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
направление подготовки (специальности)
09.03.04, Программная инженерия
шифр, наименование
Беляков Артемий Александрович
фамилия, имя, отчество

Руководитель
Виноградова Людмила Николаевна
фамилия, имя, отчество
доцент, кандидат технических наук
должность

Дата представления работы
«__26__»_____декабря_____2023 г.

Заключение о допуске к защите

Оценка _____, _____
количество баллов
Подпись преподавателя_____

Череповец, 2023
Год

Аннотация

Данную курсовую работу по дисциплине «Программирование на ассемблере» на тему «Программирование на языке низкого уровня» выполнил студент группы 1ПИб-02-3оп-22 Института информационных технологий Череповецкого государственного университета Беляков Артемий Александрович.

Целью курсовой работы является знакомство с принципами работы с языками низкого уровня, а именно с языком программирования Turbo Assembler для микропроцессора Intel 8086.

Курсовая работа содержит описание разработки программы, обрабатывающей массив типа структуры, в котором содержится информация о 15 клавиатурах. Программа подсчитывает количество клавиатур розового цвета с весом не более 400г. и записывает это число в регистр SP.

Оглавление

Введение

Программы играют огромную роль в современном мире. Существует большое количество различных видов программ: драйвера, утилиты, плееры, браузеры, различные редакторы (текстовые, графические, музыкальные видеоредакторы), мессенджеры, архиваторы, антивирусы и многие другие.

Благодаря программам мы можем смотреть, обрабатывать или создавать фотографии, видео, музыку, работать с документами, обрабатывать данные и многое другое.

Программы создаются благодаря языкам программирования – формальным знаковым системам, предназначенным для описания алгоритмов обработки данных. Каждый язык программирования имеет свой алфавит и синтаксис [3].

Языки программирования разделяются на языки высокого уровня и языки низкого уровня. Уровень языка обозначает его удаленность от машинного кода целевой архитектуры процессора. Низкий уровень означает меньший масштаб преобразований, которые должен претерпеть код программы перед тем, как он может быть запущен. Соответственно, в высоких языках программирования В данной курсовой работе будет рассматриваться язык программирования низкого уровня [3].

Одним из наиболее популярных языков программирования низкого уровня является Assembler (Ассемблер). Ассемблер представляет собой промежуточное звено между машинным кодом и языками программирования высокого уровня. Этот язык используется для написания программного обеспечения, которое управляет компьютером на более низком уровне, непосредственно взаимодействуя с аппаратным обеспечением [4].

Программы на языке ассемблера представляют собой набор инструкций, каждая из которых соответствует определенной команде процессора. Затем эти инструкции транслируются, или, по-другому, ассемблируются в машинный код – набор двоичных чисел, которые распознает центральный процессор и

выполняет соответствующие введенным инструкциям операции.

При этом, для каждого процессора существует свой собственный язык ассемблера. В данной курсовой работе будет рассматриваться язык ассемблера для процессоров, созданных на базе архитектуры микропроцессора Intel 8086, выпущенного компанией Intel в 1978 году. В процессорах данной фирмы реализована преемственность, поэтому программы, написанные для младшей модели, могут быть без изменений запущены на более старшей.

Целью курсовой работы является знакомство с принципами работы с языками низкого уровня, а именно с языком программирования Assembler для микропроцессора Intel 8086.

В рамках курсовой работы необходимо написать программу на языке Ассемблер, которая обрабатывает массив типа структуры с информацией о блюдах и подсчитывает количество блюд без мяса.

Для создания программы использовались транслятор Turbo Assembler, компоновщик Turbo Linker и эмулятор операционной системы MS-DOS DOSBox.

1. Изучение и описание предметной области

Для реализации разрабатываемой программы необходимо использовать массивы и структуры данных.

Массив – это структура данных, которая хранит набор значений какого-либо одного типа (элементов массива), идентифицируемых по одному или нескольким индексам. В зависимости от количества индексов определяется размерность массива: одномерные – с одним индексом, двумерные – с двумя и т. д. В разрабатываемой программе необходимо использовать одномерный массив [2].

Массивы в ассемблере описываются по директивам определения данных. Допускается использование конструкции повторения DUP:

```
A DB 1, 2, 3          ; определение массива A типа BYTE с начальными значениями 1, 2, 3
B DW 10 DUP (5)    ; определение массива B типа WORD из 10 элементов с повторяющимся начальным значением 5
```

Чтобы обратиться к элементу массива, нужно указать адрес начала массива и смещение элемента в массиве, например, A[2] или B[BX]. Смещение первого элемента массива равно 0. Смещения остальных элементов массива зависят от размера элементов.

Структура – это составной тип данных, занимающий несколько соседних ячеек памяти. Структура состоит из нескольких различных компонентов, называемых полями структуры. При этом, в отличие от элементов массива, поля могут быть разного типа (размера) [2].

Тип структуры описывается следующим образом:

```
TIME STRUC       ; TIME – имя типа, STRUC – директива для описания структуры
HOUR DB 12       ; определение поля HOUR типа BYTE с начальным значением 12
MIN DB 40        ; определение поля MIN типа BYTE с начальным значением 40
SEC DB 37        ; определение поля SEC типа BYTE с начальным значением 37
```

```
TIME ENDS        ; окончание описания типа структуры
```

После описания типа структуры можно описывать переменные данного типа. Эти переменные называются структурами. Описываются они следующим образом:

```
T TIME <20, 39, 47>      ; определение переменной T типа структуры TIME со значениями по умолчанию <20, 39, 47>
```

Чтобы обратиться к полю структуры, нужно указать название переменной и название поля, например, T.HOUR.

В ассемблере удобнее одной директивой описывать сразу несколько структур с помощью массива, элементами которого являются структуры. Для этого в директиве указывается несколько операндов и/или конструкция повторения DUP, например:

```
T TIME <13, 27, 41>, <15, 18, 12>, <17, 30, 25> ; определение массива T, элементами которого являются переменные типа структуры TIME со значениями по умолчанию <13, 27, 41>, <15, 18, 12>, <17, 30, 25>

T TIME <19, 25, 12>, 5 DUP (<>) ; определение массива T, элементами которого являются переменные типа структуры TIME, первая из которых имеет значение по умолчанию <19, 25, 12>, а остальные 5 не определены
```

2. Постановка задачи

Для реализации программы необходимо написать код на языке ассемблера. Он должен содержать структуру Klava, содержащую информацию о клавиатуре и состоящую из полей:

- keyboard_type – тип клавиатуры (число в пределах от 0 до 100, тип WORD);
- key_count – количество клавиш (число в пределах от 68 до 104, тип WORD);
- weight – вес клавиатуры (число типа WORD);
- color – цвет клавиатуры (число типа WORD, в значениях таблицы ANSI).

Программа должна содержать информацию о 15 блюдах. Их необходимо записать в массив Kbs типа структуры Klava.

Программа перебирает элементы массива и подсчитывает количество блюд без мяса, т. е. количество структур, в которых поле weight имеет значение не более 400, а значение цвета равно 33. Полученный результат должен быть записан в регистр SP.

3. Выбор структур данных

Структуры данных, которые были использованы в программе, представлены в табл. 1.

Таблица 1

Структуры данных

Наименование	Обозначение	Тип данных
Структура	Klava	STRUC
Поле структуры	keyboard_type	WORD
Поле структуры	key_count	WORD
Поле структуры	weight	WORD
Поле структуры	color	WORD
Массив	Kbs	Klava

4. Логическое программирование

Разрабатываемая программа выполняет следующий алгоритм:

1. В сегменте DATA определяется структура Klava и массив Kbs;
2. Сегмент DATA записывается в регистр AX;
3. Значение регистра AX записывается в регистр DS;
4. Значения регистров AX, BX, SP и DI обнуляются с помощью директивы XOR (исключающее ИЛИ);
5. В регистр CX записывается значение 15 (количество клавиатур). В дальнейшем этот регистр будет играть роль счетчика в цикле;
6. Значение значения регистра CX сравнивается с нулем с помощью команды сравнения CMP;
7. Если значение регистра CX больше нуля, с помощью оператора перехода JA (больше) процессор передает управление в метку iterate.
8. Метка iterate:
   - 8.1. Значение CX уменьшается на 1;
   - 8.2. Значение BX увеличивается на размер структуры с помощью сочетания команд ADD и TYPE Klava;
   - 8.3. Обнуляется значение регистра DX с помощью директивы XOR (исключающее ИЛИ);
   - 8.4. В регистр DX записывается значение поля weight структуры, которая является элементом массива Kbs с номером – значением регистра BX;
   - 8.5. Значение регистра DX сравнивается с 400 с помощью команды сравнения CMP;
   - 8.6. Если значение регистра DX меньше или равно 400, с помощью оператора перехода JLE (меньше) процессор передает управление в метку check_color. Иначе – продолжается выполнение метки iterate;
   - 8.7. Значение значения регистра CX сравнивается с нулем с помощью команды сравнения CMP;

8.8. Если значение регистра CX больше нуля, с помощью оператора перехода JA (больше) процессор передает управление в метку iterate. Иначе – безусловный переход (с помощью директивы JMP) в метку stop.

9. Метка check_color:

9.1. В регистр DX записывается значение поля color структуры, которая является элементом массива Kbs с номером – значением регистра BX;

9.2. Значение регистра DX сравнивается с числом 33 (ANSI значение для розового цвета);

9.3. Если значение регистра DX равно 33, с помощью оператора перехода JE (равно) процессор передает управление в метку increase ;

9.4. Значение значения регистра CX сравнивается с нулем с помощью команды сравнения CMP;

9.5. Если значение регистра CX больше нуля, с помощью оператора перехода JA (больше) процессор передает управление в метку iterate. Иначе – безусловный переход (с помощью директивы JMP) в метку stop.

9.6. Иначе – безусловный переход (с помощью директивы JMP) в метку stop.

10. Метка increase:

10.1. Значение регистра SP увеличивается на 1

10.2. Значение значения регистра CX сравнивается с нулем с помощью команды сравнения CMP;

10.3. Если значение регистра CX больше нуля, с помощью оператора перехода JA (больше) процессор передает управление в метку iterate. Иначе – безусловный переход (с помощью директивы JMP) в метку stop.

11. Метка stop: вызывается прерывание 21h

Блок-схема данной программы представлена на рис. 1.

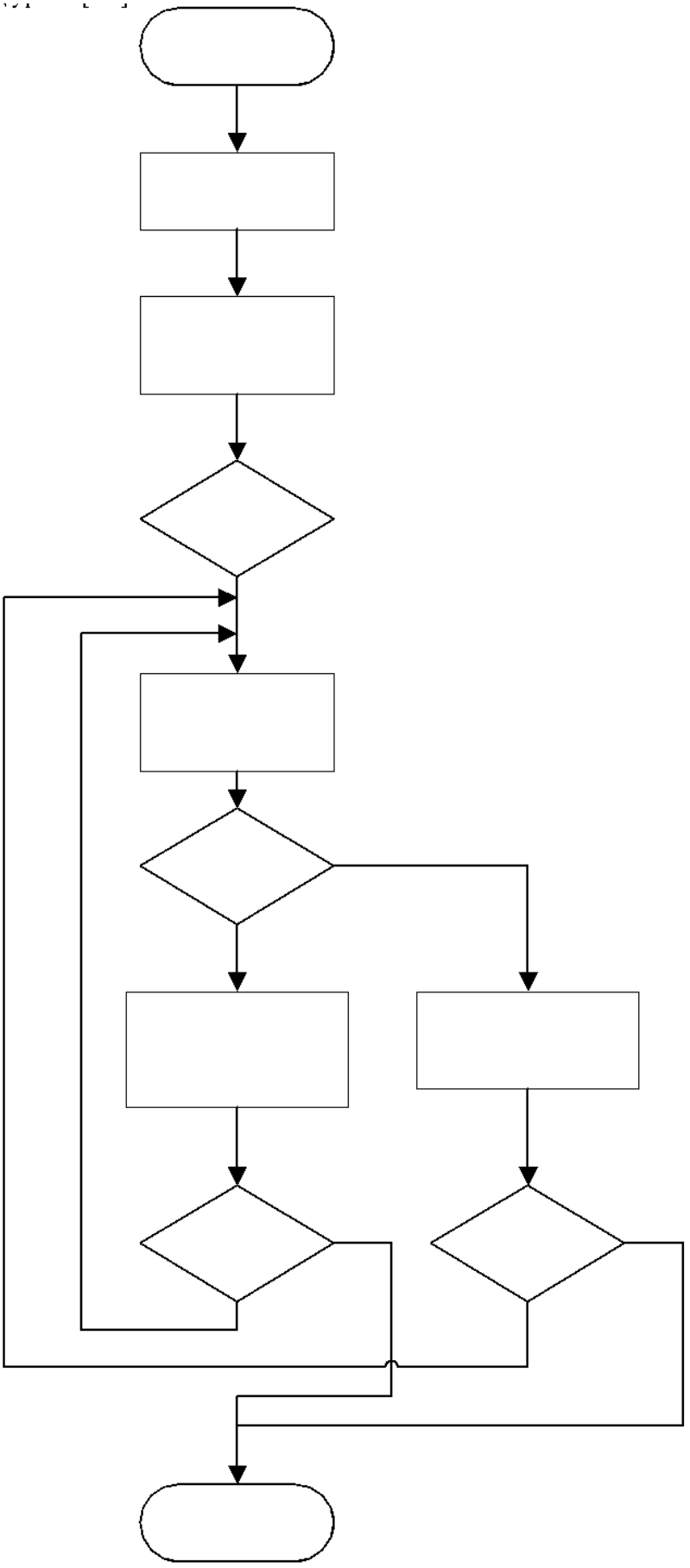

Рис. 1. Блок-схема программы

5. Физическое программирование

Программа состоит из одного файла structs.asm, в котором содержится код программы на языке ассемблера. Процедуры и функции в программе не используются.

Спецификация меток, использованных в программе, отображена в табл. 2.

Таблица 5

Спецификация меток

Имя	Тип	Формальные параметры	Назначение
start	-	-	Установка начальных значений регистров
iterate	-	-	Обработка массива, запись значений элементов в регистр DL, проверка веса
check_color	-	-	Проверка цвета
increase	-	-	Увеличение значения регистра SP
stop	-	-	Вызов прерывания INT 21h, окончание программы

6. Кодирование

В начале программы описывается структура данных Klava:

```
Klava STRUC
    keyboard_type   dw 100  ; percentage
    key_count       dw 104  ;
    weight          dw 500  ;
    color           dw 33   ; ANSI
Klava ENDS
```

Далее описывается массив D типа структуры Klava, ему задаются значения по умолчанию:

```
Kbs Klava<60, 68, 300, 33>, <100, 104, 350, 35>, <80, 87, 600, 34>...
```

Затем в сегменте CODE регистрам присваиваются начальные значения.

```
start:
    mov ax, @data           ; set AX to data segment
    mov DS, AX              ; set DS to point to data segment
    xor AX, AX              ; null AX
    mov CX, 15              ; set CX to the amount of keyboards (15)
    xor BX, BX
    xor DI, DI      ; null registers
    XOR SP, SP
```

Значение регистра CX (счётчик цикла) сравнивается с нулем. Если он больше нуля – управление передается в метку iterate:

```
    cmp CX, 0
    JA iterate
```

Описывается метка iterate – цикл, в котором определяется, весит ли клавиатура меньше 400 грамм:

```
 iterate:
    DEC CX          ; decrement CX
    XOR DX, DX              ; empty DX
    MOV DX, Kbs.weight[BX]  ; DX = i'th kb weight
```

Значение регистра-счётчика CX уменьшается на 1. Регистр DX обнуляется, затем ему присваивается значение weight текущей клавиатуры. DX сравнивается с 400. Если DX меньше 400 – управление передается в метку check_color. Иначе – значение регистра CX (счетчик цикла) сравнивается с нулем. Если он больше нуля – управление передается обратно в начало метки iterate, в противном случае – управление передается в метку stop:

```
JLE check_color          ; true => jump to color check
ADD BX, TYPE Klava       ; shift address
CMP CX, 0                ; false => check CX != 0
JA iterate          ; true => next iteration
JMP stop            ; false => stop
```

Описывается метка check_color – цикл, в котором проверяется, розовая ли клавиатура. Значение регистра DX обнуляется, затем ему присваивается значение color текущей клавиатуры. :

```
check_color:       ; weight correct => check color
XOR DX, DX               ; empty DX
MOV DX, Kbs.color[BX]   ; DX = i'th kb color
CMP DX, 33               ; DX == 33
```

Если оно равно 33 (magenta в ANSI), управление передаётся в метку increase. В противном случае, CX сравнивается с нулём. Если CX > 0, управление передаётся в метку iterate, иначе - в stop:

```
JE increase         ; true => jump to addition
ADD BX, TYPE Klava       ; shift address
CMP CX, 0                ; check CX != 0
JA iterate          ; true => next iteration
JMP stop            ; false => stop
```

В метке stop вызывается прерывание. Программа завершается:

```
stop:
MOV AX, 4C00h
INT 21h
```

Полный код программы представлен в приложении 3.

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	<60, 68, 526, 29>, <100, 104, 214, 28>, <80, 87, 304, 33>, <60, 68, 305, 26>, <100, 104, 548, 34>, <80, 87, 379, 28>, <60, 68, 313, 34>, <100, 104, 235, 30>, <80, 87, 282, 35>, <60, 68, 346, 29>, <100, 104, 532, 26>, <80, 87, 367, 29>, <60, 68, 567, 35>, <100, 104, 254, 31>, <80, 87, 313, 34>	struct.asm	1
2	<60, 68, 300, 33>, <100, 104, 350, 35>, <80, 87, 600, 34>, <60, 68, 300, 33>, <100, 104, 350, 35>, <80, 87, 600, 34>, <60, 68, 300, 33>, <100, 104, 350, 35>, <80, 87, 600, 34>, <60, 68, 300, 33>, <100, 104, 350, 35>, <80, 87, 600, 34>, <60, 68, 300, 33>, <100, 104, 350, 35>, <80, 87, 600, 34>	struct.asm	5
3	<60, 68, 526, 29>, <100, 104, 214, 28>, <80, 87, 604, 33>, <60, 68, 305, 26>, <100, 104, 548, 34>, <80, 87, 379, 28>, <60, 68, 313, 34>, <100, 104, 235, 30>, <80, 87, 282, 35>, <60, 68, 346, 29>, <100, 104, 532, 26>, <80, 87, 367, 29>, <60, 68, 567, 35>, <100, 104, 254, 31>, <80, 87, 313, 34>	struct.asm	0

Продолжение табл. 3

1	2	3	4
4	<60, 68, 461, 35>, <100, 104, 202, 30>, <80, 87, 485, 32>, <60, 68, 272, 36>, <100, 104, 392, 35>, <80, 87, 345, 34>, <60, 68, 453, 35>, <100, 104, 216, 33>, <80, 87, 361, 35>, <60, 68, 227, 33>, <100, 104, 401, 32>, <80, 87, 208, 36>, <60, 68, 312, 34>, <100, 104, 348, 30>, <80, 87, 356, 31>	struct.asm	2
5	<'Seld pod shuboy','Seld',0,189>, <'Kartofel pecheniy', 'Kartofel', 0, 145>, <'Kraboviy salat', 'Krabovie palochki', 0, 128>, <'Holodec', 'Myaso', 1, 350>, <'Salat Mimoza', 'Kartofel', 0, 183>, <'Cezar','Salat',1,218>, <'Yaichnitsa s kolbasoy','Yaico',1,230>, <'Kuritsa pechenaya', 'Kuritsa', 1, 223>, <'Golubci', 'Kapusta', 1, 139>, <'Mannaya kasha', 'Mannaya krupa', 0, 83>	struct.asm	15

Результаты тестов представлены в таблице 4.

Таблица 4

Результаты тестирования

Дата тестирования	Тестируемый модуль	Тестирование проводил	Описание теста	Результат тестирования
1	2	3	4	5
25.12.2023	structs.asm	Беляков А. А.	Тестирование с набором данных №1.	Успех
25.12.2023	structs.asm	Беляков А. А.	Тестирование с набором данных №2.	Успех
25.12.2023	structs.asm	Беляков А. А.	Тестирование с набором данных №3.	Успех
25.12.2023	structs.asm	Беляков А. А.	Тестирование с набором данных №4.	Успех
25.12.2023	structs.asm	Беляков А. А.	Тестирование с набором данных №5.	Успех

Заключение

В ходе выполнения курсовой работы была разработана программа, которая работает с массивом типа структуры, в котором записана информация о 15 клавиатурах. Программа подсчитывает количество клавиатур легче 400 грамм и розового цвета, и записывает это число в регистр SP.

В ходе разработки программы были освоены базовые возможности языка Assembler для процессоров на базе архитектуры микропроцессора Intel 8086, а также приобретены навыки работы с транслятором Turbo Assembler и компоновщиком Turbo Linker.

Список литературы

1. Методика и организация самостоятельной работы студентов: учебно-методическое пособие / Е.В. Ершов, Л.Н. Виноградова, В.В. Селивановских [и др.]. – Череповец: ФГБОУ ВПО ЧГУ, 2015. – 243 с.

2. Виноградова, Л. Н. Системное программирование: Учеб. Пособие. – Череповец: ФГБОУ ВПО ЧГУ, 2016. – 210 с.

3. Язык программирования — Википедия [электр.ресурс] https://ru.wikipedia.org/wiki/Язык_программирования. Дата обращения: 20.12.2023.

4. Ассемблер — SkillFactory [электр.ресурс] https://blog.skillfactory.ru/glossary/assembler. Дата обращения: 20.12.2023.

Приложение 1

МИНОБРАНАУКИ 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. _____ Ершов Е.В.

«____» _____2023 г.

Программирование на языке низкого уровня

Техническое задание на курсовую работу

Листов 8

Руководитель: Виноградова Л. Н.

Исполнитель: студент гр.

1ПИб-02-3оп-22

Беляков А. А.

21

2023 г.

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: 9 ноября 2023 года.

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

2. Назначение разработки

Основной задачей курсовой работы является создание программы, которая обрабатывает массив Kbs типа структуры данных Klava с информацией о клавиатурой, подсчитывает количество клавиатур розового цвета и легче 400 грамм, и записывает его в регистр SP.

3. Требования к программе

3.1. Требования к функциональным характеристикам

К разрабатываемой программе предъявляются следующие требования:

- программа должна обрабатывать массив Kbs типа структуры данных

Klava с использованием функций DOS;

- структура Klava должна содержать поля «Тип клавиатуры», «Количество клавиш», «Вес», «Цвет»;
- поля «Тип клавиатуры», «Количество клавиш», «Вес», «Цвет» должны представлять собой беззнаковые числа типа WORD;
- описание типа структуры может размещаться в любом месте программы, но обязательно до описания переменных этого типа;
- имена полей не должны совпадать с именами других объектов программы;
- не допускается вложенность структур;
- в массив Kbs должна быть записана информация о 15 клавиатурах;
- программа должна записывать в регистр SP число – количество клавиатур розового цвета с весом не более 400 грамм в данном массиве.

3.2. Требования к надежности

Программа должна работать без ошибок. Для этого после создания она должна быть протестирована. При возникновении каких-либо ошибок они должны быть исправлены.

3.3. Условия эксплуатации

Для корректной работы программы необходимо:

- наличие любого устройства, на котором возможно запускать файлы формата .asm (компьютер, ноутбук);
- наличие на устройстве современной версии любой операционной системы.

3.4. Требования к составу и параметрам технических средств

Для корректной работы программы необходимо:

- оперативная память: не менее 1 Гб для 32-разрядной ОС, 2 Гб для 64-разрядной ОС;
- процессор на базе архитектуры микропроцессора 8086 не менее чем с 2 ядрами и тактовой частотой не ниже 1 ГГц;
- разрешение экрана не менее 800 x 600;
- видеокарта с видеопамятью не менее 1 Гб;
- наличие мыши и клавиатуры.

3.5. Требования к информационной и программной совместимости

Для корректной работы программы необходимо:

- установленная операционная система (например, Windows XP, Vista, 7, 8, 8.1, 10, 11, macOS, Linux);
- установленный на устройстве транслятор «Turbo Assembler»;
- установленный на устройстве компоновщик «Turbo Linker»;
- установленный на устройстве эмулятор операционной системы MS-DOS «DOSBox».

3.6. Требования к маркировке и упаковке

Требования к маркировке и упаковке программы не предъявляются.

3.7. Требования к транспортированию и хранению

Программа может храниться и транспортироваться в виде файлов, необходимых для ее работы, на носителе информации (флешка, CD-диск). Такую программу можно распространять, передавая его с одного компьютера на другой с помощью флешки или диска, либо отправляя ее по электронной почте.

3.8. Специальные требования

Специальные требования к программе не предъявляются.

4. Требования к программной документации

4.1. Содержание расчетно-пояснительной записки

Программная документация должна содержать расчётно-пояснительную записку с содержанием:

1. Титульный лист
2. Аннотация
3. Оглавление
4. Введение
5. Изучение и описание предметной области
6. Постановка задачи
7. Описание созданного приложения
   - 7.1. Логическое проектирование
   - 7.2. Физическое проектирование
   - 7.3. Кодирование
   - 7.4. Тестирование
8. Заключение
9. Список литературы
10. Приложения
    - 10.1. Техническое задание
    - 10.2. Руководство пользователя
    - 10.3. Текст программы

4.2. Требования к оформлению

В данном пункте представлены требования к оформлению документации в соответствии с 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.
Страницы	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 кода программы – 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

В данном пункте описаны стадии и этапы разработки программы (табл. П1.2).

Таблица П1.2

Стадии и этапы разработки

Наименование этапа разработки	Сроки разработки	Результат выполнения	Отметка о выполнении
Определение темы для курсовой работы	09.11.2023	Утверждена тема для разработки	
Оформление технического задания	17.11.2023	Оформленное техническое задание	
Изучение и описание предметной области	21.11.2023	Получены теоретические знания для разработки программы	
Выбор структур данных	28.11.2023	Выбраны структуры данных	
Логическое проектирование	05.12.2023	Подготовлен проект логической части программы	
Физическое проектирование	12.12.2023	Подготовлен проект физической части программы	
Написание программы	18.12.2023	Готовая программа	
Тестирование и отладка	19.12.2023	Конечный вариант программы	
Оформление сопроводительной документации	20.12.2023	Оформленная сопроводительная документация	

7. Порядок контроля и приемки

В данном пункте описан порядок контроля и приемки курсовой работы (табл. П1.3).

Таблица П1.3

Порядок контроля и приемки

Наименование контрольного этапа выполнения курсовой работы	Сроки контроля	Результат выполнения	Отметка о приемке результата контрольного этапа
Сдача технического задания	17.11.2023	Согласованное техническое задания	
Сдача расчетно-пояснительной записки	21.12.2023	Согласованная расчетно-пояснительная записка	
Сдача курсовой работы	26.12.2023	Получение оценки за выполненную работу	

Приложение 2

Руководство пользователя

1. Общие сведения о программе

Файл программы называется «structs.exe». При запуске программа перебирает массив с информацией о кулинарных блюдах и записывает в регистр DI количество блюд без мяса.

2. Описание установки

Установка программы не требуется. Однако для ее запуска потребуется установить эмуляторе операционной системы MS-DOS.

3. Описание запуска

Программу необходимо запускать в эмуляторе операционной системы MS-DOS. Например, в программе «DOSBox».

Для начала необходимо смонтировать виртуальный диск с помощью команды mount и привязать к нему директорию с файлом кода на диске компьютера, например, «mount d: c:\asm», где d – название виртуального диска, c:\asm – расположение программы.

Затем нужно открыть смонтированный виртуальный диск, введя его название, например, «d:» (рис. П2.1).

После этих действий можно запустить программу в отладчике с помощью команды «td structs», где structs – название программы (рис. П2.2).

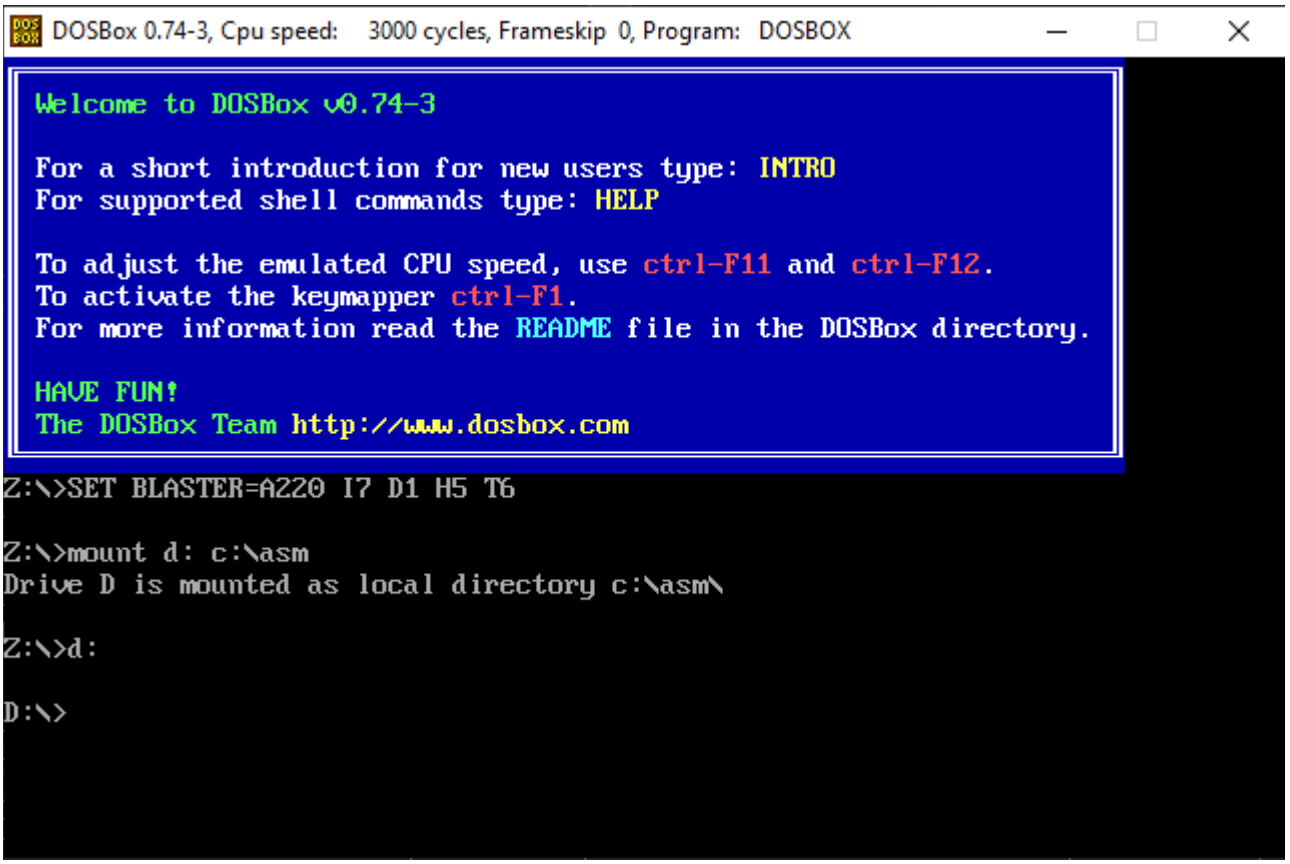

Рис. П2.1. Монтирование виртуального диска

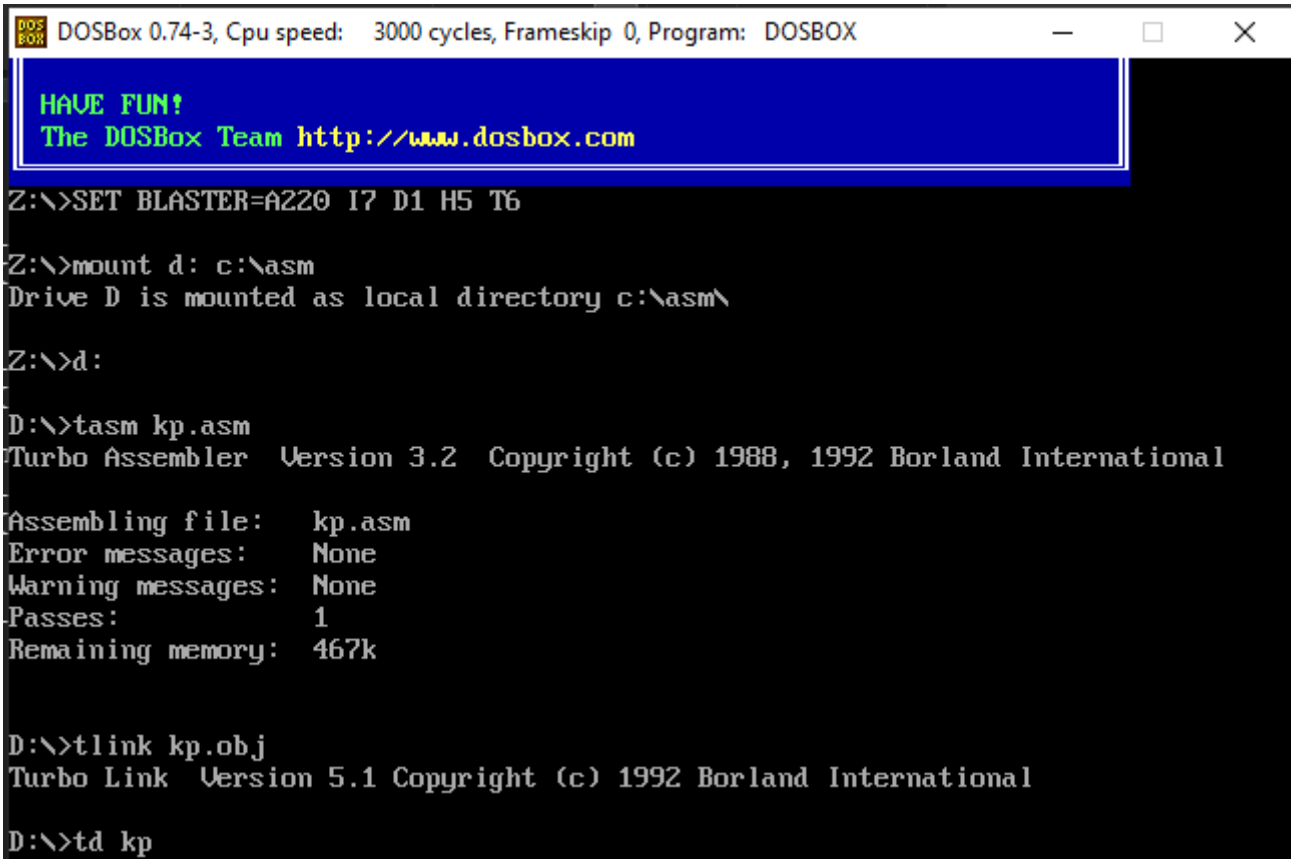

Рис. П2.2. Трансляция и компоновка программы, запуск в отладчике

4. Инструкции по работе

После запуска отладчика можно переходить к работе с программой (рис. П2.3). Программу можно выполнять пошагово, нажимая на клавишу F7 до тех пор, пока в коде не встретится прерывание. Либо можно выполнить всю программу сразу, нажав на клавишу F9. После выполнения программы количество блюд без мяса будет записано в регистр DI (рис П2.4).

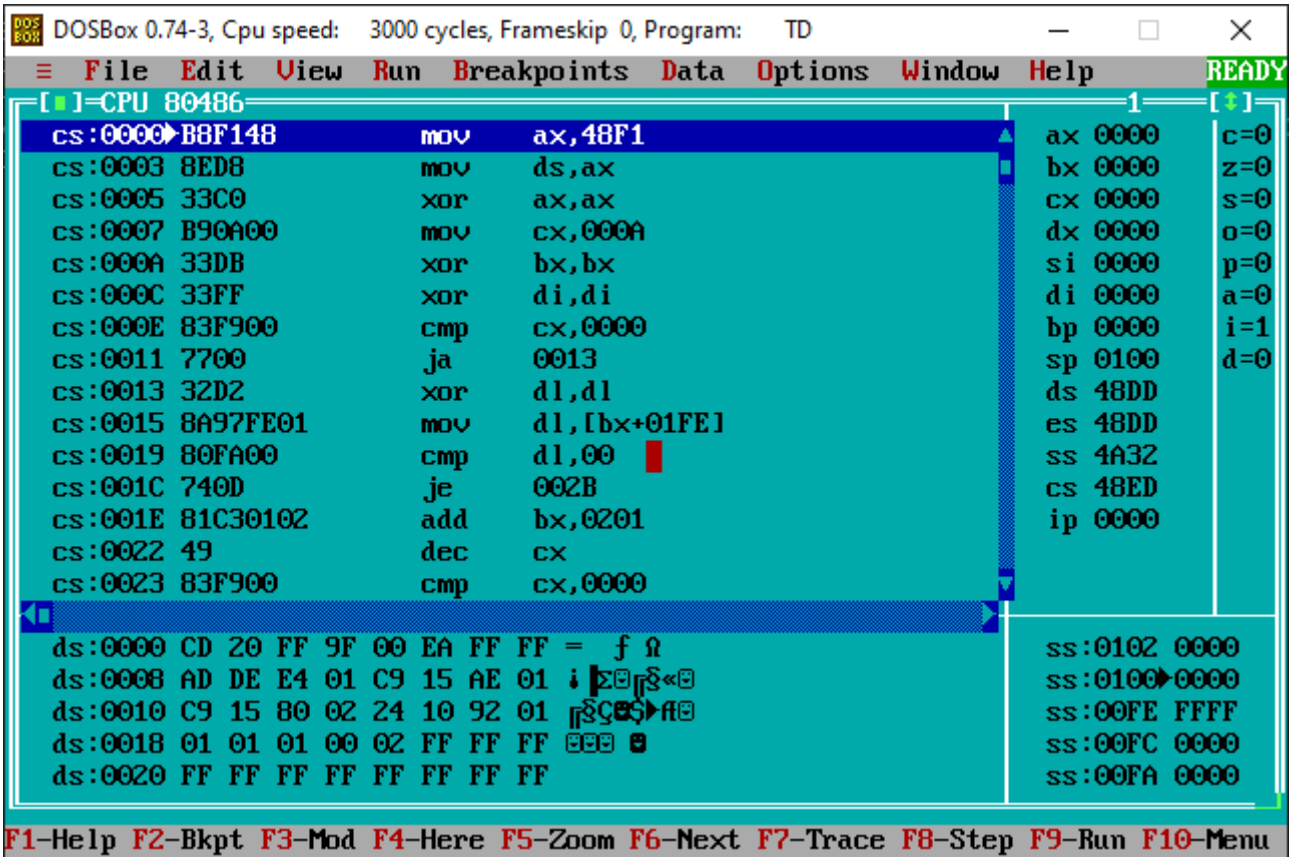

Рис. П2.3. Отладчик DOSBox в начале работы программы

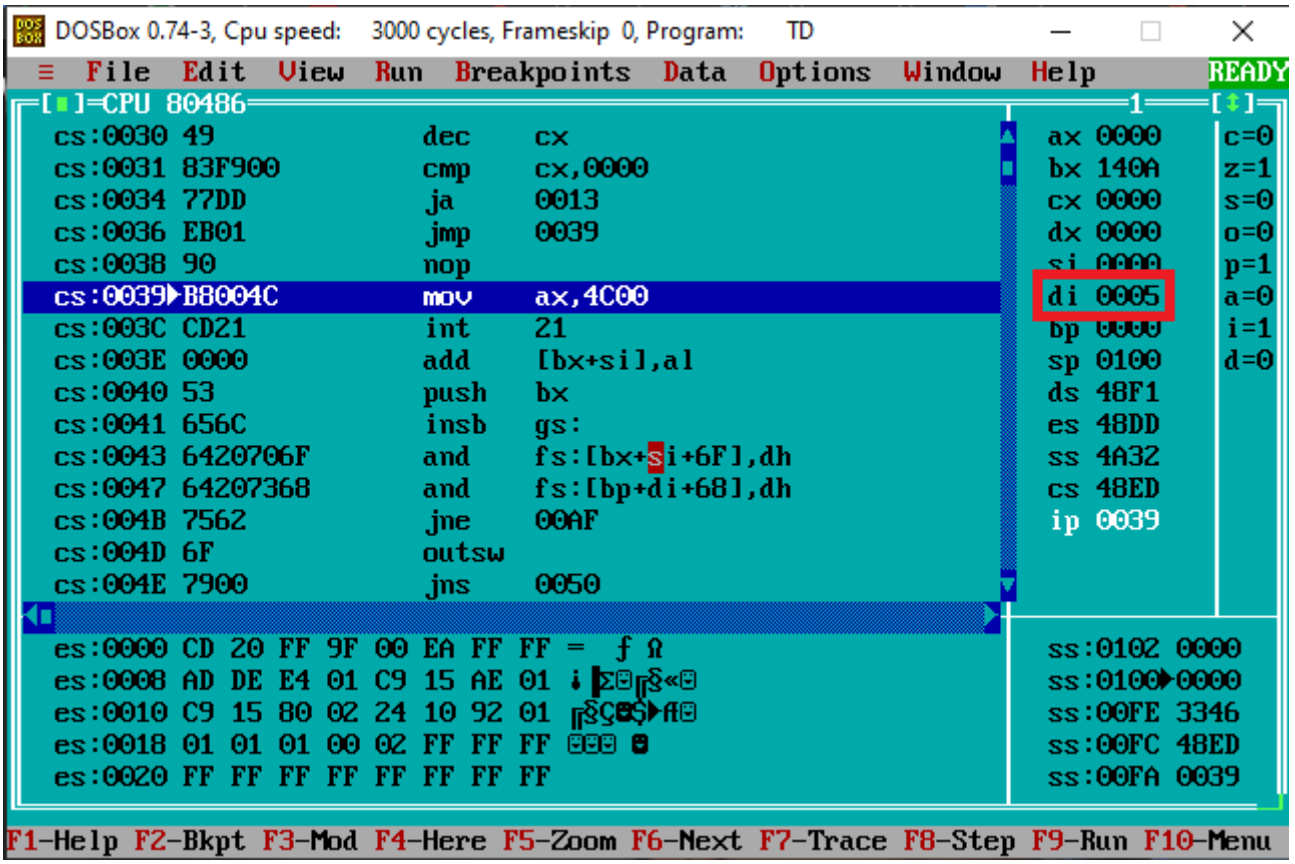

Рис. П2.4. Результат работы программы в регистре DI

5. Сообщения пользователю

При указании несуществующей директории файла появится сообщение «Directory … doesn’t exist!» («Директория … не существует!») (рис. П2.5). В случае появления такого сообщения нужно перепроверить правильность указания директории, а также её наличие на диске, и ввести ее еще раз.

При попытке открыть еще не смонтированный диск появится сообщение «Drive … does not exist!» («Диск … не существует!») (рис. П2.6). В случае появления такого сообщения нужно перемонтировать диск и попробовать открыть его еще раз.

При попытке запустить отладчик программы до привязки диска будет выведено сообщение «Illegal command: td» («Недопустимая команда: td») (рис. П2.7). В таком случае необходимо смонтировать диск и привязать к нему директорию с файлом программы.

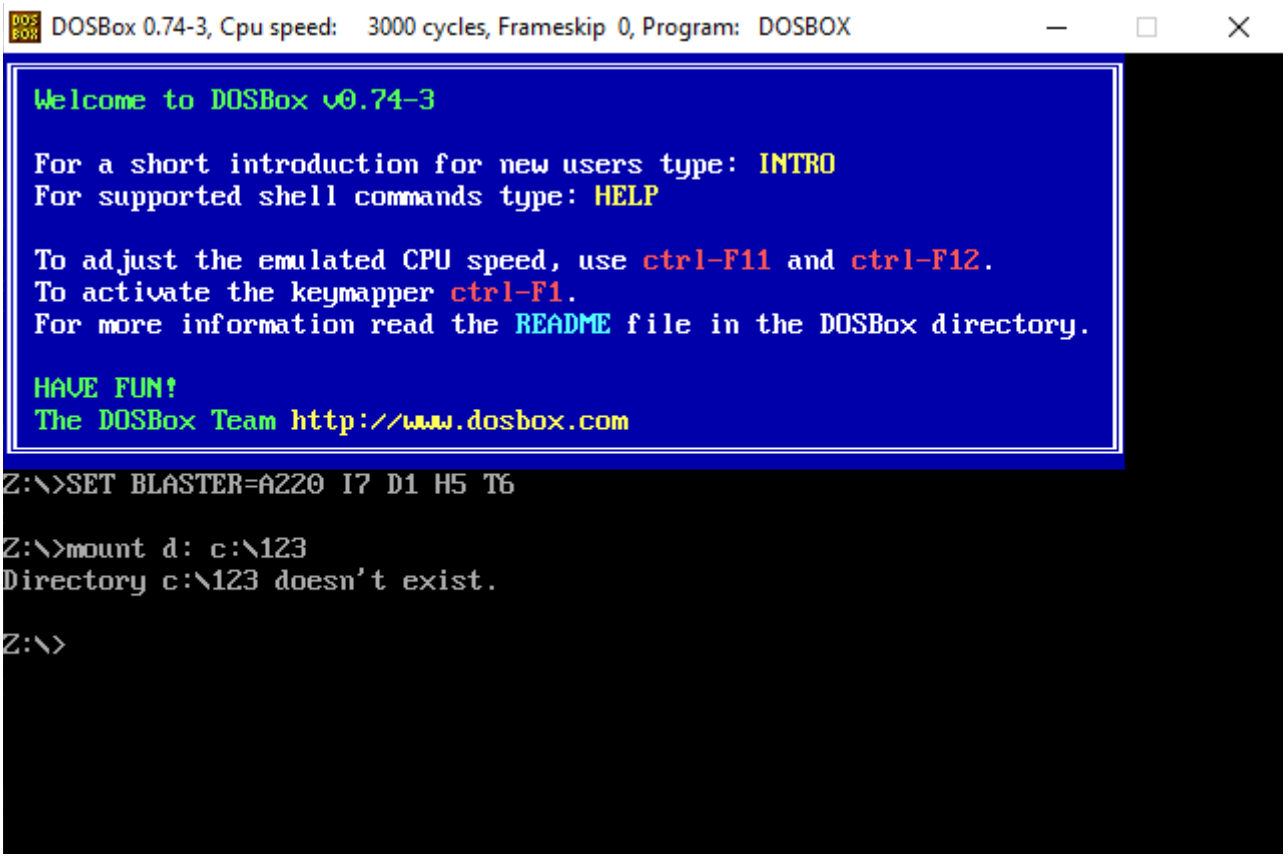

Рис. П2.5. Сообщение «Directory … doesn't exist»

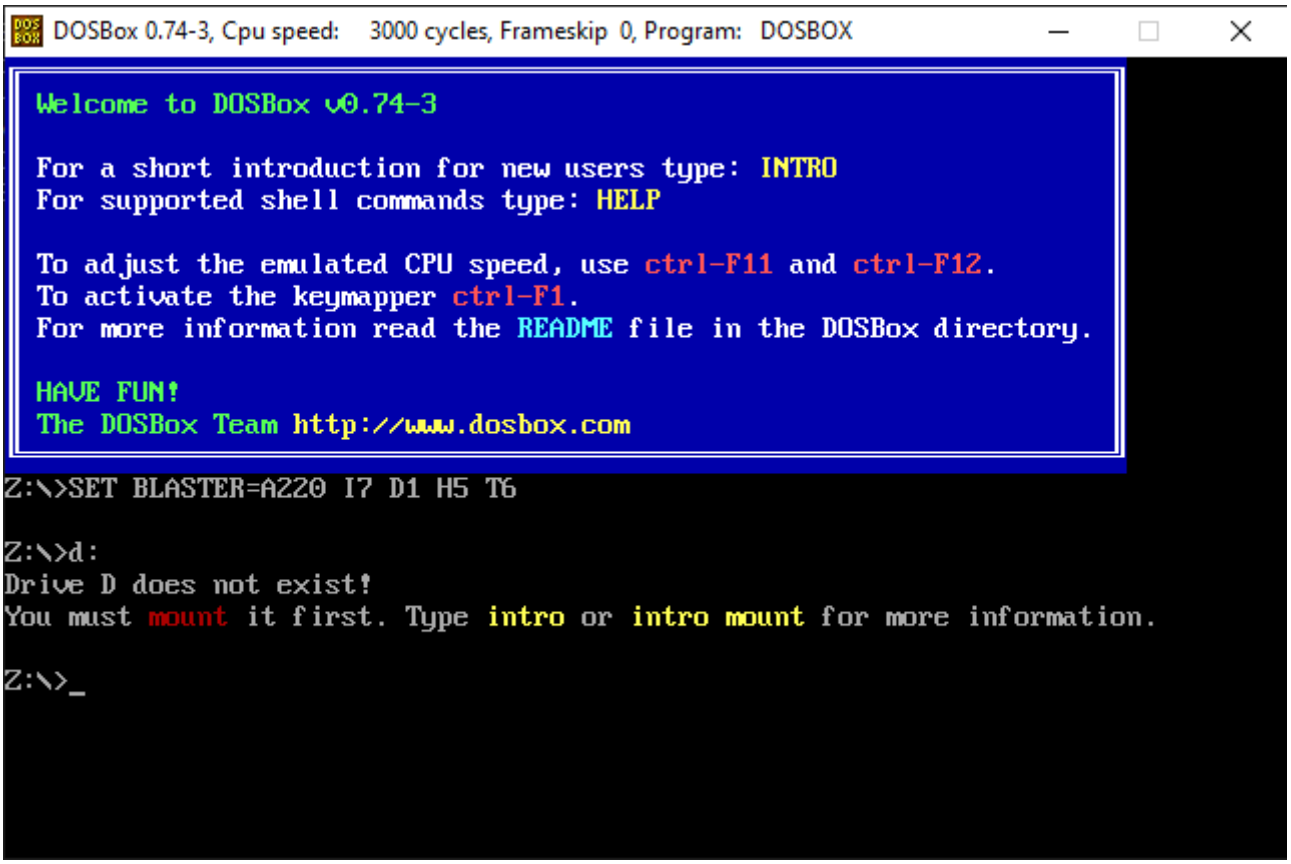

Рис. П2.6. Сообщение «Drive … does not exist»

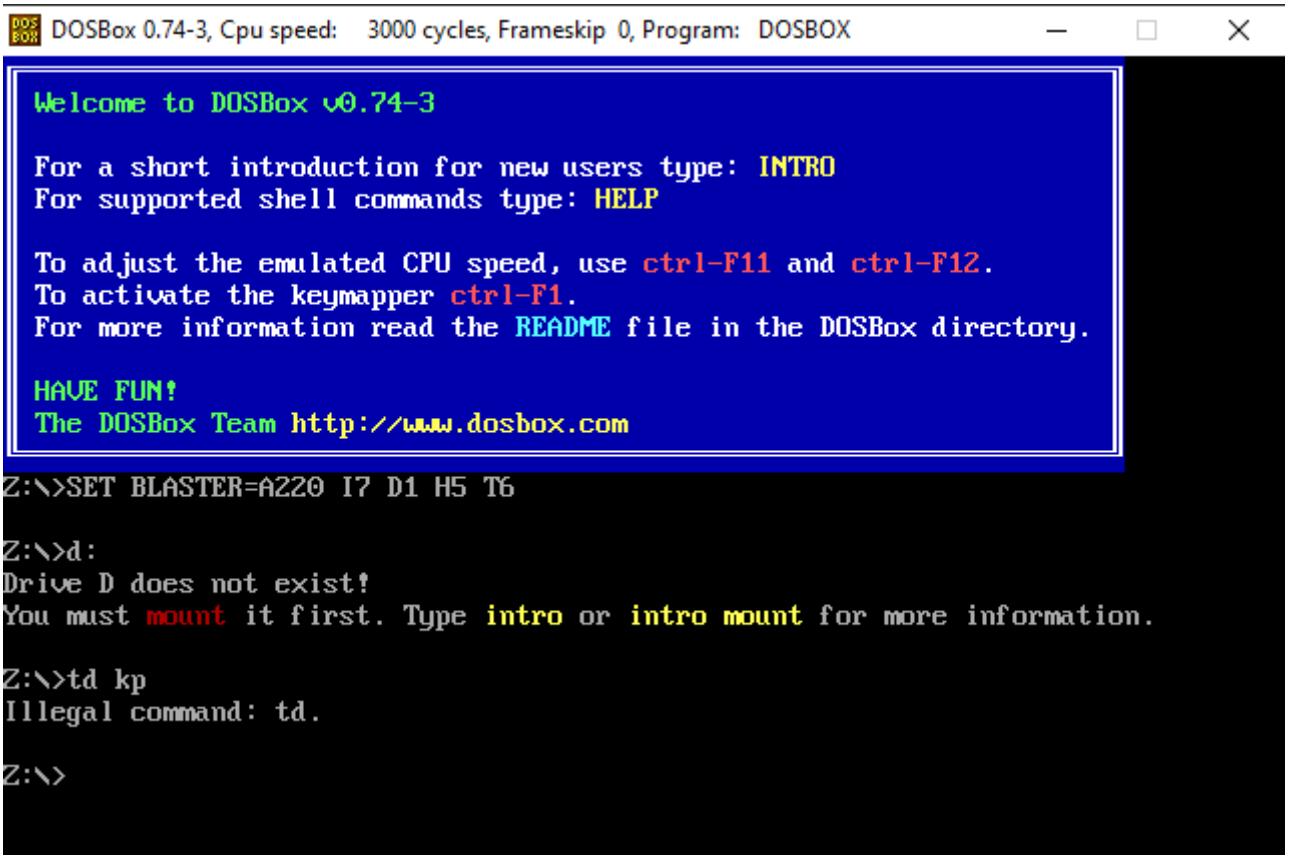

Рис П2.7. Сообщение «Illegal command: td»

Приложение 3

Текст программы

```
.model small
.stack 100h

.data
; define struct "Klava"
; size of Klava = 8 bytes
Klava STRUC
	keyboard_type   dw 100  ; percentage
	key_count		dw 104  ;
	weight      dw 500  ;
	color       dw 33   ; ANSI
Klava ENDS
; TKL: 80%, 87 keys
; 60%: 60%, 68 keys
; FS : 100%, 104 keys
Kbs Klava<60, 68, 1, 33>, <100, 104, 1, 33>, <80, 87, 1, 33>, <60, 68, 1, 33>, <100, 104, 1, 33>, <80, 87, 1, 33>, <60, 68, 1, 33>, <100, 104, 1, 33>,
<80, 87, 1, 33>, <60, 68, 1, 33>, <100, 104, 1, 33>, <80, 87, 1, 33>, <60, 68, 1, 33>, <100, 104, 1, 33>, <80, 87, 1, 33>,
.code
ORG 100h
start:
	mov ax, @data		; set AX to data segment
	mov DS, AX		; set DS to point to data segment
	xor AX, AX		; null AX
	mov CX, 15		; set CX to the amount of keyboards (15)
	xor BX, BX
	xor DI, DI		; null registers
	XOR SP, SP
	cmp CX, 0
	JA iterate

	iterate:
	DEC CX			; decrement CX
	XOR DX, DX		; empty DX
	MOV DX, Kbs.weight[BX]  ; DX = i'th kb weight
	CMP DX, 400		; DX <= 400
	JLE check_color		; true => jump to color check
	ADD BX, TYPE Klava		; shift address
	CMP CX, 0		; false => check CX != 0
	JA iterate		; true => next iteration
	JMP stop		; false => stop
	check_color:		; weight correct => check color
	XOR DX, DX		; empty DX
	MOV DX, Kbs.color[BX]   ; DX = i'th kb color
	CMP DX, 33		; DX == 33
	JE increase		; true => jump to addition
	ADD BX, TYPE Klava		; shift address
	CMP CX, 0		; check CX != 0
	JA iterate		; true => next iteration
	JMP stop		; false => stop
	increase:
	INC SP			; color correct => increment SP
	ADD BX, TYPE Klava		; shift address
	CMP CX, 0		; check CX != 0
	JA iterate		; true => next iteration
	JMP stop		; false => stop
	stop:
	INT 21h
END start
```