**АННОТАЦИЯ**

Данная курсовая работа представляет собой приложение, обеспечивающее генерацию звука с помощью системного таймера.

Так как в настоящее время воспроизведение звука происходит посредством звуковых плат, обычный динамик PC практически никем не используется, поэтому главное его применение – звуковое сопровождение при выдаче сообщений об ошибках.

1. ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ

1.1 Исследование предметной области и постановка задачи

Целью данной курсовой работы является разработка приложения, которое может генерировать звук, используя системный таймер.

Для достижения данной цели необходимо реализовать ряд задач, а именно:

* изучить принцип работы системного таймера
* разработать алгоритмы для перехвата и обработки прерываний системного таймера
* создать процедуру для извлечения очередного звука
* создать процедуру для обработки символов, введенных с клавиатуры

1.2 Общие сведения о языке

Язык ассемблера - это символическое представление машинного языка. Все процессы в машине на самом низком, аппаратном уровне приводятся в действие только командами (инструкциями) машинного языка. Отсюда понятно, что, несмотря на общее название, язык ассемблера для каждого типа компьютера свой. Это касается и внешнего вида программ, написанных на ассемблере, и идей, отражением которых этот язык является.

Язык ассемблера был создан для облегчения написания программ на низком уровне. Вместо машинных команд программист использует мнемоники, которые представляют собой символическое представление машинной команды (или нескольких команд), что облегчает чтение текста программы. Так как одна мнемоника при трансляции преобразовывается в минимальное количество команд, то программы, написанные на ассемблере, выполняются с максимально возможной скоростью и занимают минимальный объём памяти. Именно поэтому программы на ассемблере (или ассемблерные вставки) используются там, где быстродействие является наиболее критичным фактором (обработка компьютерной графики, ядро операционных систем), или требуется максимально полно использовать возможности операционной системы (вирусы и антивирусы), или там, где эти два качества являются одинаково важными (драйверы различных устройств). Хорошим качеством Ассемблера является то, что он позволяет управлять действиями процессора поэтапно (по операциям) и с максимальной эффективностью. Ассемблер предоставляет программисту полную свободу действий при разработке программы, что одновременно является и его достоинством, и недостатком, так как требует от разработчика знания системы команд данного компьютера и его операционной системы. Данный язык позволяет программисту непосредственно работать со всем аппаратным обеспечением, что особенно важно при программировании устройств - ввода-вывода, где требуется контроль над отдельными разрядами регистров устройства. Ассемблер предоставляет возможность создания компактных и быстродействующих программ, оптимизации работы с аппаратными средствами. Синтаксис инструкций процессора предоставляет программисту широкий выбор типов данных: целые числа, строки знаков, упакованные десятичные числа, числа с плавающей точкой, структуры и записи. В языке имеется возможность подключения фрагментов исходного текста из других файлов. Ассемблер имеет набор макро-директив, использование которых позволяет относительно просто повторять общие блоки предложений несколько раз или заменять макро-имена в исходном тексте целыми последовательностями предложений.

1.3 Элементы языка

Для написания курсовой работы были использованы следующие средства языка:

1. Команды пересылки данных
   * Команда *MOV* приемник, источник

Назначение: пересылка данных между регистрами или регистрами и памятью.

* + Команда *XCHG* операнд\_1,операнд\_2

Назначение: обмен двух значений между регистрами или между регистрами и памятью. Команду xchg можно использовать для выполнения операции обмена двух операндов с целью изменения порядка следования байт, слов, двойных слов или их временного сохранения в регистре или памяти.

* + Команда *LEA* приемник, источник

Назначение: получение эффективного адреса (смещения) источника. Алгоритм работы команды зависит от действующего режима адресации (use16 или use32):

если use16, то в регистр приемник загружается 16-битное значение смещения операнда источник;

если use32, то в регистр приемник загружается 32-битное значение смещения операнда источник.

* + Команда *IN* аккумулятор,ном\_порта

Назначение: ввод значения из порта ввода-вывода. Передает байт, слово, двойное слово из порта ввода-вывода в один из регистров al/ax/eax

* + Команда *OUT* ном\_порта,аккумулятор

Назначение: вывод значения в порт ввода-вывода. Передать байт, слово, двойное слово из регистра al/ax/eax в порт, номер которого определяется первым операндом.

* + Команда *PUSHA*

Назначение: размещение в стеке регистров общего назначения в следующей последовательности: ax, cx, dx, bx, sp, bp, si, di. Уменьшает значение указателя стека esp/sp на 32/16 (в зависимости от значения атрибута размера адреса - use16 или use32), включиет в стек последовательно значения регистров общего назначения ax, cx, dx, bx, sp, bp, si, di. Содержимое di при этом будет на вершине стека. В стек помещается содержимое sp по состоянию до выполнения команды.

* + Команда *POPA*

Назначение: извлечение из стека регистров общего назначения di, si, bp, sp, bx, dx, cx, ax. Извлекает из стека последовательно значения и загрузжает ими регистры общего назначения di, si, bp, sp, bx, dx, cx, ax. Содержимое di восстанавливается первым. Содержимое sp извлекается, но не восстанавливается. Увеличивает значение указателя стека esp/sp на 16.

1. Арифметические команды
   * Команда *INC* операнд

Назначение: увеличение значения операнда в памяти или регистре на единицу.

* Команда *DEC* операнд

Назначение: уменьшение значения операнда в памяти или регистре на единицу.

1. Команды логической обработки данных

* Команда *AND* приемник, источник

Назначение: операция логического умножения для операндов приемник и источник размерностью байт, слово или двойное слово. Выполняет операцию логического умножения над операндами источник и приемник: каждый бит результата равен 1, если соответствующие биты операндов равны 1, в остальных случаях бит результата равен 0.Результат операции записывается в приемник. Устанавливаются флаги.

* Команда *OR* приемник, маска

Назначение: операция логического ИЛИ над битами операнда назначения, используя в качестве маски второй операнд - маска. При этом бит результата равен 0, если соответствующие биты операндов маска и назначения равны 0, в противном случае бит равен 1. Результат операции записывается в источник (маска неизменна). Устанавливаются флаги.

* Команда *XOR* приемник, источник

Назначение: операция логического исключающего ИЛИ над двумя операндами размерностью байт, слово или двойное слово. Бит результата равен 1, если значения соответствующих битов операндов различны, в остальных случаях бит результата равен 0. Результат операции записывается в приемник. Устанавливаются флаги.

* Команда *TEST* приемник, источник

Назначение: операция логического сравнения операндов приемник и источник размерностью байт, слово или двойное слово, при этом бит результата равен 1, если соответствующие биты операндов равны 1, в остальных случаях бит результата равен 0. Устанавливаются флаги.

* Команда *SHL* операнд, количество\_сдвигов

Назначение: логический сдвиг операнда влево. Происходит сдвиг всех битов операнда влево на один разряд, при этом выдвигаемый слева бит становится значением флага переноса cf, одновременно слева в операнд вдвигается нулевой бит. Указанные выше два действия повторяются количество раз, равное значению второго операнда.

1. Команды передачи управления

* Команда *CMP* операнд1,операнд2

Назначение: сравнение двух операндов. Выполняет вычитание (операнд1-операнд2). В зависимости от результата устанавливает флаги, операнд1 и операнд2 не изменяются

* Команда *JMP* метка

Назначение: используется в программе для организации безусловного перехода как внутри текущего сегмента команд, так и за его пределы. При определенных условиях в защищенном режиме работы команда jmp может использоваться для переключения задач.

1. Команда *CALL* цель

Назначение: передача управления близкой или дальней процедуре с запоминанием в стеке адреса точки возврата и переключение задач.

Команда call передает управление внешней процедуре, предварительно сохранив в стеке информацию для последующего возврата в вызывающую процедуру при помощи команды ret. Команда call имеет различную форму записи в зависимости от типа вызываемой процедуры (дальняя или ближняя). Команда ret, которой завершается вызываемая процедура, должна иметь тот же тип (дальний или ближний), что и вызывающая процедуру команда call. Адрес вызываемой процедуры может быть задан непосредственно в команде call, в памяти или в регистре.

1. Команда *RET*

Назначение: возврат управления из процедуры вызывающей программе.

Команда ret возвращает управление из вызванной процедуры команде, следующей за командой call. Если возврат осуществляется из ближней процедуры, возврат является внутрисегментным (содержимое регистра CS остается неизменным). При возврате из дальней процедуры возврат является межсегментным (из стека восстанавливаются значения CS и IP). Если в команде задано необязательное значение, команда ret добавляет это значение к указателю стека SP. Это позволяет пропускать параметры, передаваемые через стек перед командой call.

1. Команда *IRET*

Назначение: используется в той точке программы обработки прерывания, откуда необходимо вернуть управление прерванной программе. Команду iret необходимо применять для восстановления сохраненных командой int регистров флагов, указателя команд и сегментного регистра кода. Число этих команд в программе обработки прерывания должно соответствовать количеству точек выхода из нее.

1. Команда прерываний *INT* число

Назначение: Вызов прерывания

INT помещает в стек содержимое регистров EFLAGS, CS и EIP, после чего передает управление программе, называемой «обработчик прерывания» с указанным в качестве операнда номером (число от 0 до 0FFh), аналогично команде call. В реальном режиме адреса обработчиков прерываний считываются из таблицы, начинающейся в памяти по адресу 0000h:0000h. Адрес каждого обработчика занимает 4 байта, так что, например, адрес обработчика прерывания 10h находится в памяти по адресу 0000h:0040h. В защищенном режиме адреса обработчиков прерываний находятся в таблице IDT и обычно недоступны для прямого чтения или записи, так что для установки собственного обработчика программа должна обращаться к операционной системе. В DOS вызовы прерываний используются для выполнения большинства системных функций - работы с файлами, вводом/выводом и т.д.

1. Процедуры

Синтаксис описания процедуры:

имя\_процедуры PROC [модификатор языка] [расстояние]

…

[имя\_процедуры] ENDP

Процедура, часто называемая также подпрограммой, - 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 уровня и возврата управления в эту точку. В простейшем случае программа может состоять из одной процедуры. Другими словами, процедуру можно определить как правильным образом оформленную совокупность команд, которая, будучи однократно описана, при необходимости может быть вызвана в любом месте программы.

Среди большого количества операндов директивы PROC следует особо выделить [расстояние]. Этот атрибут может принимать значения near или far и характеризует возможность обращения к процедуре из другого сегмента кода. По умолчанию атрибут [расстояние] 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 и соответственно будет осуществлять выполнение команд процедуры.

1. Системный таймер

Таймеру соответствуют четыре порта ввода/вывода со следующими адресами:

* 40h - канал 0;
* 41h - канал 1;
* 42h - канал 2;
* 43h - управляющий регистр.

Одно из наиболее распространенных применений таймера - генерация звуковых сигналов и воспроизведение музыки. Таймер позволяет воспроизводить музыку в фоновом режиме, т.е. во время работы программы может звучать музыка.

Как мы уже говорили, канал 2 микросхемы 8254 связан с громкоговорителем компьютера. Однако громкоговоритель не просто соединен с выходом OUT канала 2. Порт вывода 61h также используется для управления громкоговорителем. Младший бит порта 61h подключен ко входу GATE канала 2 таймера. Этот бит при установке в 1 разрешает работу канала, т.е. генерацию импульсов для громкоговорителя.

Дополнительно для управления громкоговорителем используется бит 1 порта 61h. Если этот бит установлен в 1, импульсы от канала 2 таймера смогут проходить на громкоговоритель.

Таким образом, для включения звука надо выполнить следующие действия:

* запрограммировать канал 2 таймера на нужную частоту (т.е. загрузить регистр счетчика канала нужным значением);
* для включения звука установить в 1 два младших бита порта 61h.

Так как остальные 6 битов порта 61h используются для других целей, установка младших битов должна выполняться таким образом, чтобы значения остальных битов не были изменены. Для этого вначале надо считать байт из порта 61h в рабочую ячейку памяти, установить там нужные биты, затем вывести новое значение байта в порт 61h.

Очевидно, что для выключения звука надо сбросить два младших бита порта 61h в 0 (при этом нельзя изменять значение остальных битов этого порта).

Мелодия (одноголосая), как известно, состоит из нот, разделенных или не разделенных паузами. При проигрывании мелодии необходимо для каждой ноты программировать соответствующим образом канал 2 таймера и включать громкоговоритель (с помощью порта 61h) на определенное время, равное длительности ноты. Затем программа должна выключить динамик и выдержать паузу перед проигрыванием следующей ноты, если такая пауза требуется.

Основная идея заключается в использовании прерывания 1Ch, которое вырабатывается таймером с частотой примерно 18,2 Гц. Пользовательский обработчик этого прерывания осуществляет контроль за выборкой нот из массива, содержащего мелодию, и программирование микросхемы 8254. Если подготовить таблицы частот и длительностей, то можно проигрывать простейшие мелодии.

Для определения значения, которое должно быть записано в регистр счетчика канала 2 таймера, надо разделить 1193180 на требуемую частоту в герцах.

В таблице ниже приведены частоты нот для второй октавы. При повышении (понижении) тона на октаву частота соответствующей ноты умножается (делится) на два.

Таблица 1 - Частоты нот

|  |  |
| --- | --- |
| **Нота** | **Частота, Гц** |
| До | 261,7 |
| До-диез | 277,2 |
| Ре | 293,7 |
| Ре-диез | 311,1 |
| Ми | 329,6 |
| Фа | 349,2 |
| Фа-диез | 370 |
| Соль | 392 |
| Соль-диез | 415,3 |
| Ля | 440 |
| Ля-диез | 466,2 |
| Си | 493,9 |

2. КОНСТРУКТОРСКАЯ ЧАСТЬ

2.1 Общие сведения

Данная программа написана на ассемблере языке программирования низкого уровня, создана с использованием программных средств пакета Tasm и редактора AsmEdit. Исполняемый файл программы находится в формате.exe и называется kur.exe.

2.2 Функциональное назначение

Данная программа позволяет пользователю выбрать одну из двух представленных мелодий для генерации и воспроизведения.

2.3 Описание логической структуры программы

2.3.1 Алгоритм работы программы

- Вывод на экран строки

- Сохранение старого вектора прерывания

- Установка нового вектора прерывания

- Опрос клавиатуры

- Если выбран номер мелодии, то поместить массив в аккумулятор

- Иначе: восстановление обработчика прерывания и выход

2.3.2 Используемые методы

В начале программы происходит инициализация сегментного регистра CS. Все данные, которые описаны в программе, размещаются в регистре команд. После запуска программы на экран выводится сообщение. Происходит сохранение старого вектора прерывания и инициализации нового. В зависимости от выбора пользователя, после опроса клавиатуры, программа продолжает работу и передает текущую ноту в регистр-аккумулятор, либо происходит восстановление прерывания и выход их программы. Выход из программы происходит при нажатии кнопки “q”. Основная процедура программы - *Start proc (см.Приложение 1, стр. 22).*

Ниже описаны подпрограммы, которые вызывают в главной:

- *new\_int1c proc near* – новый обработчик прерываний *(см. Приложение 1, стр.23)*.

Происходит сохранение всех регистров, вызывается процедура генерации звука, восстановление всех регистров и происходит возврат из программы обработки прерывания.

- *kbin proc near* - проверка введенного символа для дальнейшей передачи управления*(см. Приложение 1, стр.24)*.

Происходит вызов клавиатурного прерывания для считывания символа и выход из процедуры.

- *muz proc near* – процедура извлечения очередного звука *(см. Приложение 1, стр.23)*.

Первым делом осуществляется проверка, установлена ли переменная iniflag (она используется как флаг). Если нет, что считывается состоянии системного порта РВ ППИ, в 0 и 1 бит заносятся 0 и записывается обратно в порт. Таким образом происходит запрет звучания.

Если флаг установлен, то считывается текущая нота. Если она равно 255 (то есть это пауза), то происходит запрету звучания и продолжение считывания нот. В противном случае проверяется, если нота равно 0 (то есть конец массива нот), то происходит переход на начало выполнения программы и происходит выход, если флаг не возведен.

Если текущая нота не равно 255 или 0, то происходит считывание частоты ноты, младший и старший байты которой заносятся во второй канал таймера. Далее происходи разрешение звучания, то есть считывается значение системного порта РВ ППИ, в 0 и 1 бат заносятся единицы и отправляется обратного в порт. И происходит нормальный выход из процедуры.

2.3.3 Вызов и загрузка

Вызов программы осуществляется путём запуска файла «kur.exe». Если файл не существует, его необходимо скомпилировать из файла исходного текста программы под названием «kur.asm».

2.3.4 Входные данные

Входными данными программы является ввод пользователем пункта меню.

2.3.5 Выходные данные

Выходными данными программы является звуковой сигнал динамика. Так же на экран выводится простое пользовательское меню.

3. ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ

3.1 Общие сведения

Для создания курсовой работы использовалось следующее программное обеспечение:

* Компилятор TASM
* Компоновщик Tlink.exe
* Отладчик TD.EXE
* Текстовый редактор – ASM Editor
* MS Office Word 2003
* MS Office Visio 2003

3.2 Руководство системного программиста

3.2.1 Системные требования

Минимальные системные требования:

* Процессор 80286 и выше
* Оперативная память – 1 Мб
* Свободного места на жестком диске 1 Мб
* Операционная система DOS 3.3 и выше / Windows 9x/ME
* Клавиатура
* Мышь

3.2.2 Структура программы

Программа представляет собой один исходный файл для выполнения и называется Kur.exe. Она не имеет никаких связей с другими программами и работает автономно.

3.2.3 Тестирование программы

После запуска произойдет размещение программы в памяти. При наборе на клавиатуре 1 произойдет загрузка в регистр-аккумулятор мелодии «Чижик-пыжик», которая будет сгенерирована и воспроизведена через системный динамик. При наборе цифры 2 произойдет загрузка мелодии «Подмосковные вечера», которая будет воспроизводиться аналогично первому случаю. Если будет набрано «q», то произойдет завершение работы программы и выгрузка её из памяти.

3.2.4 Выходные и выходные данные

Входными данными является номер выбранного пункта меню.

Выходные данные – это сгенерированная мелодия в соответствии с выбранным пунктом.

3.3 Руководство оператора

3.3.1 Назначение программы

Данная программа предназначена для генерации звука с использованием системного таймера посредством системного динамика.

3.3.2 Выполнение программы

1. Для запуска программы необходимо запустить приложение Kur.exe
2. Выбрать любой из предложенных пунктов меню.
3. Для генерации мелодии «Чижик-Пыжик» выберите пункт 1
4. Для генерации мелодии «Подмосковные вечера» выберите пункт 2
5. Для выхода из программы выход из программы выберите пункт q

3.3.3 Сообщения оператору

При загрузке программы выводится простое пользовательское меню:

Enter the point of menu:

1- CHIZHIK-PYZHIK

2 - PODMOCKOVNYE VECHERA

q – Qiute

Если пользователь введет пункт меню, который не предусмотрен программой, то на экран будет выведено сообщение.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы было создано приложение, которое способно генерировать мелодии «Чижик-Пыжик» и «Подмосковные вечера».

Для реализации поставленной цели были реализованы следующие задачи:

* Изучены принципы работы системного таймера
* Разработаны алгоритмы для перехвата и обработки прерываний системного таймера
* Создана процедура для извлечения очередного звука
* Создана процедура для обработки символов, введенных с клавиатуры

СПИСОК ЛИТЕРАТУРЫ

* 1. Абель П. Язык Ассемблера для IBM PC и его программирование - М.: Высшая школа,1992. – 252с.
  2. С.В. Зубков Ассемблер для DOS, Windows и Unix – М.: ДМК Пресс, 1999. – 630 с.
  3. И. Юров Assembler – СПб.: Питер, 2001. – 624 с.

ПРИЛОЖЕНИЕ 1

Листинг программы

Assume CS: Code

Code SEGMENT

286

Start proc; Основная процедура

mov ax,cs

mov ds,ax; DS = CS

iniflag db 0; Флаг звучания

old\_int1c\_off dw 0; Смещение старого вектора

old\_int1c\_seg dw 0; Сегмент старого вектора

tek\_mel dw?; Адрес текущей ноты выбранной мелодии

mov ax,351ch

int 21h

mov cs:old\_int1c\_off,bx

mov cs:old\_int1c\_seg,es

lea dx,new\_int1c

mov ax,251ch

int 21h; DS:DX – адрес новой программы обр.

mov ah,09h

lea dx,menu

int 21h

beg1:

call kbin; Опрос клавиатуры

cmp al,'1'

jnz beg2 ; Нет

mov byte ptr iniflag,1; Взведение флага звуч.

lea ax,mel1

mov tek\_mel,ax

jmp beg1; Переход на начало цикла

beg2:

cmp al,'2'

jnz beg3

mov byte ptr iniflag,1

lea ax,mel2

mov tek\_mel,ax

jmp beg1

beg3:

cmp al,'q'

jnz beg4

jmp beg5

beg4:

mov ah,09h

lea dx,er

int 21h

jmp beg1

beg5:

Восстановление старого вектора 1с и выход

mov dx,old\_int1c\_off

mov ax,old\_int1c\_seg

mov ds,ax; DS:DX – адрес устанавл. вектора

mov ax,251ch

int 21h

movax,4c00h

int21h

start endp

Новый обработчик прерывания 1ch

new\_int1c proc far

pusha

call muz; Вызов процедура извлечения звука

popa

iret

new\_int1c endp

Процедура извлечения очередного звука

tek\_mel – адрес текущей ноты выбранной мелодии

muz proc near

test byte ptr cs:iniflag,0ffh; Проверка флага

jnz muz1

muze:

in al,61h ; Чтение состояния системного порта В

and al,0fch; Запрещение звучания (биты 0 и 1)

out 61h,al; Запись в системный порт В

jmp ex ; Выход, если флаг не взведен

muz1:

mov si,cs:tek\_mel; Адрес текущей ноты

mov bl,cs:[si]; Текущая нота

cmp bl,255; Пауза?

jnz muz2

Выключение звука

in al,61h

and al,0fch

out 61h,al

inc cs:tek\_mel; Переход к адресу след. ноты

jmp ex

muz2:

or bl,bl; = 0?

jnz muz3

jmp muze

muz3:

xor bh,bh

mov ax,cs:noty [bx]

Программирование делителя частоты 2 канала

inc cs:tek\_mel; Переход к адресу след. ноты

out 42h,al; Мл.байт частоты. канал 2 таймера

xchg al,ah; AH. AL

out 42h,al; Ст. байт частоты. канал 2 таймера

Разрешение звучания

in al,61h

or al,3; Разрешение звучания (биты 0 и 1)

out 61h,al; Запись в системный порт В

ex:

ret

muz endp

kbin proc near; Ввод с клавиатуры и проверка на выбор игры

mov ah,0

int 16h

ret

kbin endp

Мелодия "Чижик Пыжик"

mel1 db 17,17,255,13,13,255,17,17,255,13,13,255,18,18,255

db 17,17,255,15,15,15,15,255,255

db 8,8,255,8,8,255,8,8,255,10,255,12,255

db 13,13,255,13,13,255,13,13,13,13

db 0

Мелодия "Подмосковные вечера"

mel2 db 1,1,1,4,4,4,8,8,8,4,4,4,6,6,6,6,6,6,4,4,4,3,3,3

db 8,8,8,8,8,8,6,6,6,6,6,6,1,1,1,1,1,1,1,1,1,1,1,1

db 0

menu db 10 dup(0ah), 30 dup(20h), 'Enter the point of menu:'

db 0dh,0ah,30 dup(20h),'1 - CHIZHIK-PYZHIK'.0dh,0ah,30 dup(20h),'2 - PODMOCKOVNYE VECHERA'.0dh,0ah,30 dup(20h), 'q - Quit',5 dup(0ah), "$ "

er db 'Incorrect input! Try again!',0dh,0ah,"$ "

Коэффициенты деления для нот

noty dw 0eeeh,0e18h,0d49h,0c8eh,0bdfh,0b2fh,0abeh

dw 9f7h,968h,8e0h,861h,7e8h,777h,70ch,6a5h,647h

dw 5edh,597h,547h,4fbh,4b4h,470h,430h,3f4h

dw 3bbh,386h,352h,323h,2f6h,2cbh,2a3h,27dh,25ah,238h,218h,1fah

dw 1ddh,1c3h,1a9h,192h,17bh,166h,152h,13fh,12dh,11ch,10ch,0fdh

dw 0

code ends

END

ПРИЛОЖЕНИЕ 2

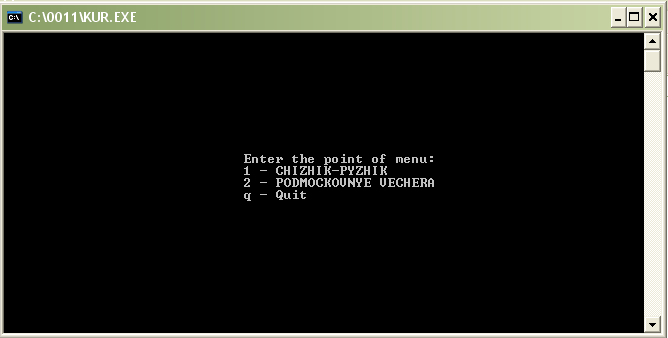


Рис. 1 - Пользовательское меню



Рис. 2 - Некорректный ввод