

Материалы к занятию 1 апреля 2020

Тема: Использование тригонометрических функций, использование команд сопроцессора, прикладная задача (соответствующие программы находятся в папке 20_03_18, а также прилагаются в архиве с этим текстом).

Сначала прямая цитата из пособия

В простейшем случае в IBM PC для генерации звука используется микросхема интегрального таймера 8253 или 8254. Эта микросхема имеет три независимых канала, каждый из которых может программироваться для работы в режиме делителя частоты или генератора одиночных импульсов (рисунок 1). Каждый канал содержит 16-разрядный счетчик, в который записывается значение делителя частоты или коэффициента пересчета (в зависимости от режима работы). Каждый канал имеет вход частоты (clk) и вход разрешения (gate). На вход частоты всех каналов подается импульсный сигнал частотой 1,19 МГц. Канал 0 микросхемы таймера используется для выработки сигнала прерывания по таймеру (частотой 18,2 Гц). Канал 1 работает в режиме генерации одиночных импульсов через каждые 15 мкс. Этот сигнал используется (использовался ранее) для регенерации динамической памяти ЭВМ.

Канал 2 микросхемы исходно программируется для работы в режиме делителя частоты. Выход канала используется для генерации звука через встроенный динамик. Для управления звуком используются биты 0 и 1 системного порта В (микросхема 8255). Бит 0 используется для разрешения прохождения сигнала на выход канала 2 таймера. Сигнал с выхода канала 2 подается на схему "И", на второй вход которой подается сигнал бита 1 системного порта В. Этот сигнал может разрешать или запрещать прохождения сигнала с выхода канала 2 таймера, а при закрытом канале 2 (битом 0 порта В) сигнал бита 1 порта В может использоваться для непосредственной генерации звука в динамике.

Адрес системного порта В – 61h, адреса каналов таймера – 40h, 41h, 42h, 43h – для каналов 0, 1, 2 и управляющего регистра соответственно. Ниже рассмотрены примеры генерации звука с помощью сигнала бита 1 системного порта В, а также с помощью таймера. Рассмотрен случай извлечения звука с использованием прерывания.

Здесь стоит добавить, что при запрещенном канале 2 таймера на его выходе логическая 1 (высокий потенциал), которая разрешает прохождение сигнала со второго входа вентиля **И** на выход (на самом деле там стоит вентиль **И-НЕ**, но в данном случае это не имеет значения). Это позволяет формируя на выходе порта **pb1** широтно-импульсный сигнал, подавать на динамик сигналы совершенно произвольной формы (вплоть до человеческой речи). Когда частоты процессора и периферии были достаточно низкими (порядка единиц или десятка мегагерц, качество такого сигнала было достаточно низким. В современных компьютерах, когда частота работы периферии достигает ста и более мегагерц, можно получить достаточно

разборчивую речь. (Хотя для этого сейчас используют прямую работу с сэмплами через звуковую карту)

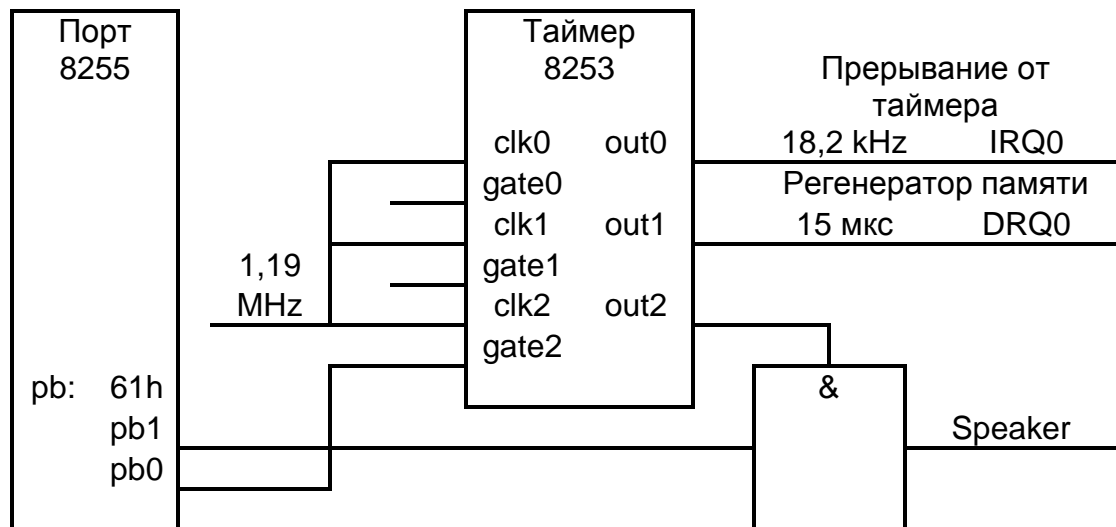


Рисунок 1 – Схема генерации звука в IBM PC

Принцип формирования звука при помощи командного цикла показан на рисунке 2. Для двух звуков различной высоты в программе (см. ниже) в строках 23 и 24 определены константы **freq1** и **freq2**, которые определяют длительность удержания сигнала на динамике на верхнем и нижнем уровне. Для этого используется команда **loop**, а значение соответствующей константы записывается в **CX**. Но конкретно об этой команде ниже. Переменные **n_cycl1** и **n_cycl2** определяют количество повторений таких переходов от нуля к единице, то есть, длительность соответствующего звука.

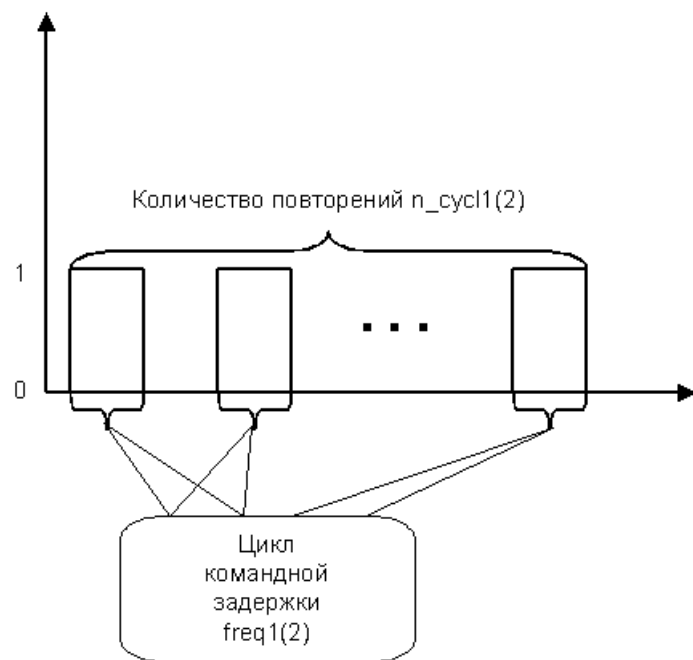


Рисунок 2 – Принцип формирования звука на динамике с помощью командного цикла

Таким образом, из рисунка 2 понятно, что константы **freq1** и **freq2** определяют высоту соответствующего звука, а переменные **n_cycl1** и **n_cycl2** – их длительность.

Итак, начнем разбор первой программы **zw1.asm** (находится в папке 20_04_01). Про комментарии говорить не будем. Начальные строки программы (20-22) уже также очевидны, поэтому на них останавливаться не будем. (Если вопросы, все-таки, возникнут, задавайте по почте.)

Итак, строки 23 и 24 определяют длину цикла задержки (см. выше), а константы в строках 25 и 26 (**n_cycl1** и **n_cycl2**) определяют количество повторений переходов с низкого уровня сигнала на динамике на высокий для первого и второго звуков, соответственно. Последняя константа **port_b** задает адрес системного порта **b** (как мы уже говорили, это адрес **61h**).

В строке стоит инструкция компилятору **.286** (вспомните, что это означает).

Основную программы мы здесь оформляем в виде процедуры (название которой **Start**, и эта же метка определяет точку входа программы, но есть, первую исполняемую команду программы, которая расположена строкой ниже; здесь заметим, что если метка стоит в строке одна, без команды), то ее адрес относится к первой же следующей за ней команде). Отметим также, что определение ближняя (**near**) можно было бы не писать, так как компилятор по умолчанию все процедуры (если не указано другое) делает ближними.

Основная программа состоит из цикла опроса клавиатуры, и определения, какой именно звук надо извлекать.

Первая команда программы (**beg1: call kbin**) вызывает процедуру ввода с клавиатуры (с ожиданием нажатия – функция **0 int 16h**). Затем идут проверки. Строка 31 (**cmp al,'1'**) сравнивает введенный **ASCII** код (в регистре **AL**) с кодом цифры 1. Если это не единица, происходит условный переход на метку **beg2** (строка 32 – **jnz beg2**). Если это, все же, единица, переход не выполняется, а выполняется следующая команда (строка 33 – **call ton1**), то есть, вызывается процедура извлечения первого звука. При этом отметим, что хотя мы говорим о процедуре, так как вызывается последовательность команд, завершающаяся командой возврата, формально как процедура она не оформляется (нет скобок **proc** и **endp**).

После возврата из процедуры извлечения первого звука происходит безусловный переход на начало опроса клавиатуры (строка 34 – **jmp beg1**).

На 35 строке находится метка **beg2**, куда выполняется переход, если введена не единица. В этой строке проверяется введенный символ на совпадение с кодом цифры 2 (**beg2: cmp al,'2'**). Вспомним, что символ в апострофах или в кавычках обозначает **ASCII** код этого символа. Здесь картина та же. Если введена не двойка, выполняется условный переход на метку **beg3** (строка 36 – **jnz beg3**). Если введена двойка, то переход не выполняется, а выполняется команда на следующей строке (строка 37 – **call ton2**), вызывающая процедуру извлечения второго звука. Как и ранее, после

возврата из процедуры производится возврат на начало цикла опроса клавиатуры (строка 38 – **jmp beg1**).

Если не была введена ни единица, ни двойка, осталось проверить, не был ли введен **Esc** (строка 39 – **beg3: cmp ah,1**). Обратим внимание на то, что здесь сравнивается содержимое регистра **АH** (скан-код) с единицей (а не с **ASCII** кодом). Если введен и не **Esc**, выполняется переход на начало цикла опроса клавиатуры (строка 40 – **beg3: cmp ah,1**), так как все символы, кроме единицы, двойки и **Esc**, программа игнорирует.

Если был введен **Esc**, то переход в строке 40 не выполняется, а выполняется команда в строке 41 (**int 20h**) – выход из программы.

На этом основная программа завершается, и далее идет в строке 42 замыкание процедуры (**start endp**).

Далее начинаются процедуры извлечения звуков. Точнее, это одна процедура, но с двумя входами **ton1** и **ton2**. На каждом входе в регистры **DX** и **DI** записываются константы **n_cycl1(2)** и **freq1(2)**, соответственно, а затем, с уже подготовленными параметрами выполняется переход на процедуру собственно извлечения звука **ton0**.

В строках 43 и 44 начинается процедура извлечения звука (**ton2: mov dx,n_cycl2**), где в регистры **DX** и **DI** записываются константы **n_cycl2** и **freq2**. Затем в строке 45 выполняется безусловный переход на процедуру **ton0** собственно извлечения звука (**jmp ton0**).

Следующие строки (46 – **ton1: mov dx,n_cycl1** и 47 – **mov di,freq1**) аналогичны строкам 43 и 44, но в регистры **DX** и **DI** записываются константы **n_cycl1** и **freq1**. Переход после этого не нужен, так как процедура **ton0** начинается сразу после строки 47 (не считая коммандарную строку 48).

Теперь, со строки 49 начинается процедура собственно извлечения звука (**ton0: cli**). Как видим, начинается она с команды запрещения прерываний. Это необходимо, так как мы начинаем манипуляции с системным портом (в котором кроме двух битов управления динамиком, есть еще 6 битов, отвечающих за другие важные для системы вещи). Поэтому перед началом манипуляций с системным портом мы запрещаем прерывания до окончания этих манипуляций.

Затем в строке 50 выполняется чтение содержимого системного порта **b** (**in al,port_b**) в младший байт аккумулятора, так как порт 8-разрядный. Затем (строка 51 – **and al,11111110b**) выполняется поразрядная команда **И** между регистром **AL** и 8-разрядной константой, содержащей все единицы, кроме младшего бита (при этом младший бит сбрасывается, а остальные остаются неизменными). Таким образом мы запрещаем работу второго таймера, отвечающего за системный звук (см. рисунок 2).

Затем (строка 52 – **ton0: or al,00000010b**) принудительно включается динамик (первый по номеру бит системного порта **b** переводится в состояние 1) поразрядной командой **ИЛИ** к 8-разрядной константой, состоящей из всех нулей, кроме первого бита, равного единице (так как это команда **ИЛИ**, первый по номеру бит устанавливается в единицу, а

остальные остаются неизменными). До сих пор все действия мы выполняли в аккумуляторе (**AL**). Теперь (строка 58 – **out port_b,al**) мы записываем это значение в системный порт **b**. Именно с этого момента динамик включен.

Затем выполняем инициализацию счетчика цикла **CX** (строка 59 – **mov cx,di**), записывая туда задержку, и в строке 60 выполняется цикл задержки (**loop \$**). Вспомним, что символ доллара **\$** обозначает текущий адрес компиляции. Таким образом, команда **loop** передает управление на себя же столько раз, сколько записано в **CX**.

Когда цикл кончится (когда **CX** обнулится) управление переходит к следующей команде (строка 61 – **dec dx**) в которой счетчик количества повторений уменьшается на 1. Затем (строка 62 – **jnz ton01**), если в **DX** еще не нуль, управление передается на начало цикла звучания (**ton01**).

Когда **DX** обнулится, выполняется следующая команда (строка 63 – **sti**), которая разрешает все прерывания, а затем (строка 68 – **ret**) выполняется возврат из процедуры. Так как формально мы эту процедуру не оформляли, то здесь нет инструкции завершения процедуры (**endp**).

Затем расположена процедура ввода с клавиатуры (стандартная, уже рассмотренная ранее), формально оформленная как процедура (строки 65-69).

А затем стандартное завершение программы – замыкание сегмента (**code ends**) и обозначение точки входа (**END Start**).

Следует сразу предупредить, что в чистой **DOS** эта программа прекрасно звучит. Константы, использованные в программе подобраны для процессора с частотой примерно 1,5 ГГц. В **DOSBox**'е со звуком, чаще всего, будут возникать проблемы.

В виртуальной машине все зависит от настройки виртуальной машины и от быстродействия процессора (из-за эмуляции процессора все команды выполняются на несколько порядков медленнее, чем самого процессора). Поэтому в виртуальной машине со звуком также будут проблемы.

ВАЖНО! Чтобы в виртуальной машине был хотя бы какой-то звук, необходимо в **Setup** компьютера разрешить виртуализацию (у разных биосов она называется по-разному – виртуализация, механизмы виртуализации и т. д.), а в виртуальной машине разрешить sound (**Virtual PC от Microsoft**) или audio (**Oracle VM VirtualBox**). Причем, если в **Virtual PC от Microsoft** достаточно просто разрешения, то в **Oracle VM VirtualBox** надо еще выбрать подходящую поддержку в выпадающем списке, что не всегда удастся (на одной из машин у меня не получилось). Наибольшая вероятность получить звук именно на **Virtual PC от Microsoft** (она и запускается существенно быстрее).

Возможные эксперименты с программой:

- попробовать различные значения переменных **n_cycl1** и **freq1**, а также **n_cycl2** и **freq2**, определяют количество повторений таких переходов от нуля к единице, то есть, длительность соответствующего звука.

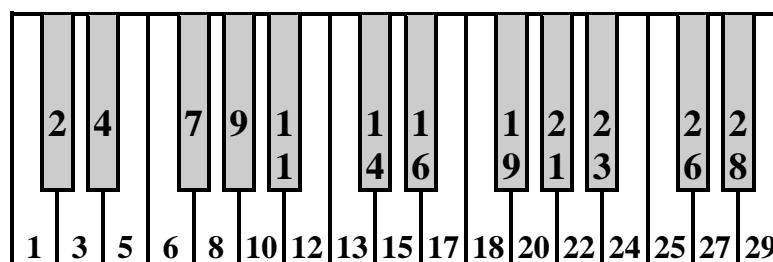
;Написать программу, издающую различные звуки при нажатии на

```

;клавиши '1' и '2'. Выход из программы по клавише Esc.
Assume CS: Code, DS: Code
Code    EGMEN
        org    100h
freq1   equ    30000      ;Задержка переключения 1
freq2   equ    50000      ;Задержка переключения 2
n_cycl1 equ    300        ;Количество циклов (длит.)
n_cycl2 equ    200        ;Количество циклов (длит.)
port_b  equ    61h        ;Адрес системного порта В
        .286
Start   proc    near
beg1:   call    kbin       ; Опрос клавиатуры
        cmp     al,'1'     ; = '1' ?
        jnz     beg2       ; Нет
        call    ton1       ; Звук высоты 1
        jmp     beg1       ; Переход на начало цикла
beg2:   cmp     al,'2'     ; = '2' ?
        jnz     beg3       ; Нет
        call    ton2       ; Звук высоты 2
        jmp     beg1       ; Переход на начало цикла
beg3:   cmp     ah,1       ; Esc ?
        jnz     beg1       ; Нет
        int     20h
start   endp
ton2:   mov     dx,n_cycl2
        mov     di,freq2
        jmp     ton0
ton1:   mov     dx,n_cycl1
        mov     di,freq1
;Звук DX-кол-во циклов,DI-задержка
ton0:   cli     ; Запрещение прерываний
        in      al,port_b  ; Чтение сост. системн. порта В
        and     al,11111110b ; Отк. динамика от таймера
ton01:  or      al,00000010b ; Включение динамика
        out     port_b,al   ; Запись в системный порт В
        mov     cx,di       ; Счетчик цикла задержки
        loop    $           ; Задержка
; Выключение звука
        and     al,11111101b ; Выключение динамика
        out     port_b,al   ; Запись в системный порт В
        mov     cx,di       ; Счетчик цикла задержки
        loop    $           ; Задержка
        dec     dx         ; Декремент счетчика колич. циклов
        jnz     ton01
        sti     ; Разрешение прерываний
        ret
kbin    proc    near ;Ввод с клавиатуры
        mov     ah,0       ;Функция 0
        int     16h        ;клавиатурного прерывания
        ret
kbin    endp
code    ends
        END    Start

```

Опять цитата из пособия. Для исполнения мелодии сначала формируется массив делителей частоты, соответствующих различным нотам звукоряда. Для нот используются номера от 1 до 48. Условное соответствие номеров нот и их значений показано на рисунке 3. Массив делителей частоты для нот называется `noty.asm`. Значение 0 используется в качестве признака окончания мелодии. Значение 255 используется для обозначения паузы.



В программе сформированы массивы для трех мелодий – "Чирик Пырик" (mel1), "Подмосковные вечера" и «Кан-кан» (mel3). Длительность элементарного звука берется равной количеству «тиков» таймера (примерно 1/18 сек.) в переменной **temp** (исходно там записано значение 2). Для получения большей длительности в массиве мелодии записываются подряд несколько одинаковых нот. Для получения четко выраженных соседних одинаковых нот используется пауза.

Как следует из сказанного выше, для определения длительности ноты используется системное прерывание от таймера, а для определения высоты ноты используется выход второго канала таймера с предварительной установкой нужного для данной ноты делителя частоты. Значения коэффициента деления для каждой ноты рассчитывается исходя из исходной (входной) частоты канала (1,19 МГц) и нужной частоты для каждой ноты.

Частоты нот определяются стандартным звуковым рядом: Ля первой октавы должен иметь частоты 440 Гц, в октаве всего 12 полутонов, а частоты нот в соседний октавах отличаются в два раза. То есть, Если Ля первой октавы – 440 Гц, Ля малой октавы – 220 Гц, Ля контроктавы – 110 Гц, а субконтроктавы – 55 Гц. В обратную сторону – Ля второй октавы – 880 Гц, Ля третьей октавы – 1760 Гц и т. д. Каждый соседний (вверх) полутоном отличается от исходного на $\sqrt[12]{2}$, то есть, Си первой октавы – $440 \times \sqrt[12]{2}$, До второй октавы – $440 \times (\sqrt[12]{2})^2$, До диез второй октавы – $440 \times (\sqrt[12]{2})^3$ и т. д., вплоть до Соль диез второй октавы – $440 \times (\sqrt[12]{2})^{11}$. Соответственно, Ля второй октавы – $440 \times (\sqrt[12]{2})^{12} = 440 \times 2 = 880$ Гц.

Подключаемый файл noty.asm содержит комплект 16-разрядных делителей частоты для четырех октав (еще две октавы – нижние – закомментированы).

Подключаемый файл **muz.asm** включает в себя три комплекта нот для исполнения трех разных мелодий. Обратите внимание на периодически встречающиеся значения 255, который соответствуют паузе, отделяющей соседние ноты друг от друга. Отметим также, что признаком окончания мелодии является нулевое значение (арифметическое, а не код нуля).

Итак, начнем разбор программы **zw3.asm** (см. ниже).

Первые строки программы с 5 по 9 соответствуют начальным строкам предыдущей программы.

Со строки 10 по 14 расположены переменные. Первая байтовая переменная **iniflag** (флаг, разрешающий звучание). О нем ниже.

Следующие две словные переменные (строка 11 – **old_int1c_off**, строка 12 – **old_int1c_seg**) предназначены для хранения старого вектора прерывания **int 1ch** (который мы будем перехватывать), который должен быть восстановлен перед выходом из программы.

Словная переменная **tek_mel** (строка 13) предназначена для хранения адреса (смещения) текущей исполняемой мелодии. И, наконец, байтовая переменная **temp**, о которой уже говорили выше, хранит количество тиков таймера на элементарную длительность ноты.

Строки с 15 по 22 содержат команды, сохраняющие старый вектор прерывания **1ch**, и устанавливающие новый.

В строках 15 и 16 (**beg: mov ax, 351ch, int 21h**) при помощи функции **35h** (номер функции в **АН**, номер прерывания в **AL**) системного прерывания **int 21h** берется вектор прерывания **1ch** (напомним, что эта функция возвращает вектор в регистрах **ES** – сегмент и **BX** – смещение). Полученные смещение и сегмент записываются в переменные **old_int1c_off** (**mov old_int1c_off, bx**) и **old_int1c_seg** (**mov old_int1c_seg, es**), соответственно.

В строках с 19 по 21 подготавливаются параметры для функции **25h** системного прерывания **int 21h** (установка нового вектора прерывания): смещение новой процедуры прерывания в **DX** (не забываем, что сегмент новой процедуры должен быть в сегментном регистре **DS**, но об этом уже позаботилась **DOS**, так как у нас программа типа **.com**), номер функции (**25h**) в **АН**, и номер прерывания (**1ch**) и **AL**. Завершает все вызов системного прерывания (**int 21h**) в строке 22. Это была подготовительная секция.

Затем начинается цикл обработки нажатия клавиш, похожий на такой же в предыдущей программе.

Цикл начинается в строке 23 (**beg1: call kbin**) с вызова процедуры ввода с клавиатуры (такой же, как и в предыдущей программе). Разумеется, в этой строке присутствует метка (**beg1:**), так как цикл надо будет замыкать. Всего в цикле надо будет обработать пять нажатий – 1, 2, 3, 4 и **Esc**.

В строке 24 проверяется, был ли введен символ '1' (**cmp al, '1'**), что должно вызвать звучание первой мелодии. Если это была не единица, то происходит условный переход на метку **beg1:** (строка 25 – **jnz beg2**), где будет проверяться двойка. Если все-таки была нажата единица, то переход на **beg2** не происходит, а инициализируется флаг звучания (строка 26 – **mov**

byte ptr iniflag,1) и в переменную **tek_mel** двумя командами (строка 27 – **mov tek_mel,offset mel1**) записывается адрес (смещение) мелодии, которая должна звучать, в данном случае, 1 мелодия). Далее, командой в строке 28 (**jmp beg1**) выполняется безусловный переход на начало цикла обработки нажатия клавиш.

Далее, на строке 29 (куда произошёл переход, если была нажата не единица) начинается последовательность команд, как и выше, но с обработкой цифры 2. В строке 29 (**cmp al,'2'**) проверяется, не была ли нажата цифра 2. Если это была не она, на строке 30 выполняется условный переход на метку **beg3** (**jnz beg3**), где проверка продолжится.

Если все же была нажата цифра 2, переход не выполняется и выполняется команда на строке 31 (**mov byte ptr iniflag,1**), которая, как и выше, взводит флаг звучания. Следующая команда (строка 32 – **mov tek_mel,offset mel2**) записывает в переменную **tek_mel** адрес (смещение) второй мелодии. Затем, как и ранее, (строка 33 – **jmp beg1**) выполняется безусловный переход на начало цикла обработки нажатия клавиш.

Далее в строках с 34 по 38 производятся аналогичные действия для цифры 3 и третьей мелодии. Если была нажата не цифра 3, производится условный переход на метку **beg4** (строка 35 – **jnz beg4**).

С метки **beg4** на строке 39 начинается проверка на нажатую цифру 4 (**beg4: cmp al,'4'**). Если нажата не цифра 4, в строке 40 выполняется условный переход на метку **beg5**. Если, все же была нажата цифра 4, это значит, что надо прекратить звучание. Для этого сбрасывается флаг звучания (строка 41 – **mov byte ptr iniflag,0**), а затем в строке 42 опять выполняется безусловный переход на начало цикла обработки нажатия клавиш.

Если не была нажата ни одна из цифр с 1 по 4, управление попадает на метку **beg5** (строка 43 – **beg5: cmp ah,1**), где выполняется проверка на нажатие клавиши **Esc**. Если нажата не **Esc**, больше проверять ничего не надо (так как любые другие клавиши мы игнорируем) выполняется безусловный переход на начало цикла обработки нажатия клавиш. Если же нажата именно **Esc**, переход на начало цикла не выполняется, а выполняются команды, начиная со строки 46, где восстанавливается старый вектор прерывания **1ch** и осуществляется выход из программы.

Прежде, чем рассматривать эти действия, обратим внимание на то, что цикл обработки нажатия клавиш имеет следующие последствия:

- при нажатии любой из клавиш с 1 по 3, взводится флаг звучания, и в переменной **tek_mel** оказывается адрес (смещение) одной из мелодий (**mel1 ... mel3**),
- при нажатии клавиши 4 сбрасывается флаг звучания (значение переменной **tek_mel** в данном случае не имеет значения),
- при нажатии клавиши **Esc** восстанавливается старый вектор прерывания **1ch**, и выполняется выход из программы.

Вся работа о звучании при этом лежит на новом обработчике прерывания **1ch** (о чем ниже).

Итак, был нажата клавиша **Esc**, и мы пришли на метку **beg5**. В строках 46, 47 в пару регистров **DS:DX** записываются из переменных **old_int1c_seg** и **old_int1c_off** (строка 46 – **mov dx,old_int1c_off**, строка 47 – **mov ds,old_int1c_seg**) сегмент и смещение старого (сохраненного) вектора прерывания **1ch**, соответственно. Затем в строке 48 задается функция **25h** – установка вектора прерывания – (**AH=25h**) системного прерывания **int 21h** для вектора прерывания **1ch** (**AL=1ch**) – **mov ax,251ch**.

Затем в строке 49 вызывается собственно системное прерывание **int 21h**, а в строке 50 осуществляется выход из программы (**int 20h**).

На этом собственно программа закончилась. Осталось описать новую процедуру обработки прерывания **1ch**. Напомню, что это пользовательское прерывание по таймеру, которое каждый раз вызывается из системной процедуры обработки прерывания по таймеру. Исходно вся пользовательская процедура обработки прерывания **1ch** состоит лишь из одной команды: **iret**. Мы перехватываем это прерывание (заменяем в векторе прерываний с номером **1ch** адрес на адрес своей процедуры, название которой уже упоминалось в строке 19 (**new_int1c**)).

Итак, новая процедура обработки прерывания **1ch** – **new_int1c** (строка 52). Мы ее оформляем как процедуру, естественно, дальнего типа (подумайте, почему).

Начинается процедура с команды сохранения всех регистров общего назначения в стеке (зачем?) – строка 53 (**pusha**).

Подумаем, что должна делать эта процедура. Ее задача очень проста: уменьшать при каждом ее вызове переменную **temp** на единицу. Если **temp** не равна нулю, выходить из процедуры без последствий. Когда **temp** обнуляется, надо вызывать процедуру звучания (**muz**), не забывая записать в переменную **temp** нужное значение (у нас 2).

Поэтому вторая команда процедуры в строке 54 – (**dec byte ptr cs:temp**) уменьшает на единицу значение переменной **temp**, а затем в строке 55, если не взведен флаг нуля (то есть, в переменной **temp** не ноль) выполняется условный переход на метку **ni1** (то есть, на выход из процедуры).

Если в переменной **temp** все же ноль, в строке 56 в эту переменную опять записывается значение 2 (**mov byte ptr cs:temp,2**), и вызывается процедура звучания (строка 57 – **call muz**).

Здесь стоит остановиться на том, что в строках 54 и 56 перед переменной **temp** стоит префикс переопределения сегмента (**cs:temp**). На первый взгляд он не нужен, так как по умолчанию переменная ищется в сегменте данных (**DS**), который заботами **DOS** содержит то же значение, что и **CS**. В этих рассуждениях мы забыли о том, что процедура **new_int1c** вызывается не из нашей программы, а из системной процедуры обработки прерывания **int 8**, в которой содержимое регистра **DS** соответствует именно этой процедуре. Поэтому сегментный префикс здесь необходим (позже мы обсудим это в экспериментах).

Наконец, после возврата из процедуры **muz** (или, если в переменной **temp** еще не нуль, то без этой процедуры) мы попадаем на строку 58 (**ni1: popa**), где из стека восстанавливаются все регистры общего назначения, после чего в строке 59 выполняется команда возврата из процедуры (на сей раз процедура имеется в наличии) – **iret**.

Завершается процедура стандартным предложением в строке 60 (**new_int1c endp**).

Теперь начинается процедура извлечения звука (строка 61 – **muz proc near**). Эта процедура ближняя, так как она находится в том же сегменте, что и место ее вызова. Не забываем, однако, что в сегментном регистре **DS** находится «чужое» значение, поэтому у всех переменных в этой процедуре необходимо использовать сегментный префикс **ES**:

Первая команда процедуры (строка 62 – **test byte ptr es:iniflag,0ffh**) – проверка на неравенство нулю флага звучания (можно было бы сравнивать с 1, так как при инициализации этого флага мы записываем в него именно единицу, но, на всякий случай, сравниваем с **0ffh**).

Если флаг звучания не равен нулю (сброшен флаг нуля), то на строке 63 (**jnz muz1**) выполняется команда условного перехода на организацию собственно звучания. Если же флаг звучания нулевой, то переход на строке 63 не выполняется, в строке 64 выполняется переход на процедуру выключения звука (**call off**).

Строка 65 – **muz1: mov si,cs:tek_mel**. На эту метку передается управление, если флаг звучания взведен (не нуль). Команда в этой строке помещает в регистр **SI** адрес (смещение) мелодии, которая должна звучать из переменной **tek_mel**. На самом деле, в этой переменной, как это будет видно позже, хранится текущая нота текущей мелодии (при нажатии клавиши 1, 2 или 3 в переменную записывается адрес первой ноты выбранной мелодии). Не забываем о необходимости сегментного префикса **cs**:

В строке 66 номер текущей ноты записывается в регистр **BL** (**mov bl,cs:[si]**). В строке 67 (команда **cmp bl,255**) это значение проверяется на значение 255 (пауза). Если это не пауза, в строке 68 (**jnz muz2**) выполняется условный переход на метку **muz2** для продолжения анализа. Если значение было 255, переход в строке 68 не выполняется и в строке 69 вызывается процедура выключения звука (**call off**). После чего в строке 70 выполняется переход на следующую ноту мелодии (**inc cs:tek_mel**), после которого выполняется выход из процедуры (строка 71 – **ret**).

Если это не пауза, то выполняется условный переход на метку **muz2** (строка 72 – **muz2: or bl,bl**), где проверяется, не является ли текущая нота нулем (признаком окончания мелодии). Отметим, что для проверки здесь используется команда **ИЛИ** для регистра **BL** с самим собой. При этом флаг нуля будет взведен только, если в **BL** все нули. (Очевидно, что для той же цели и с таким же эффектом можно было использовать команду **И**).

Если в **BL** не нуль, значит мелодия продолжается, и в строке 73 выполняется условный переход на продолжение обработки мелодии (**jnz muz3**). Если это был все же нуль, мелодия закончилась, и надо выключить

звучание, что и делается переходом процедуру выключения звучания (строка 74 – **jmp off**).

Как уже говорилось, при ненулевом значении ноты управление попадает на метку **muz3**. Здесь (строка 75 – **muz3: shl bl,1**) производится умножение номера ноты на 2 (путем сдвига значения на один двоичный разряд влево). Удвоение значения ноты необходимо потому, что значение делителя для каждой ноты имеет формат слова (два байта), и для перехода от одной ноты к следующей необходимо добавить к адресу (смещению) двойку.

Для получения 16-разрядного адреса обнуляется регистр **ВН** (строка 76 – **xor bh,bh**), после чего в **ВХ** получился 16-разрядный удвоенный номер ноты.

В строке 77 значение делителя для текущего номера ноты загружается в **АХ** (**mov ax,cs:noty[bx]**). Косвенная адресация **cs:noty[bx]** означает, что в сегменте **CS** берется значение по адресу **noty + <ВХ>**, то есть, от начала таблицы **noty** отсчитывается столько слов, какое значение хранится в **ВХ**.

Затем в строке инкрементируется содержимое переменной **tek_mel** (**inc cs:tek_mel**) ждя перехода к следующей ноте мелодии. Отметим, что, хотя из команды не очевиден формат операнда (переменной **tek_mel**), **word ptr** здесь указывать не обязательно, так как компилятор по умолчанию все делает именно такого формата (хотя для очистки совести, может быть, и стоило указать **word ptr**).

В строках 80-82 производится загрузка делителя частоты во второй таймер. Таймер запрограммирован на двухбайтовую загрузку 16-разрядного делителя частоты в последовательности младший байт, старший байт (возможно много других вариантов двухбайтовой и однобайтовой загрузки). Происходит это следующим образом: в строке 80 в порт **42h** (порт второго таймера) записывается младший байт делителя из **АЛ** (**out 42h,al**), затем (так как 8-разрядный порт в базовой системе команд может загружаться только из **АЛ**) содержимое **АН** и **АЛ** меняется местами (строка 81 – **xchg al,ah**). Наконец, старший байт делителя (который теперь находится в **АЛ**, также записывается в порт **42h** (строка 82 – **out 42h,al**).

Вслед за этим вызывается процедура включения звука (строка 83 – **call on**). После этого выполняется выход из процедуры (строка 84 – **ret**).

Завершается все замыканием процедуры **muz** (строка 85 – **muz endp**).

Осталось лишь рассмотреть три процедуры: ввода с клавиатуры (**kbin**), которую мы уже рассматривали (она оформлена в виде процедуры), выключения звука (**off**) и включения звука (**on**), причем последние две не оформлены, как процедуры (хотя и завершаются командой возврата из процедуры).

Процедуру ввода с клавиатуры мы повторно рассматривать не будем.

Процедура выключения звука начинается в строке 92 (**off: in al,61h**). Собственно выключение звука заключается в сбросе двух младших битов системного порта **b** (**61h**). Прежде всего, здесь имеется метка **off:**. Здесь в **АЛ** читается системный порт **b** (адрес **61h**), затем с помощью команды поразрядного **И** сбрасываются два младших бита **АЛ**, запрещая таким

образом любое звучание – от таймера или непосредственное (строка 93 – **and al,0fch**). Вспомним, что **0fch=11111100b**, то есть командой **И** обнуляются биты 0 и 1. Получившееся значение (со сброшенными двумя младшими битами опять записывается в системный пор **b** (строка 94 – **out 61h,al**). Обратим внимание на то, что все остальные биты, кроме сброшенных, остаются неизменными, что очень важно. На этом функция процедуры, поэтому в строке 95 осуществляется выход из нее (**ret**). Надеюсь, не надо напоминать, что, так как процедура ближняя, то компилятор заменит эту команду на **retb**.

В процедуре включения звука происходит все наоборот – взводятся два младших бита системного порта **b** (**61h**). Для этого в строке 97 (**on: in al,61h**). в **AL** читается системный порт **b** (адрес **61h**), затем с помощью команды поразрядного **ИЛИ** принудительно устанавливаются два младших бита **AL**, разрешая таким образом звучание (строка 98 – **or al,3**). Вспомним, что **3=00000011b**, то есть командой **ИЛИ** взводятся биты 0 и 1. Получившееся значение (с установленными двумя младшими битами опять записывается в системный пор **b** (строка 99 – **out 61h,al**). Обратим внимание на то, что опять все остальные биты, кроме установленных, остаются неизменными. Затем идет естественный выход из процедуры (строка 100 – **ret**).

Затем идут две строки включения файлов мелодий и нот строки 101 – **include muz.asm** и 102 – **include noty.asm**). Они приведены ниже.

В строках 103 и 104 – стандартное завершение программы (замыкание сегмента кода – **code ends** и обозначение точки входа – **END Start**).

Так как высота нот в данной программе определяется делителем частоты, а их длительность – системным таймером, то эксперименты, аналогичные предыдущей программе здесь выполнить не удастся. Здесь можно провести эксперименты с темпом (переменная **temp**). Для начала попробуйте замедлить темп вдвое (увеличив вдвое значение в переменной **temp**). При этом помните, что значение этой переменной определяется не только в строке 14 (**temp db 2**), но и в строке 56 (**mov byte ptr cs:temp,2**). Интересно также попробовать задать значение 0. Попробуйте объяснить, что при этом происходит.

Программа zw3.asm

```
; Программа, играющая мелодию. Исполняет при нажатии клавиш
; '1' "чирик-пырик", '2' - "подмосковные вечера", '3' - кан-кан
; При нажатии клав. '4' - звук прекращается.
; Используется прерывание от системного таймера.
Assume CS: Code, DS: Code
Code SEGMENT
    org     100h
    .286

Start:    jmp     beg
iniflag   db      0 ; Флаг звучания
old_intlc_off dw    0 ; Смещение старого вектора
old_intlc_seg dw    0 ; Сегмент старого вектора
tek_mel   dw      ?
```

```

temp          db          2 ;Темп
beg:  mov      ax,351ch ; Сохранение старого вектора 1с
      int      21h
      mov      old_int1c_off,bx ; Запись смещения
      mov      old_int1c_seg,es ; Запись сегмента
      lea      dx,new_int1c ; Запись нового вектора 1с
      mov      ah,25h ; Функция установки вектора прерыв.
      mov      al,1ch ; Номер вектора прерывания
      int      21h ; DS:DX - адрес новой программы обр.
beg1: call     kbin ; Опрос клавиатуры
      cmp      al,'1' ; = '1' ?
      jnz      beg2 ; Нет
      mov      byte ptr iniflag,1 ; Взведение флага звуч.
      lea      ax,mel1
      mov      tek_mel,offset mel1 ; 1 Мелодия
      jmp      beg1 ; Переход на начало цикла
beg2: cmp      al,'2' ; = '2' ?
      jnz      beg3 ; Нет
      mov      byte ptr iniflag,1 ; Взведение флага звуч.
      mov      tek_mel,offset mel2 ; 2 Мелодия
      jmp      beg1 ; Переход на начало цикла
beg3: cmp      al,'3' ; = '3' ?
      jnz      beg4 ; Нет
      mov      byte ptr iniflag,1 ; Взведение флага звуч.
      mov      tek_mel,offset mel3 ; 3 Мелодия
      jmp      beg1 ; Переход на начало цикла
beg4: cmp      al,'4' ; = '4' ?
      jnz      beg5 ; Нет
      mov      byte ptr iniflag,0 ; Сброс флага звуч.
      jmp      beg1 ; Переход на начало цикла
beg5: cmp      ah,1 ;Esc ?
      jnz      beg1 ; Нет
; Восстановление старого вектора 1с и выход
      mov      dx,old_int1c_off ; Смещение старого вектора
      mov      ds,old_int1c_seg ; Сегмент старого вектора
      mov      ax,251ch ; Установка старого вектора 1с
      int      21h
      int      20h
; Новый обработчик прерывания 1с
new_int1c proc far
      pusha
      dec      byte ptr cs:temp
      jnz      nil
      mov      byte ptr cs:temp,2
      call     muz ; Вызов процедуры извлечения звука
nil:  popa
      iret
new_int1c endp
muz   proc      near
      test     byte ptr cs:iniflag,0ffh ; Проверка флага
      jnz      muz1 ; Продолжение
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
        ret                ; 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
        call       off        ;Выключение звука
        inc        cs:tek_mel;Переход к адресу след. ноты
        ret
muz2:  or         bl,bl      ;= 0 ?
        jnz        muz3      ;Продолжение
        jmp        muze       ;Выход, если признак конца
muz3:  shl        bl,1       ;Умножение bl на 2
        xor        bh,bh     ;bh = 0
        mov        ax,cs:noty[bx]; В DI частота ноты
; 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
        call       on        ;Включение звука
        ret                ;Нормальный выход
muz     endp
kbin   proc      near       ;Ввод с клавиатуры и проверка на выбор игры
        mov        ah,0      ; Функция 0
        int        16h       ; клавиатурного прерывания
        ret
kbin   endp
;Выключение звука
off:   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
        ret
;Включение звука
on:    in         al,61h     ;Чтение состояния системного порта В
        or         al,3       ;Разрешение звучания (биты 0 и 1)
        out        61h,al    ;Запись в системный порт В
        ret
include muz.asm
include noty.asm
code    ends
        END            Start

```

Подключаемые файлы

noty.asm

;noty.asm Ноты 4 октавы (всего 6 октав)

```

;noty
8e88h,868dh,7effh,77d1h,7120h,6ac0h,64cch,5f1eh,59c5h,54beh,4ff9h,4b7dh    dw
;noty
4744h,4346h,3f7fh,3be8h,3890h,3560h,3266h,2f8fh,2ce2h,2a5fh,27fch,25beh    dw
noty
23a2h,21a3h,1fbfh,1df4h,1c48h,1ab8h,1933h,17c7h,1671h,152fh,13feh,12dfh    dw
11dlh,10dlh,0fdfh,0efah,0e24h,0d58h,0c99h,0be3h,0b38h,0a97h,09ffh,096fh    dw
08e8h,0868h,07efh,077dh,0712h,06ach,064ch,05f1h,059ch,054bh,04ffh,04b7h    dw
0474h,0434h,03f7h,03beh,0389h,0356h,0326h,02f8h,02ceh,02a5h,027fh,025bh    dw

```

muz.asm

```
;muz.asm Мелодии: чижик-пыжик, подмосковные вечера, кан-кан
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 8,8,255,8,8,255,8,8,255,10,12,255
      db 13,13,255,13,13,255,13,13,13,13
      db 0
mel2 db 13,13,13,16,16,16,20,20,20,16,16,16,18,18,18
      db
18,18,18,16,16,16,15,15,15,20,20,20,20,20,20,18,18,18,18,18,18
      db 13,13,13,13,13,13,0
mel3 DB 18,6,25,13,22,18,25,13,20,1,23,8,22,5,20,1
      DB 25,6,13,13,25,10,13,13,25,10,27,13,22,10,23,13,20,1
      DB 11,11,20,5,11,11,20,1,23,11,22,5,20,11,18,6,30,18
      DB 29,17,27,15,25,13,23,11,22,10,20,8,18,6,13,13,18,10
      DB 13,13,20,1,23,8,22,5,20,8,25,6,13,13,25,10,13,13
      DB 25,6,27,13,22,10,23,13,20,1,8,8,20,5,8,8,20,1,23,8
      DB 22,5,20,8,18,6,25,10,20,13,22,10,18,6,6,6,6,6,6,6
      DB 34,8,24,12,24,15,34,12,32,1,25,5,25,8,29,5,30,6,34,13
      DB 37,10,34,13,34,1,32,8,32,5,8,8,34,8,24,12,24,15,34,12
      DB 32,1,25,5,25,8,29,5,29,3,27,7,29,10,27,13,34,12,32,8
      DB 34,6,32,3,34,8,24,15,24,12,34,15,32,1,29,8,25,5,29,8
      DB 30,6,34,13,37,10,34,13,34,1,32,5,32,8,5,5,34,8,24,15
      DB 24,12,34,15,32,1,25,6,25,5,29,8,29,3,27,7,29,10,27,7
      DB 32,8,30,6,29,5,27,3,25,1,8,8,25,5,8,8,27,12,30,15
      DB 29,8,27,12,32,1,8,8,32,5,8,8,32,1,34,8,29,5,30,8,27,8
      DB 15,15,27,12,15,15,27,8,30,12,29,15,27,12,25,1,37,1
      DB 36,5,34,6,32,8,30,8,29,10,27,12,25,1,8,8,25,5,8,8
      DB 27,8,30,15,29,12,27,15,32,1,8,8,32,5,8,8,32,1,34,8
      DB 29,5,30,8,27,8,15,15,27,12,15,15,27,8,30,15,29,12
      DB 27,15,25,1,32,8,27,5,29,8,25,1,32,32,37,37,0
```

Дополнительно рассмотрим программу, которую уже рассматривали на занятиях очно, но там не были рассмотрены некоторые тонкости.

Прежде всего, опишем функцию ввода в буфер (ф. **0ah** системного прерывания **int 21h**). Подробно она описана в **rhelp**'е.

Для такого ввода надо подготовить буфер в таком формате:

max	?	?	?	?	...	?	odh
------------	----------	----------	----------	----------	------------	----------	------------

Здесь **max** – максимально допустимая длина ввода (от 1 до 254). Как видно из рисунка, буфер должен быть, как минимум, на 2 байта больше **max**.

Ввод завершается либо возвратом каретки, либо по заполнении буфера. Функция позволяет ввести **max – 1** символ (так как в конце так или иначе добавится возврат каретки – **0dh**). Если вводимая строка завершается нажатием **Enter**, ввод заканчивается, при этом во второй байт буфера записывается реально введенное количество символов (символ **0dh** не засчитывается в длину введенной строки). Если достигнут максимум ввода (**max – 1**), то каждое следующее нажатие клавиши вызывает системный писк, а введенный символ игнорируется. После нажатия **Enter** ввод завершается.

Ниже показано состояние буфера после завершения ввода.

max	len	Символ	Символ	Символ	...	Символ	odh
------------	------------	--------	--------	--------	-----	--------	------------

Начнем анализ программы (называется `palind.asm`, содержится в прилагаемой в архиве папке **04_01N**).

Строки 2-4 стандартные, описания не требуют.

Далее идет описание двух макросов (напоминаю, что перед компиляцией текст макросов будет подставлен в места, где они упомянуты, а на этом месте от них ничего не останется).

Первый макрос вывода текста на консоль. Для этого используется функция записи в файл, а в качестве файла выступает консоль вывода (экран) с предопределенным дескриптором 2.

Итак, строка 6 – **`disp macro text,num`**. **`disp`** – это имя макроса, под которым он будет использоваться, **`macro`** – ключевое слово, обозначающее, что здесь начинается описание макроса. Далее идут формальные параметры (также, как и в Си) **`text`** и **`num`**, на место которых при записи макроса быдыт подставляться фактические значения. Параметр **`text`** – адрес выводимой строки, параметр **`num`** – количество записываемых (выводимых на экран) символов.

Дальше идет оформление функции. В строке 7 (**`mov cx,num`**) в **`CX`** записывается количество выводимых символов, в строке 8 (**`mov ah,40h`**) в **`AH`** записывается номер используемой функции, в строке 9 (**`lea dx,text`**) в **`DX`** записывается адрес выводимой строки (не забываем, что в **`DS`** уже заботами **`DOS`** записан сегментный адрес этой строки, в **`BX`** (**`mov bx,2`**) записывается дескриптор консоли, как файла, и, наконец, в строке 11 вызывается собственно системное прерывание (**`int 21h`**).

Ключевое слово **`endm`** в строке 12 означает, что описание ранее объявленного макроса завершено.

Следом идет описание макроса выхода из программы с ожиданием нажатия клавиши. Макросом без параметров, поэтому начало и конец макроса описывают стандартные скобки начало макроса (строка 13 – **`outp macro`**) и конец его (строка 17 – **`endm`**). В теле макроса всего три команды обнуление **`AH`** – нулевая функция (строка 14 – **`xor ah,ah`**), клавиатурное прерывание (строка 15 – **`int 16h`**) и выход из программы (строка 16 – **`int 20h`**).

Далее начинается собственно программа (строка 18 – **`org 100h`**, понятно для чего, и строка 19 – **`Start: jmp begin`**). Здесь метка для точки входа и перескок через переменные на метку **`begin`**.

Далее идут переменные. В строках 20 и 21 сыормирован входной буфер для функции буферизованного ввода **`0ah`**, описанной выше. Этот буфер здесь состоит из двух кусочков. Первый кусок в строке 20 (**`b db 20,0`**) определяет первые два байта буфера функции. В первом байте записано 20 (определяющее максимально возможное количество вводимых символов), во втором байте 0 (хотя можно было и ?), куда **`DOS`** запишет реальную длину

введенной строки. Второй кусок буфера собственно для вводимых символов (на всякий случай его объем взят заведомо большим, чем может быть введено символов; вдруг Вам захочется увеличить количество вводимых символов). Он определен в строке 21 (**buf db 64 dup('0')**). Если кто не помнит, то **buf** – это метка буфера, **db** – признак байтовых данных, **64 dup('0')** – 64 подряд байта, заполненных нулями.

В строке 22 определен точно такой же вспомогательный буфер – **bufo db 64 dup('0')** (о его назначении будем говорить ниже).

Последняя переменная (строка 23 – **n dw ?**) – словная переменная в которой хранится количество введенных символов (эта переменная нужна для манипулирования с введенными строками).

Далее в строке 24 начинаются команды программы.

Строка 24 – только лишь метка **begin:**, на которую производится перескок со стартовой метки **Start**.

Строки с 25 по 27 реализуют функцию буферизованного ввода. Строка 25 (**mov ah,0ah**) задает номер функции **0ah**, строка 26 (**lea dx,b**) задает адрес специального буфера для этой функции (начинающегося за 2 байта до начала вводимых символов, см. выше), строка 27 (**int 21h**) – собственно вызов системного прерывания.

Далее регистр **BX** подготавливается для работы со входным буфером (строка 28 – **lea bx,buf**). Для этого в **BX** загружается адрес первого введенного символа (если был ввод). Как мы помним, перед первым символом после ввода находится количество действительно введенных символов, которое загружается в **AL** (строка 29 – **mov al,[bx-1]**).

Далее, в строке 31 проверяется, а был ли ввод (**test al,0ffh**). Если после этой команды взведен флаг нуля, значит там был нуль. Поэтому в строке 32 записан условный переход по флагу нуля на метку **jb3 (jz b3)**, с которой начинается вывод сообщения о необходимости ввода символов с последующим выходом из программы.

Если переход на строке 32 не произошел, это значит, что ввод был не нулевой, и мы в строке 34 превращаем количество введенных символов *путем обнуления старшего байта аккумулятора) из байта в слово (**xor ah,ah**), после чего записываем это значение в переменную **n** (строка 35 – **mov n,ax**).

Затем в строке 36 вызывается печать возврата каретки и перевода строки, чтобы отделить последующий вывод сообщений от введенной строки (**call wk**).

Затем на экран выводятся введенные символы (строка 37 – **disp buf,ax**) просто, чтобы убедиться, что ввод выполнен верно. Как видно из команды, для вывода строки на экран используется соответствующий макрос, в качестве адреса текста указан **buf**, который содержит только введенные символы (без первых служебных байтов), а количество выводимых на экран символов задается значением в регистре **AX**, в который ранее было сформировано 16-разрядное значение.

Для проверки введенной последовательности на палиндром используется вспомогательный буфер, в котором введенные символы будут расположены в обратном порядке (для упрощения процедура сравнения символов).

В строке 38 в регистр **DI** записывается адрес вспомогательного буфера **bufo** (**lea di,bufo**), который будет использоваться при анализе. В строке 39 к этому адресу добавляется количество введенных символов (**add di,n**), в результате чего получился адрес **n+1** байта (так как номера идут с нуля). Чтобы получить адрес **n**-ного байта, в строке 40 полученный адрес уменьшается на 1 (**dec di**).

Далее в строке 41 инициализируется счетчик пересылки (**mov cx,n**), а в строке 42 в регистр **BX** записывается адрес первого символа введенной строки (**lea bx,buf**). Хотя мы уже записывали этот адрес в **BX** в строке 27, надо учитывать, что макрос **disp** портит этот регистр.

Затем в строке 43 начинается цикл переписи введенных символов из буфера **buf** в буфер **bufo** в обратном порядке (**b1: mov al,[bx]**). Текущий символ (вначале первый) из первого буфера записывается в **AL**, затем символ из **AL** записывается на последнее место во втором буфере (строка 44 – **mov [di],al**), затем адрес в первом буфере сдвигается на следующий байт (строка 45 – **inc bx**), адрес во втором буфере сдвигается на предшествующий байт (строка 46 – **dec di**), и цикл замыкается (строка 47 – **loop b1**). Таким образом, из первого буфера во второй будут переписаны в обратном порядке все введенные символы, так как команда **loop** каждый раз уменьшает **CX** на 1, и передает управление на указанную метку до тех пор, пока в **CX** не нуль.

После выхода из цикла в строке 48 на экран выводятся перевод строки и возврат каретки (**call wk**), а затем на экран выводится содержимое второго буфера (строка 49 – **disp bufo,n**), в котором символы расположены в обратном порядке. Две предыдущие печати выполняются лишь для наглядного представления символов в первом и втором буферах.

Теперь пора сравнить, совпадают ли символы в первом и втором буферах (то есть, проверить, является ли введенная последовательность палиндромом).

Для сравнения будет использована строковая команда, для которой надо подготовить адрес первого операнда в **DS:SI**, адрес второго операнда в **ES:DI**, а в **CX** – количество сравниваемых символов. Именно это и делается в строках 50-52.

Формирование счетчика – (строка 50 – **mov cx,n**), формирование адреса первого операнда – (строка 51 – **lea si,buf**), формирование адреса второго операнда – (строка 52 – **lea di,bufo**). При этом не забываем, что благодаря заботам **DOS** у нас содержимое всех сегментных регистров одинаково, и совпадает с **CS**.

Далее, в строке 53 – главная команда **repe cmpsb** (префикс повторения **repe** будет повторять команду до тех пор, пока есть совпадение, сама команда **cmpsb** сравнивает байты (потому что на конце команда буква **b**) первого и второго операнда, взводя соответствующие флаги. После

окончания выполнения команды сравнения проверяется флаг нуля. Если он взведен, значит все символы до последнего совпали. Если флаг сброшен, было несовпадение. При желании место несовпадения можно определить, посмотрев содержимое **CX**, в котором находится расстояние несовпадения от конца сравнения. Нам это место неважно, поэтому мы просто проверяем флаг нуля (строка 54 – **jnz b2**). Эта команда выполняет условный переход (при сброшенном флаге нуля, то есть, при несовпадении) на метку **b2**, где выводится сообщение о том, что введенная последовательность не палиндром. Если переход в строке 54 не выполняется, значит было совпадение, и введенная последовательность является палиндромом, сообщение о чем и формируется в строках 55-57. В строке 55 задается функция вывода строки на экран **mov ah,9**, в строке 56 в **DX** загружается адрес сообщения о том, что введен палиндром (**lea dx,Yes**), в строке 57 вызывается системное прерывание (**int 21h**). После этого в строке 58 выполняется выход из программы (с ожиданием нажатия клавиши) при помощи соответствующего макроса (**outp**).

Далее в строке 59 (**b2: mov ah,9**) находится метка **b2**, на которую передается управление при несовпадении (то есть, когда введенные символы не являются палиндромом). В этой строке опять задается функция вывода строки на экран **mov ah,9**, в строке 60 в **DX** загружается адрес сообщения о том, что введен не палиндром (**lea dx,No**), в строке 61 вызывается системное прерывание (**int 21h**). После этого в строке 62 выполняется выход из программы (как и ранее).

В строках 65-68 расположена процедура сообщения о том, что символы не были введены с последующим выходом из программы.

В строке 65 находится метка **b3:**, куда передается управление при отсутствии (нулевом) вводе (**b3: mov ah,9**). В строке 66 в **DX** загружается адрес сообщения о том, что не введены символы (**lea dx,noin**), в строке 67 вызывается системное прерывание (**int 21h**). После этого в строке 68 выполняется уже описанный выход из программы.

Далее, в строках 70-72 помещены тексты выводимых сообщений. Пояснений эти строки не требуют – все стандартно. Надо только запомнить, что в 9 функции в качестве терминатора сообщения выступает символ доллара (\$), а числа 10 и 13 в сообщениях отображают перевод строки и возврат каретки, соответственно.

Ниже пошли процедуры печати одного символа и другие вспомогательные процедуры, которые мы неоднократно разбирали ранее.

В строках 103 и 104 находится стандартное завершение программы.

Текст программы **palind.asm**

```
;Проверка на палиндром - 1 вариант
Assume CS: Code, DS: Code
Code SEGMENT
    .486
disp macro text,num ;Кол.выводимых символов
    mov cx,num
    mov ah,40h ;Запись в файл
```

```

        lea dx,text;Выводимый текст
        mov bx,2    ;Дескриптор консоли (выв)
        int 21h
    endm
    outp macro
        xor ah,ah
        int 16h
        int 20h
    endm
    org 100h
    Start:    jmp begin
    b         db 20,0 ;max 20,кол-во ввода
    buf       db 64 dup('0');
    bufo      db 64 dup('0')
    n         dw ?    ;Кол-во ввода
    begin:
        mov ah,0ah ;Ввод в буфер
        lea dx,b    ;За 2 байта до строки
        int 21h
        lea bx,buf   ;Адр.буф.ввода
        mov al,[bx-1];Кол-во ввода
;*****
        test al,0ffh ;Проверка на 0
        jz b3        ;Нет ввода
;*****
        xor ah,ah    ;Оно же 16-разр
        mov n,ax     ;Оно же в перем.
        call wk
        disp buf,ax ;Печать 1 буфера
        lea di,bufo ;Второй буфер
        add di,n ;+Кол-во символов
        dec di ;Указание на последний символ
        mov cx,n ;счетчик символов
        lea bx,buf ;Адр.вх.буф.
;Перепись из первого буфера во второй в обр.порядке
b1: mov al,[bx] ;Символ из первого буфера
    mov [di],al;Во второй буфер в конец
    inc bx ;След.символ в 1 буфере
    dec di ;Пред.симв. во 2 буфере
    loop b1 ;Цикл по CX
    call wk
    disp bufo,n;Печать 2 буфера
    mov cx,n ;Счетчик сравнений
    lea si,buf ;DS=CS
    lea di,bufo ;ES=CS
    repe cmpsb ;Сравнение строк (пока совпадают)
    jnz b2 ;Несовпадение
    mov ah,9
    lea dx,Yes
    int 21h
    outp
b2: mov ah,9
    lea dx,No

```

```

        int 21h
        outp
;*****
;Здесь не были введены символы
        b3: mov ah,9
        lea dx,noin
        int 21h
        outp
;*****
Yes db 10,13,'Палиндром$'
No db 10,13,'Не палиндром$'
noin db 10,13,'Не были введены символы$'
;*****
; Печать одного hex символа из мл. тетр. al
prss proc near
        and al,0fh
        add al,30h
        cmp al,39h
        jle print
        add al,7
; Печать 1 ASCII символа
print proc near
        pusha
        mov bx,0fh
        mov ah,0eh
        int 10h
        popa
        ret
print endp
prss endp
; Печать пробела
wk: pusha
        mov al,0ah
        call print
        mov al,0dh
        call print
        popa
        ret
space proc near
        mov al,32 ; Код пробела
        jmp print ; Печать 1 ASCII символа
space endp
Code ends
        end Start

```

В заключении отмечу, что комментарии к очевидным вещам я здесь не писал, но если будут вопросы, задавайте.