МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Белгородский государственный технологический университет им. В.Г. Шухова»
(БГТУ им. В.Г. Шухова)

Институт ЭИТУС

Кафедра «Информационных технологий»

Курсовая работа

дисциплина: «Архитектура информационных систем»

на тему:

«Реализация на языке Assembler программы, осуществляющей перемещение куба на экране монитора и изменение его цвета через заданный интервал времени»

Выполнил:
студент группы ИТ-22
Давыдов В.О.

Проверил:
к.т.н., доцент
Коробкова Е.Н.

Белгород 2019

Содержание

Введение

Компьютеры и другая вычислительная техника окружает нас повсюду. Многие люди сталкиваются с ней каждый день и воспринимают это как должное.

Но как же она работает? Как происходит выполнение программ? Действительно ли машина обладает интеллектом и сама всё делает?

Чтобы разобраться в этом, следует погрузиться в этот мир и испытать это на своём опыте при помощи реализации программы на языке программирования низкого уровня Ассемблер.

1. Постановка задачи и определение основных требований к разрабатываемому программному средству

1.1. Постановка задачи

Построить куб. Клавишами управления курсором предусмотреть возвожность перемещения по экрану. Цвет граней различен. По нажатию любой клавиши цвет меняется автоматически. При повторном нажатии клавиши смена цвета останавливается. Если в течении 7 сек. клавиша не нажата, смена цвета также останавливается.

1.2. Основание для разработки

Программное средство разрабатывается на основе учебного плана кафедры «Информационные технологии» для специальности 09.03.02 «Информационные системы и технологии» по дисциплине «Архитектура информационных систем».

1.3. Назначение программного продукта

Программа направлена на организацию досуга пользователя.

1.4. Основные требования к программному продукту

1.4.1. Требования к функциональным характеристикам.

Программа должна обеспечивать возможность выполнения следующих функций:

- Построение куба;
- Перемещение куба по экрану;
- Смена цвета фигуры по нажатию любой клавиши;
- Автоматические завершение смены цвета по истечению 7 сек., если не нажата любая клавиша;

1.4.2. Требования к надёжности

Предусмотреть блокировку некорректных действий пользователя при работе с программой, предусмотреть контроль вводимой пользователем информации. Программа должна функционировать корректно, в соответствии с разработанным алгоритмом.

1.4.3. Требования к составу и параметрам технических средств

Необходимо наличие IBM PC - совместимого ПК с графическим адаптером VGA. Дисковое пространство должно быть не менее 10 Кб, объем свободной оперативной памяти - не менее 200 Кб. Наличие манипулятора типа "клавиатура".

1.4.4. Требования к информационно-программной совместимости

Исправно функционирующее оборудование компьютера. Установленная ОС MS DOS или наличие элулятора DOSBox.

2. Теоретические сведения

Assembler — язык программирования низкого уровня, представляющий собой формат записи машинных команд, удобный для восприятия человеком.

Команды языка ассемблера один в один соответствуют командам процессора и, фактически, представляют собой удобную символьную форму записи (мнемокод) команд и их аргументов [1].

Типичные команды языка ассемблера:

- Команды передачи данных (mov и др.)
- Команды арифметической обработки целых чисел (add, sub, imul и др.)
- Логические операции (or, and, xor, shr и др.)
- Команды передачи управления (jmp, loop, ret и др.)
- Сдвиговые операции (SHL/SAL, SHR/SAH и др.) [2].

Команды пересылки данных

- MOV <операнд-получатель>, <операнд-источник> — копирует данные из операнда-источника в операнд-получатель
- XCHG <операнд1>, <операнд2> – позволяет обменять содержимое двух операндов [3].

Команды арифметической обработки целых чисел

- MUL <операнд> – команда умножения чисел без знака. У этой команды только один операнд — второй множитель, который должен находиться в регистре или в памяти. Местоположение первого множителя и результата задаётся неявно и зависит от размера операнда;
- DIV <операнд> – команда деления чисел без знака. У этой команды один операнд — делитель, который должен находиться в регистре или в памяти. Местоположение делимого, частного и остатка задаётся неявно и зависит от размера операнда;

операнд1>, <операнд2>– команда сложения. Результат заносится в <операнд1>;

операнд1>, <операнд2> – команда вычитания. Результат заносится в <операнд1>;

- DEC <операнд> - декремент
- INC <операнд> - инкремент [4].

Регистры

Регистры – это ячейки памяти, расположенные в процессоре. Их достоинство заключается в гораздо большем быстродействии, чем у оперативной памяти. Недостаток – их очень мало, всего чуть больше десятка.

На рисунке ниже представлены регистры процессора 8086. Такие же самые регистры имеются во всех старших моделях процессоров - 286, 386, 486 и Pentium (хотя старшие процессоры имеют дополнительные регистры и расширения). Если использовать при написании программ только эти регистры, то программы (по идее) будут исполнятся на всех компьютерах с Intel совместимыми процессорами. Регистры делятся на пять категорий:

- Регистры общего назначения (AX, BX, CX, DX)
- Регистровые указатели и индексные регистры (SP, BP, SI, DI)
- Сегментные регистры (CS, DS, SS, ES)
- Регистр командного указателя (IP)
- Регистр флагов

Все регистры процессора 8086 являются 16-битовыми. Кроме того, четыре регистра общего назначения – AX, BX, CD, DX - разделены на старшую (high) и младшую (low) 8-битовые половины. Это позволяет оперировать либо всем 16-битовым регистром (например, AX), либо обращаться отдельно к старшей (AH) или младшей (AL) половинам регистра.

Регистры общего назначения — это регистры данных, каждый из которых помимо хранения операндов и результатов операций имеет еще и свое специфическое назначение:

- регистр AX (accumulator) — умножение, деление, обмен с устройствами ввода/вывода (команды ввода и вывода);
- регистр BX (base) — базовый регистр в вычислениях адреса, часто указывает на начальный адрес (называемый базой) структуры в памяти;
- регистр CX (count) — счетчик циклов, определяет количество повторов некоторой операции;
- регистр DX (data) — определение адреса ввода/вывода, так же может содержать данные, передаваемые для обработки в подпрограммы.

Регистровые указатели и индексные регистры тесно связаны с определёнными операциями.

Регистровые указатели SP и BP обеспечивают доступ к данным в сегменте стека.

- регистр SP (Stack Pointer) – всегда указывает на вершину стека, позволяет временно хранить адреса и иногда данные;
- регистр BP (Base Pointer) – обычно адресует переменные, хранимые в стеке, облегчает доступ к параметрам (данным и адресам), переданным через стек.

Индексные регистры SI и DI могут применятся для расширенной адресации, для использования в операциях сложения и вычитания, а также для операций над байтовыми строками (в языке ассемблера байтовая строка представляет собой просто ряд последовательных байт).

- регистр SI (Source Index) – является индексом источника и применяется для некоторых операций над строками (обычно связан с регистром DS);

- регистр DI (Destination Index) – является индексом назначения и применяется так же для строковых операций (обычно связан с регистром

Сегментные регистры (CS, DS, SS, ES) определяют в памяти начала четырех 64 Кбайтовых сегментов, которые называются текущими сегментами. Программа может распределять более четырех сегментов, но при этом для адресации дополнительных сегментов она должна перемещать, соответствующие им правильные значения адресов между одним или несколькими сегментными регистрами. Сегментные регистры строго специализированы. С помощью них нет возможности выполнять математические вычисления или хранить в них результаты других операций. Действительный порядок сегментов не обязательно совпадает с порядком, показанным на рисунке. Сегменты могут в любом порядке храниться в произвольных местах памяти.

- регистр CS (Code Segment) – содержит начальный адрес сегмента (начало машинного кода программы). Этот адрес плюс значение смещения в командном указателе (IP) определяет адрес команды, которая должна быть выбрана для выполнения;
- регистр DS (Data Segment) – содержит начальный адрес сегмента данных (переменных, строк и т.п. данных, которыми оперирует программа);
- регистр SS (Stack Segment) – содержит начальный адрес сегмента стека;
- регистр ES (Extra Segment) – является вспомогательным регистром, используется при некоторых операциях над строками. В большинстве программ в ES и DS содержатся одинаковые адреса, но он может упрощать некоторые операции, связанные с этими регистрами.

Регистр командного указателя IP (Instruction Pointer) содержит смещение на команду, которая должна быть выполнена. Обычно при программировании этот регистр не используют.

Регистр флагов так же состоит из 16 бит, из них используются только 9. Это регистр состояния процессора. Биты регистра состояния устанавливаются или очищаются в зависимости от результата исполнения предыдущей команды и используются некоторыми командами процессора. Биты регистра состояния могут также устанавливаться и очищаться специальными командами процессора. Отдельные биты флагов представляют одиночными буквами O, D, I, T, S, Z, A, P, C или двумя буквами OF, DF, IF, TF, SF, ZF, AF, PF И CF.

- CF (Carry Flag) — флаг переноса при арифметических операциях. Содержит перенос из старшего бита после арифметических операций, а также последний бит при сдвигах или циклических сдвигах.
- PF (Parity Flag) — флаг четности результата, показывает четность младших восьмибитовых данных (1 – четное, 0 – нечетное).
- AF (Auxiliary Flag) — флаг дополнительного переноса. Содержит перенос из 3 бита для 8-битовых данных, используется для специальных арифметических операций.
- ZF (Zero Flag) — флаг нулевого результата. Показывает результат арифметических операций и операций сравнения (0 – ненулевой, 1 – нулевой результат).
- SF (Sign Flag) — флаг знака (совпадает со старшим битом результата, 0 – плюс, 1 - минус).
- TF (Trap Flag) — флаг пошагового режима (используется при отладке).
- IF (Interrupt-enable Flag) — флаг разрешения аппаратных прерываний.
  irection Flag) — флаг направления при строковых операциях. Обозначает левое или правое направление пересылки или сравнения строковых данных.
- OF (Overflow Flag) — флаг переполнения. Указывает на переполнение старшего бита при арифметических командах.

Стек

Стеком называется структура данных, организованная по принципу LIFO («Last In — First Out» или «последним пришёл — первым ушёл»). Стек является неотъемлемой частью архитектуры процессора и поддерживается на аппаратном уровне: в процессоре есть специальные регистры (SS, BP, SP) и команды для работы со стеком.

Стек используется для сохранения содержимого регистров, используемых программой, перед вызовом подпрограммы, которая, в свою очередь, будет использовать регистры процессора «в своих личных целях».

Стек располагается в оперативной памяти в сегменте стека, и поэтому адресуется относительно сегментного регистра SS. Шириной стека называется размер элементов, которые можно помещать в него или извлекать. Для стека существуют две основные операции:

- PUSH — добавление элемента на вершину стека;
- POP — извлечение элемента с вершины стека [5].

Команды переходов

- Команда безусловного перехода – имеет следующий синтаксис: JMP <операнд>. Операнд указывает адрес перехода. Существует два способа указания этого адреса, соответственно различают прямой и косвенный переходы. При косвенном переходе в команде перехода указывается не адрес перехода, а регистр или ячейка памяти, где этот адрес находится. Содержимое указанного регистра или ячейки памяти рассматривается как абсолютный адрес перехода. Косвенные переходы используются в тех случаях, когда адрес перехода становится известен только во время работы программы. Если в команде перехода указывается метка команды, на которую надо перейти, то переход называется прямым.

- Команда условного перехода — В системе команд процессора архитектуры x86 не предусмотрена поддержка условных логических структур, характерных для языков высокого уровня. Однако на языке ассемблера с помощью набора команд сравнения и условного перехода вы можете реализовать логическую структуру любой сложности. В языке высокого уровня любой условный оператор выполняется в два этапа. Сначала вычисляется значение условного выражения, а затем, в зависимости от его результата, выполняются те или иные действия. Проводя аналогию с языком ассемблера, можно сказать, что сначала выполняются такие команды, как CMP, AND или SUB, влияющие на флаги состояния процессора. Затем выполняется команда условного перехода, которая анализирует значение нужных флагов, и в случае если они установлены, выполняют переход по указанному адресу. Что касается команд условного перехода, то их достаточно много, но все они записываются единообразно:

Прерывания

Прерывание означает временное прекращение основного процесса вычислений для выполнения некоторых запланированных или незапланированных действий, вызванных работой устройств или программы.

В зависимости от источника различают прерывания:

- Аппаратные (внешние) – реакция процессора на физический сигнал от некоторого устройства. Возникают в случайные моменты времени, а значит – асинхронные
- Программные (внутренние) – возникает в заранее запланированный момент времени — синхронные

- Исключения – разновидность программных прерываний, реакция процессора на некоторую нестандартную ситуацию возникшую во время выполнения команды;

Вектор прерываний – адрес процедуры обработки прерываний. Адреса размещаются в специальной области памяти доступной для всех подпрограмм. Вектор прерывания одержит 4 байта: старшее слово содержит сегментную составляющую адреса процедуры обработки исключения, младшее — смещение.

Вызов прерывания осуществляется с помощью директивы INT <номер прерывания>.

- 00h – 1Fh – прерывания BIOS
- 20h – 3Fh – прерывания DOS
- 40h – 5Fh – зарезервировано
- 60h – 7Fh – прерывания пользователя
- 80h – FFh – прерывания Бейсика[7].

Подпрограммы

Процедура (подпрограмма) — это основная функциональная единица декомпозиции (разделения на несколько частей) некоторой задачи. Процедура представляет собой группу команд для решения конкретной подзадачи и обладает средствами получения управления из точки вызова задачи более высокого приоритета и возврата управления в эту точку.

Синтаксис описания процедуры:

<имя процедуры> PROC <расстояние>

<тело процедуры>

<имя процедуры> ENDP

<расстояние> может принимать значения near или far и характеризует возможность обращения к процедуре из другого сегмента кода. По умолчанию

принимает значение NEAR. Процедура может размещаться в любом месте программы, но так, чтобы на нее случайным образом не попало управление. Если процедуру просто вставить в общий поток команд, то микропроцессор будет воспринимать команды процедуры как часть этого потока. Учитывая это обстоятельство, есть следующие варианты размещения процедуры в программе:

- в начале программы (до первой исполняемой команды);
- в конце (после команды, возвращающей управление операционной системе);
- промежуточный вариант — тело процедуры располагается внутри другой процедуры или основной программы;
- в другом модуле [1].

Работа с графикой

Для того, чтобы вывести на экран графическое изображение необходимо воспользоваться нижним уровнем операционной системы - базовой системы ввода-вывода (Basic In-Out System, BIOS). Программы BIOS находятся в постоянном запоминающем устройстве (ПЗУ) BIOS. В отличие от DOS, ко всем функциям которой можно обратиться с помощью прерывания int 21h, в BIOS за каждым устройством компьютера закреплено свое прерывание. Так, прерывание символьной и текстовой информации, смену шрифтов, настройку цветовой палитры, работу с графическим изображением и т.д. [1].

3. Проектирование программного продукта

3.1. Разработка структурной схемы программного средства

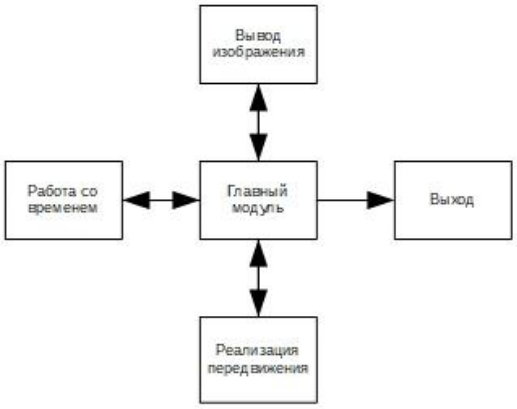


Рисунок 1 Структурная схема программы

3.2. Разработка алгоритмов программы

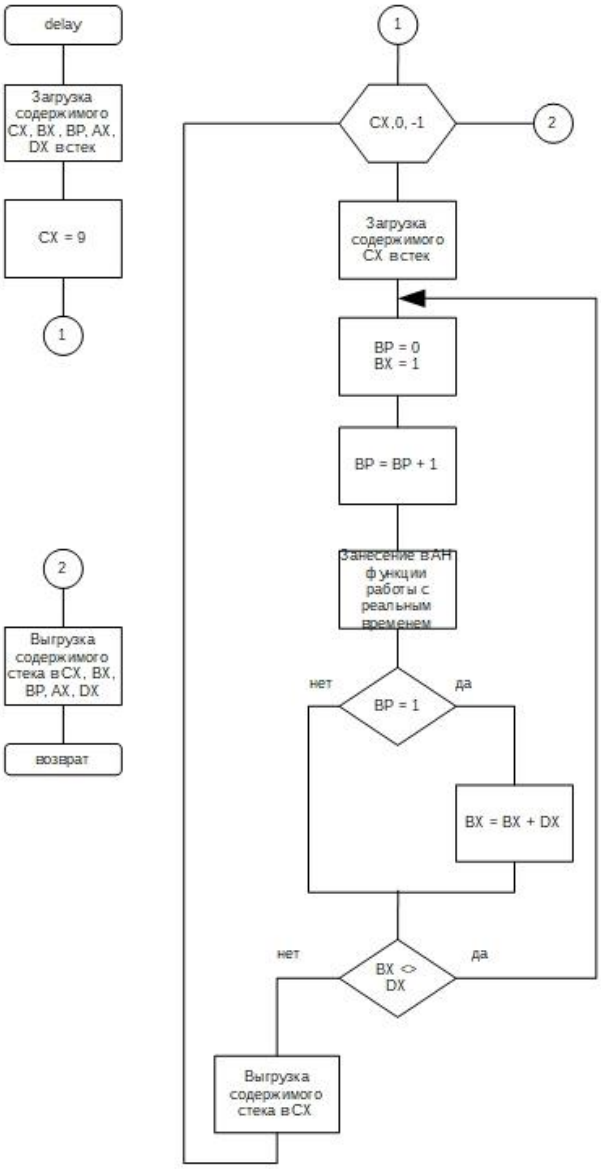


Рисунок 2 Процедура задержки в 0.5 сек

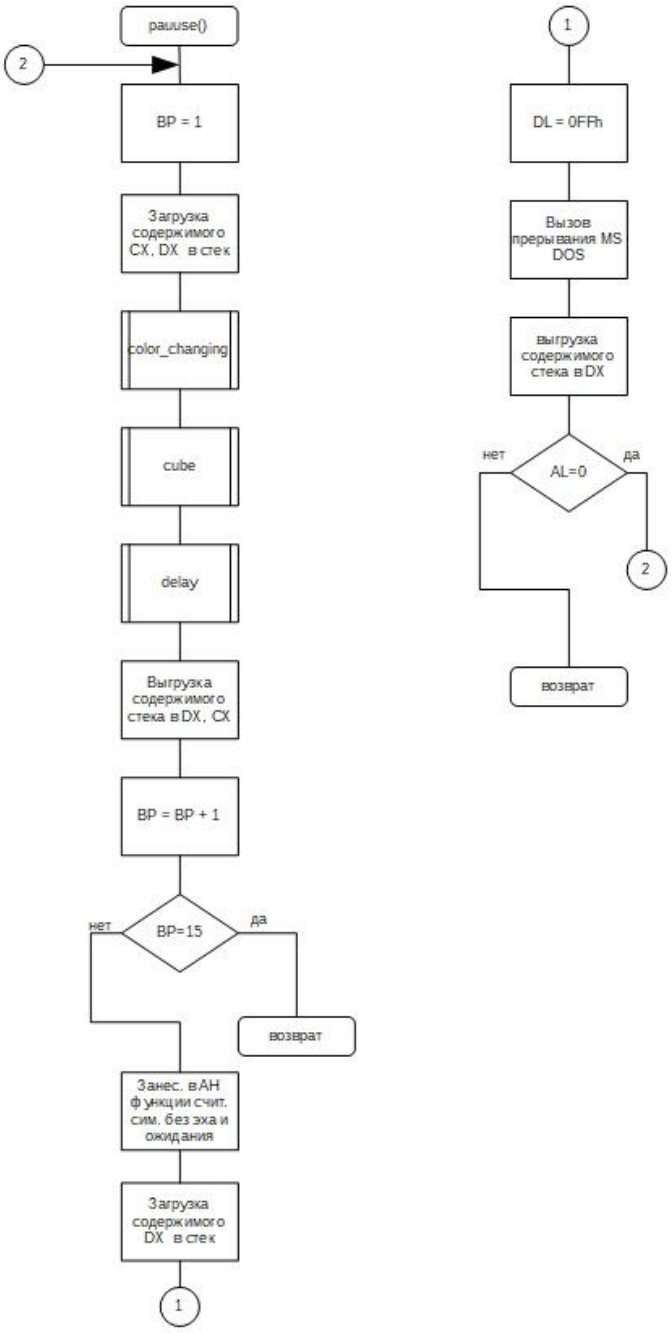


Рисунок 3 Процедура проверки нажатия клавиши во время смены цвета куба

Рисунок 4 Процедура возврата исходного цвета

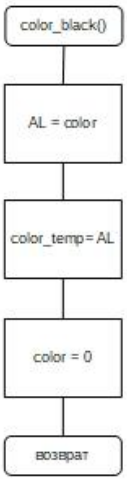


Рисунок 5 : Процедура установки чёрного цвета

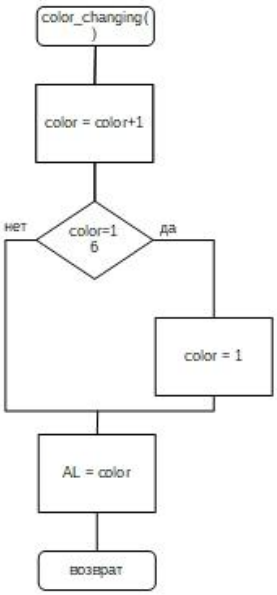


Рисунок 6 Процедура смены цвета

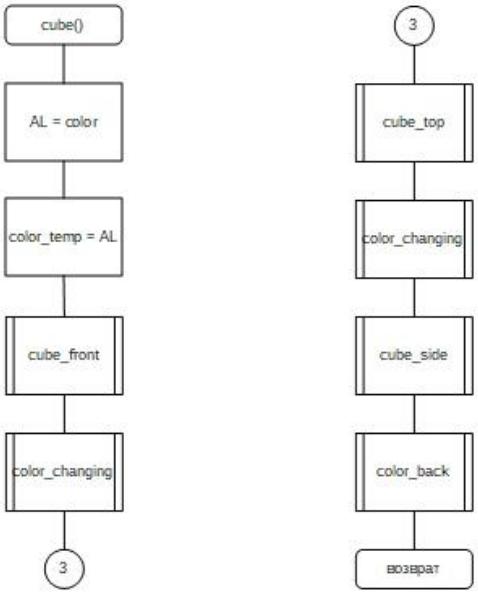


Рисунок 7 Процедура прорисовки куба

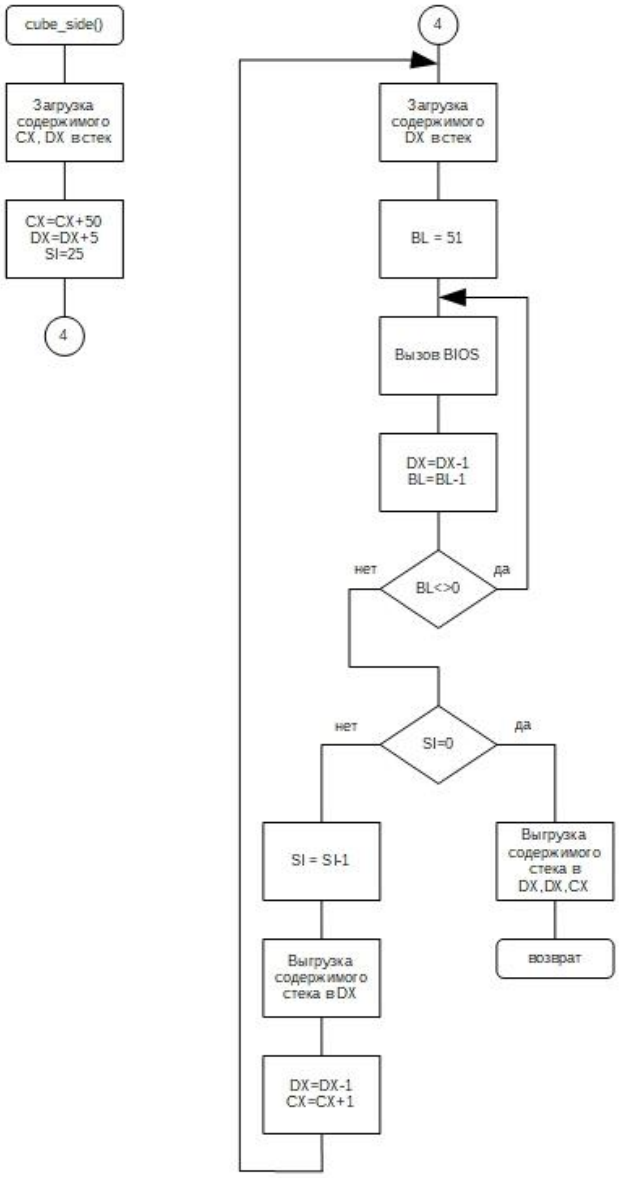


Рисунок 8 Процедура прорисовки боковой грани куба

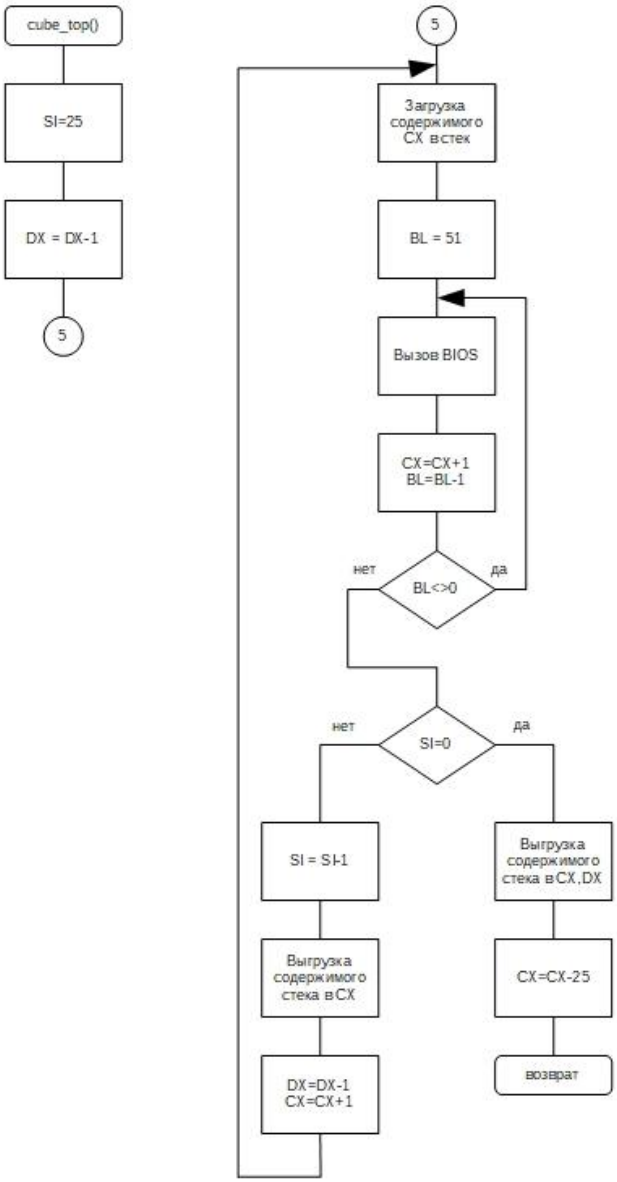


Рисунок 9 Процедура прорисовки верхней грани куба

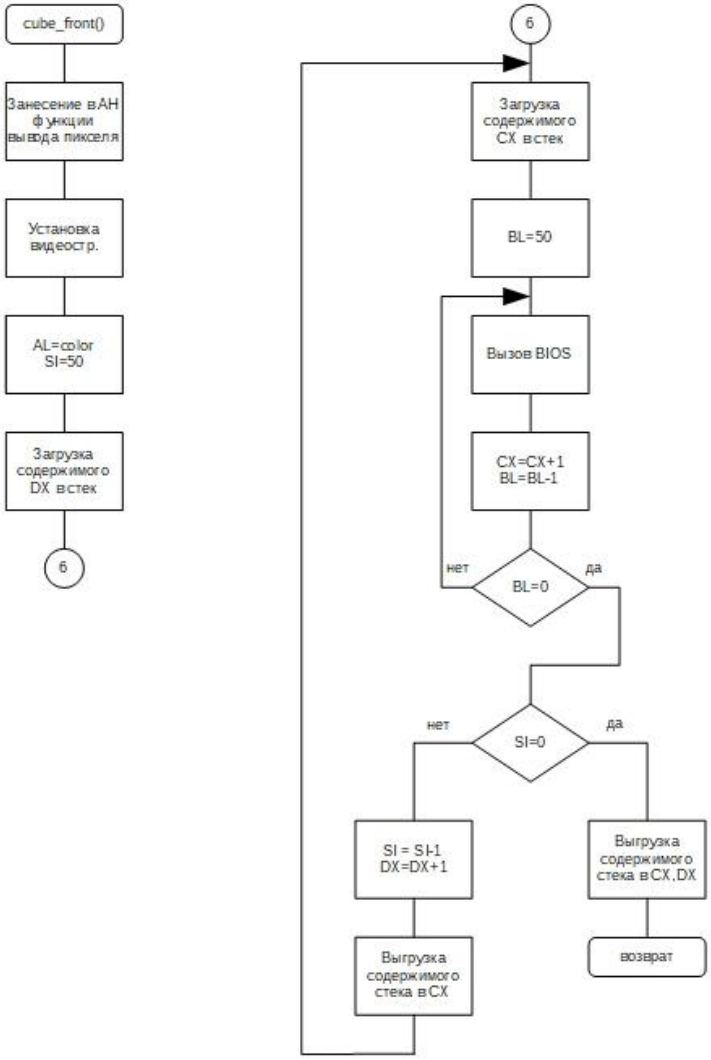


Рисунок 10 Процедура прорисовки передней грани куба

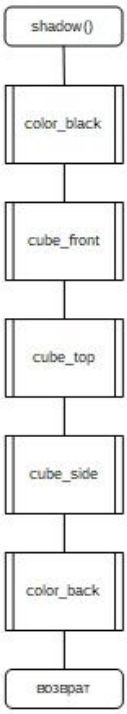


Рисунок 11 Процедура прорисовки «тени» куба

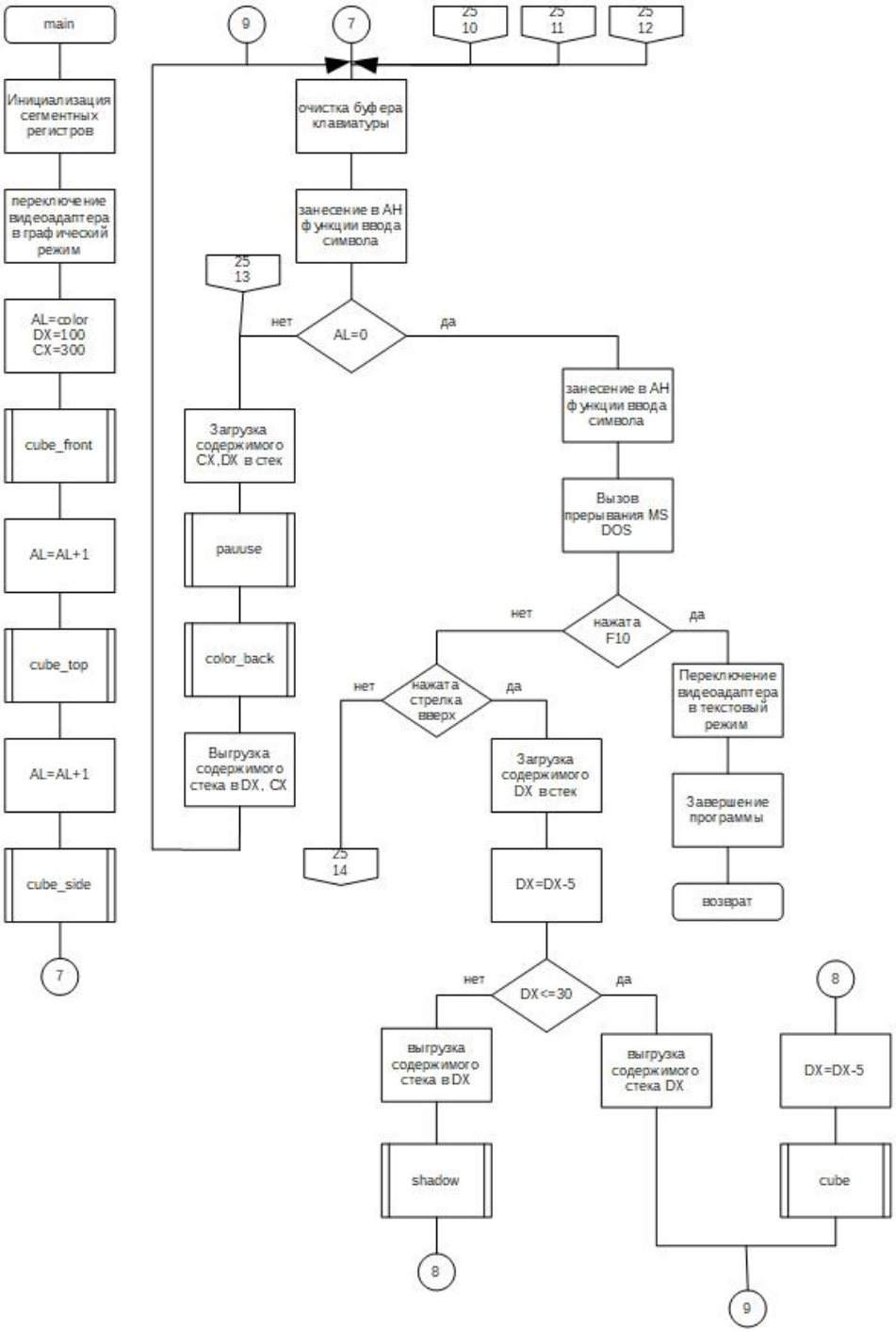


Рисунок 12 Главная процедура

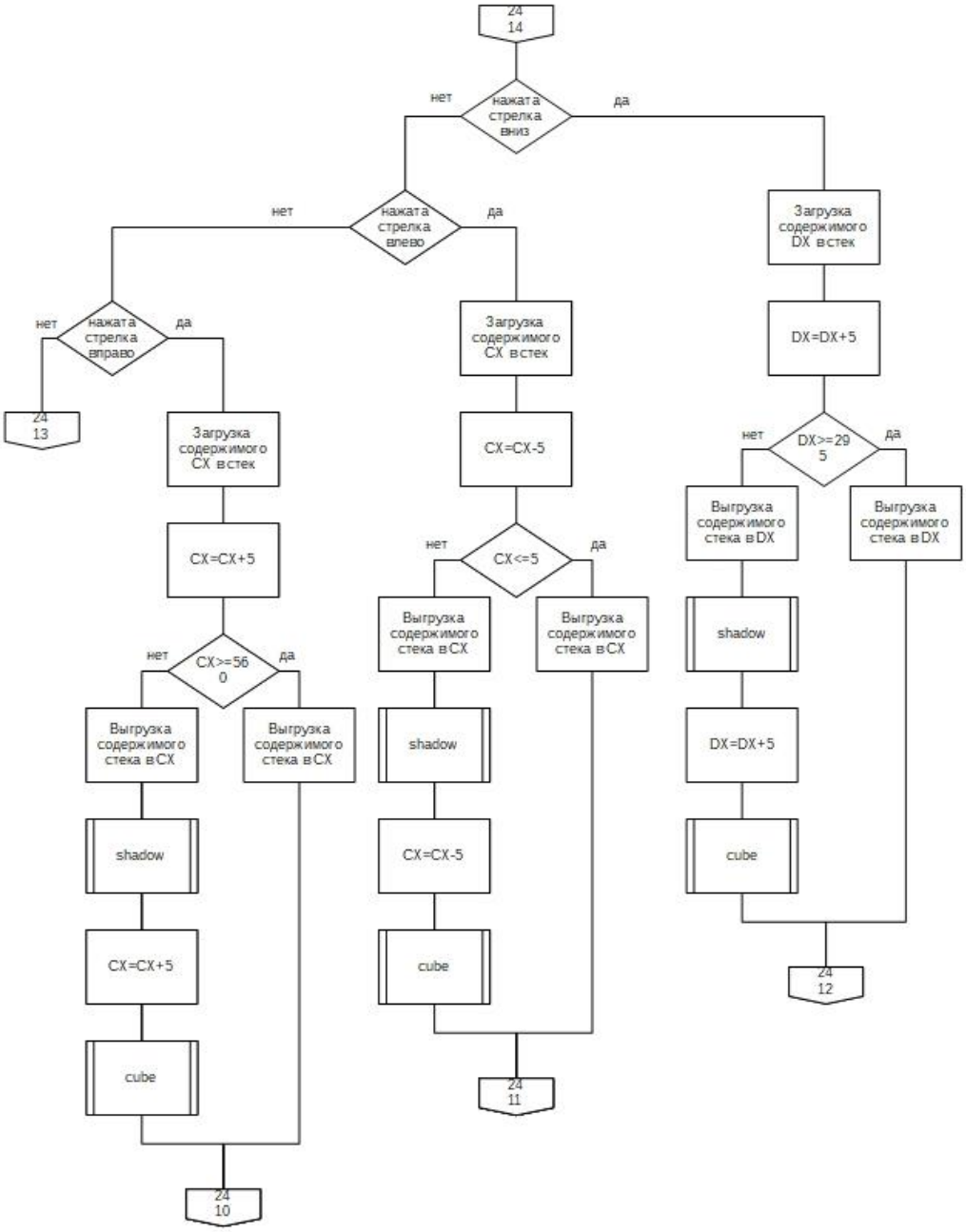


Рисунок 13 Главная процедура

Рисунок 14 Основная программа

3.3. Назначение процедур и ячеек данных

Ячейки данных:

- color — хранение кода цвета;
- color_temp — вспомогательная переменная для хранения исходного кода цвета.

Процедуры:

- delay — осуществление задержки 0.5 сек;
- pauuse — проверка нажатия клавиши во время смены цвета куба и остановка смены цвета, если не нажата любая клавиша в течении 7 сек;
- color_back — возврат исходного цвета кубика;
- color_black — установка чёрного цвета;
- color_changing — изменение цвета куба;
- cube – отрисовка куба;
- cube_side — отрисовка боковой грани куба;
- cube_top — отрисовка верхней грани куба;

- cube_front — отрисовка передней грани куба;
- shadow — отрисовка «тени» куба (закрашивание куба в чёрный цвет);
- main — главная процедура.

3.4. Описание процедур

- Процедура delay (Рисунок 2, стр.16)

Происходит сохранение используемых регистров в стек. Это значение и заносится в регистр CX. Происходит сохранение в регистре BX значение на единицу больше, чем в DX, затем эти значения сравниваются. Всё это происходит в цикле. Как только значение DX изменится (прошло 18 сек) происходит выход из цикла. Данный цикл заключён во внешний цикл zd. И поскольку внешний цикл задержка примерно равна 1 сек. Нам необходима задержка в 0.5 с, значит внешний цикл должен выполниться 9 раз. Параметром, задающим количество повторений внешнего цикла, является значение в регистре CX.

- Процедура pauuse (Рисунок 3, стр.17)

Происходит отрисовка куба и задержка в 0.5 с. После этого проверяется нажатие клавиши. Если клавиша нажата, то происходит выход из процедуры (остановка смены цвета). В противном случае смена цвета продолжается. Регистр BP хранит количество раз смены цвета. Т.к. задержка 0.5 с, то для задержки в 7 сек. в BP необходимо занести значение в 2 раза больше.

- Процедура color_back (Рисунок 4, стр.18)

В ячейке color_temp хранится исходный цвет, а в color – текущий цвет. Процедура устанавливает исходный цвет, путём занесения в color значения color_temp. Напрямую это сделать нельзя, поэтому обмен происходит через регистр AL.

- Процедура color_black (Рисунок 5, стр.18)

В ячейке color_temp хранится исходный цвет, а в color – текущий цвет. Для начала необходимо сохранить изменяемый цвет, для этого в color_temp заносится содержимое color. Напрямую это сделать нельзя, поэтому обмен происходит через регистр AL. Далее в color заносится значение 0, что соответствует коду чёрного цвета.

- Процедура color_changing (Рисунок 6, стр.18)
- Процедура cube (Рисунок 7, стр.19)

В ячейке color_temp хранится исходный цвет, а в color – текущий цвет. Для начала необходимо сохранить изменяемый цвет, для этого в color_temp заносится содержимое color. Напрямую это сделать нельзя, поэтому обмен происходит через регистр AL. Затем поочерёдно вызываются процедуры построения куба и изменения цвета для каждой отдельной грани. После отрисовки в ячейку color заносится цвет, с которого начиналось построение. Обмен снова происходит через регистр AL.

- Процедура cube_side (Рисунок 8, стр.20)

В регистре CX хранится координата куба по оси X, а в регистре DX – координата по оси Y (координата куба — левый верхний угол передней грани). Перед началом работы процедуры они сохраняются в стеке. Далее происходит задание другой точки — правого нижнего угла передней грани куба. В SI заносится ширина боковой стороны. Прорисовка происходит снизу вверх, путём построения прямых вертикальных линий, каждый раз со смещением на 1 пиксель. Как только значение SI стало равным 0, значит прорисовка завершена. Происходит загрузка начальных координат из стека.

- Процедура cube_top (Рисунок 9, стр.21)

В регистр SI заносится высота верхней грани. Также происходит сохранение в стек регистра DX, в котором хранится координата куба по оси Y. Регистр BL хранит длину линии. Прорисовка происходит слева направо, путем построения

прямых горизонтальных линий. При отрисовке линии происходит проверка равен ли BL нулю. Если да — линия нарисована полностью, следует переходить на другую. В этот момент происходит проверка равенства нулю значения в регистре SI. Если равен — прорисовка завершена, происходит восстановление начальных координат куба (левый верхний угол передней грани). В противном случае значение регистра SI декрементируется и происходит повторение построения линии со смещением линии вверх и вправо на 1 пиксель.

- Процедура cube_front (Рисунок 10, стр.22)

Происходит инициализация графического режима и установка цвета кубика. В регистр SI заносится высота передней грани, а регистр BL – ширина. Сохраняется значение координат куба (левый верхний угол передней грани) в стек. Прорисовка происходит слева направо, путем построения прямых горизонтальных линий. При отрисовке линии происходит проверка равен ли BL нулю. Если да — линия нарисована полностью, следует переходить на другую. В этот момент происходит проверка равенства нулю значения в регистре SI. Если равен — прорисовка завершена, происходит восстановление начальных координат куба (левый верхний угол передней грани). В противном случае значение регистра SI декрементируется и происходит повторение построения линии со смещением линии вниз на 1 пиксель.

- Процедура shadow (Рисунок 11, стр.23)

Происходит отрисовка кубика черным цветом. Используется во время передвижения. Устанавливается чёрный цвет и поочерёдно вызываются процедуры отрисовки граней куба. В конце восстанавливается значение цвета, которое было до начала отрисовки.

- Главная процедура (Рисунок 12 - Рисунок 13, стр.24-25)

Инициализация графического режима, установка начального цвета и отрисовка кубика. Затем происходит очистка буфера клавиатуры и анализ нажатой клавиши.

При нажатии клавиши F10 происходит переключение видеоадаптера в текстовый режим и выход из программы. При нажатии стрелочек — передвижение кубика в соответствующем направлении — закрашивание его в чёрный цвет, затем изменении соответствующей координаты (в зависимости от направления движения) и вновь прорисовка цветом. Параллельно с этим происходит проверка выхода кубика за границы. При достижении границы окна отрисовка прекращается, т. е. выйти за экран невозможно. При нажатии любых других клавиш осуществляется смена цвета кубика, путем вызова соответствующих для этого процедур.

4. Тестирование и отладка

Во время тестирования и отладки были обнаружены и исправлены следующие ошибки:

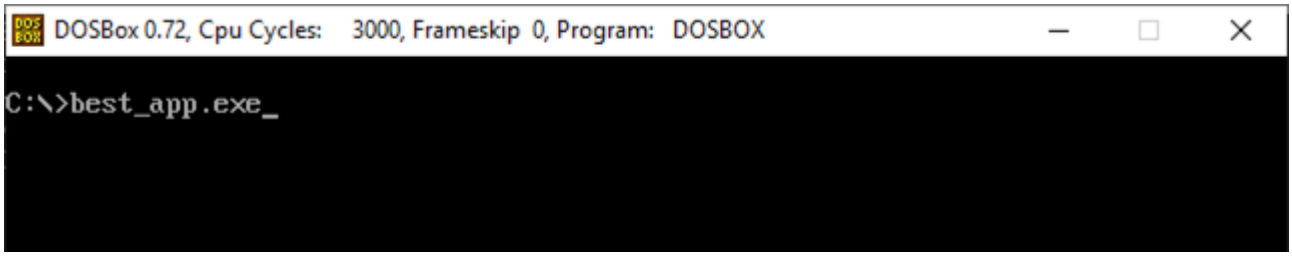
- Происходил «вылет» куба за границы экрана. Решение — проверка достижения границы экрана.
- При смене цвета грани куба закрашивались в чёрный цвет и сливались с фоном. Решение — проверка выбора цвета перед отрисовкой.
- Во время изменения цвета при нажатии клавиши остановка происходила не сразу, а после следующего изменения цвета. Решение — опрос нажатия клавиши 2 раза за одно изменение цвета.

5. Руководство пользователя

Программа best_app.exe позволяет познакомиться с возможностями языка программирования Ассемлер, а также занятно провести свободное время.

1. Запуск программы

Запуск производится в среде MS DOS или же в эмуляторе DOSBox. (Дистрибутив данного эмулятора можно загрузить с официального сайта: http://dosbox.com или использовать эмулятора dosbox.exe, поставляемый вместе с программой).



Для запуска исполняемого файла нужно в командную строку вышеописанных сред ввести: best_app.exe (Рисунок 15).

2. Перемещение куба по экрану

Для перемещения куба по экрану используются клавиши:

- СТРЕЛКА ВВЕРХ — перемещение куба вверх
- СТРЕЛКА ВНИЗ — перемещение куба вниз
- СТРЕЛКА ВЛЕВО — перемещение куба влево
- СТРЕЛКА ВПРАВО — перемещение куба вправо

!Обратите внимание, что при достижении фигурой какой-либо границы экрана дальнейшее перемещение в соответствующем направлении останавливается, т. е. выйти за границы экрана не представляется возможным! Также осуществлять перемещение во время светомузыки нельзя. Для осуществления перемещения необходимо либо дождаться окончания смены цветов, либо принудительно завершить данный процесс (путём нажатия любой клавиши)!

3. Изменение цветов куба

Для начала изменения цветов необходимо нажать любую клавишу, за исключением F10, СТРЕЛКА ВВЕРХ, СТРЕЛКА ВНИЗ, СТРЕЛКА ВЛЕВО, СТРЕЛКА ВПРАВО.

Для принудительной остановки смены цветов необходимо нажать любую клавишу, включая F10, СТРЕЛКА ВВЕРХ, СТРЕЛКА ВНИЗ, СТРЕЛКА ВЛЕВО, СТРЕЛКА ВПРАВО. При отсутствии нажатия клавиш в течении 7 сек. с начала момента запуска светомузыки смена цветов автоматически прекратится.

!Обратите внимание, что во время смены цветов выйти из программы или осуществлять перемещение кубика нельзя. Для осуществления данных действий необходимо либо дождаться окончания смены цветов, либо принудительно завершить данный процесс (путём нажатия любой клавиши)!

4. Завершение работы программы

Чтобы остановить выполнение программы, нажмите клавишу F10.

!Обратите внимание, что при изменении цвета остановка выполнения программы сразу же невозможна. Можно либо дождаться окончания изменения цвета и нажать клавишу F10, либо принудительно остановить смену цветов (путём нажатия любой клавиши) и затем нажать клавишу F10!

Чрезмерное времяпровождение за компьютером вредит Вашему здоровью! Будьте аккуратны!

Приятного использования!

Заключение

В ходе выполнения данной курсовой работы была написана программа, осуществляющая перемещение куба на экране монитора и изменение его цвета через заданный интервал времени. Можно отметить, что в ходе выполнения курсовой работы был более глубоко изучен язык программирования низкого уровня Ассемблер, а что самое главное — появилось представление о том, как на самом деле происходит работа компьютера.

Несмотря на некоторые сложности, которые возникли при написании данной программы, было создано программное средство, которое удовлетворяет всем заданным требованиям и которое позволяет скоротать досуг пользователя.

Список литературы

1. Assembler [Электронный ресурс]. URL: http://progopedia.ru/language/assembler/ (дата обращения: 18.05.2019)
2. Основные группы команд ассемблера. [Электронный ресурс]. URL: https://studfiles.net/preview/4676343/ (дата обращения: 18.05.2019)
3. Команды пересылки [Электронный ресурс]. URL: https://indigobits.com/assembler/13-komandy-peresylki.html (дата обращения:
4. Арифметические команды языка Ассемблер: аддитивные и мультипликативные команды целочисленных операций [Электронный ресурс]. URL: https://www.sites.google.com/site/sistprogr/lekcii1/lek7 (дата обращения: 18.05.2019)
5. Архитектура процессора 8086 [Электронный ресурс]. URL: %82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0+%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80%D0%B0+8086b/main.html (дата обращения: 18.05.2019)
6. Команды безусловного и условного переходов в языке Ассемблер [Электронный ресурс]. URL: https://www.sites.google.com/site/sistprogrlekcii1/lek9 (дата обращения:
7. Прерывание. Классификация. Команда INT [Электронный ресурс]. URL: http://indigobits.com/assembler/26-preryvanie-klasifikaciya-komanda-int.html (дата обращения: 18.05.2019)

Приложение 1. Исходный код

```
t
e       assume cs:text, ds:data
x
tdelay proc
s       push cx
e       push bx
g       push bp
m       push ax
e       push dx
n       mov cx, 9 ; delay 0.5s
t       zd:
c               push cx
o               mov bp, 0
d               mov bx, 1
e               cikl1:
                        inc bp
                        mov ah, 00h ; time function
                        int 1Ah
                        cmp bp, 1
                        je ii
                        jmp ii1
                ii:
                        add bx, dx ; bx>dx
                ii1:
                        cmp bx, dx ; was "tik"?
```

```
                jne cikl1
                pop cx
        loop zd
        pop cx
        pop bx
        pop bp
        pop ax
         pop dx
        ret
delay endp

pauuse proc
        mov bp,1
        mig:
                push cx
                push dx
                call color_changing
                call cube
                call delay
                pop dx
                pop cx
                inc bp
                cmp bp,15 ; 7 s
                je ex ;
                mov ah, 06h ; function for input symbol without waiting
                push dx
                mov dl, 0FFh ; function for input symbol without waiting
```

```
            int 21h
            cmp al, 0 ; if al<>0 than pressed any key
            pop dx
            je mig
            jmp ex
        ex:
        ret
pauuse endp

color_back proc
        mov al, color_temp ; saving current color
        mov color, al ; saving current color
        ret
color_back endp

color_black proc
        mov al, color ; saving current color
        mov color_temp, al ; saving current color
        mov color, 0 ; color = black
        ret
color_black endp

color_changing proc
        inc color ; next color
        cmp color, 16 ; if colors are over than start again from 1(blue color)
        jne f ; if not that set this new color
        mov color, 1
```

```
        f:
                mov al, color
                ret
color_changing endp

cube proc
        mov al, color ; saving current color
        mov color_temp, al ; saving current color
        call cube_front
        call color_changing
        call cube_top
        call color_changing
        call cube_side
        call color_back
        ret
cube endp

cube_side proc
        push cx ; save X coordinate (top left corner of the front)
        push dx ; save Y coordinate (top left corner of the front)
        add cx, 50 ; set new X coordinate (bottom right corner of the front)
        add dx, 50 ; set new Y coordinate (bottom right corner of the front)
        mov si, 25 ; width of the side
        side_width:
                push dx ; save staring point's Y coordinate
                mov bl,51 ; length of the line
                side_line:
```

```
                int 10h ; put pixel
                dec dx ; decrease Y coordinate
                dec bl ; decrese length
                cmp bl, 0 ; if not equal to zero
            jne side_line ; repeat
            cmp si, 0 ; else if draw full side
            je side_exit ;than exit
            dec si ; else decrese width of the side
            pop dx ; load Y coordinate from stack
            dec dx ; decrease Y coordinate
            inc cx ; increase X coordinate
        jmp side_width ; start over
        side_exit:
            pop dx ; extra stack value
            pop dx ;return Y coordinate (top left corner of the front)
            pop cx ;return X coordinate (top left corner of the front)
            ret
cube_side endp

cube_top proc
        mov si, 25 ; width of the top
        push dx ; save Y coordinate (top left corner of the front)
        dec dx ; decrese Y coordinate
        top_width:
            push cx ; save X coordinate (line's start point)
            mov bl,51 ; length of the line
            top_line:
```

```
                int 10h ; put pixel
                inc cx ; increase X coordinate
                dec bl ; decrease line length
                cmp bl, 0 ; if line's length  not equal to zero
            jne top_line ; repeat
            cmp si, 0 ; if drew full top
            je top_exit ; than exit
            dec si ; else descrease top's width
            pop cx ; save X coordinate to the stack
            dec dx ; decrease Y coordinate
            inc cx ; increase X coordinate
        jmp top_width ; repeat
        top_exit:
            pop cx ; load last X coordinate from the stack
            add cx, -25 ; X coordinate (top left corner of the front)
            pop dx ; load Y coordinate (top left corner of the front)
            ret
cube_top endp

cube_front proc
        mov ah, 0ch ; function to pixel output
        mov bh, 0 ;videopage
        mov al, color ; set color
        mov si, 50 ; front's height
        push dx ; save Y coordinate to the stack (top left corner of the front)
        front_width:
            push cx ; save X coordinate to the stack
```

```
            mov bl,50 ; length of the front
            front_line:
                int 10h ; put pixel
                inc cx ; increase X coordinate
                dec bl ; decrese front's length
                cmp bl, 0 ; if front's length not equal to zero
            jne front_line ; that repeat drawing again
            cmp si, 0 ; else if front's height equal to zero
            je front_exit ; that exit
            dec si ; else decrease front's height
            pop cx ; save X coordinate to stack
            inc dx ; increase Y coordinate
        jmp front_width ; repeat drawing again
        front_exit:
            pop cx ; load X coordinate (top left corner of the front)
            pop dx ; load Y coordinate (top left corner of the front)
            ret
cube_front endp

shadow proc
    call color_black ; set black color
    call cube_front ; remove
    call cube_top ; cube
    call cube_side ; by draving it by
    call color_back ; black color
    ret
shadow endp
```

```
main proc
        mov ax, data ; initialization of
        mov ds,ax ; the system registers
        mov ah, 00h ; function to set mode
        mov al, 10h ;graphic mode EGA
        int 10h
        mov al, color ; set color
        mov dx, 100 ; point y coordinate
        mov cx, 300 ; point x coordinate
        call cube_front
        inc al
        call cube_top
        inc al
        call cube_side
        input:
                mov ah, 0Ch ; clear
                mov al, 08h ; keyboard
                int 21h ; buffer
                cmp al, 0 ; pressed functional key?
                je analize
                jmp change_color
         analize:
                 mov ah, 08h ; a function to input key
                 int 21h
                 cmp al, 44h ; pressed F10?
                 jne further
```

```
        jmp exit
further:
        cmp al, 72 ; pressed key up?
        je up
        cmp al, 80 ; pressed key down?
        je down
        cmp al, 75 ; pressed key left?
        jne left_jump
        jmp left
left_jump:
        cmp al, 77 ; pressed key right?
        jne change_color
        jmp right
        change_color:
                push cx
                push dx
                call pauuse ; star changing colors
                call color_back ; set "start" color
                pop dx
                pop cx
                jmp input
        up:
                push dx
                add dx, -5 ; decrese Y coordinate of the cube
                cmp dx, 30 ; reached the top border?
                jle point_up_stop ; if yes than stop moving
                pop dx
```

```
        call shadow ; "clear" cube
        add dx, -5 ; set new Y coordinate
        call cube ; draw the cube in new place
        jmp input ; again to input
        point_up_stop:
            pop dx
            jmp input ; again to input
down:
        push dx
        add dx, 5 ; increase Y coordinate
        cmp dx, 295 ; reached the bottom border?
        jge point_down_stop ; if yes than stop moving
        pop dx
        call shadow ; "clear" cube
        add dx, 5 ; set new Y coordinate
        call cube ; draw the cube in new place
        jmp input ; again to input
        point_down_stop:
            pop dx
            jmp input ; again to input
right:
        push cx
        add cx, 5 ; increase X coordinate
        cmp cx, 560 ; reached the right border?
        jge point_right_stop ; if yes than stop moving
        pop cx
        call shadow ; "clear" cube
```

```
                add cx, 5 ; set new X coordinate
                call cube ; draw the cube in new place
                jmp input ; again to input
                point_right_stop:
                        pop cx
                jmp input ; again to input
        left:
                push cx
                add cx, -5  ; decrease X coordinate
                cmp cx, 5 ; reached the left border?
                jle point_left_stop ; if yes than stop moving
                pop cx
                call shadow ; "clear" cube
                add cx, -5 ; set new X coordinate
                call cube ; draw the cube in new place
                jmp input ; again to input
                point_left_stop:
                        pop cx
                jmp input ; again to input
    exit:
            mov ah, 00h ; function to set mode
            mov al, 03h ; text mode
            int 10h
            mov ax, 4c00h
            int 21h
main endp
text ends
```

```
data segment 'data'
        color db 1
        color_temp db ?
data ends
end main
```