h, 00ADh, 00A5h, 0020h, 00E3h, 00A4h, 00A0h, 00ABh, 00AEh, 00E1h, 00ECh, 0020h, 00E1h, 00AEh, 00A7h, 00A4h, 00A0h, 00E2h, 00ECh, 0020h, 00E4h, 00A0h, 00A9h, 00ABh, " 'input.txt'!!$"
INPUT_F db 'C:\input.txt', 0
OUTPUT_F db 'C:\output.txt', 0
BUF db ?
HANDLE_I dw ?
HANDLE_O dw ?
I_SIZE dw ?
IS_BRACKET db 0
IS_ESCAPE_CHAR db 0
data ENDS
stack SEGMENT
db 512 DUP(?)
stack ENDS
END START
```