ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

УПРАВЛЕНИЕ ДИСТАНЦИОННОГО ОБУЧЕНИЯ И ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Кафедра «Вычислительные системы и информационная безопасность»

Сборник упражнений

по дисциплине

«Программирование микропроцессорных систем»

Автор

Цветкова О.Л.

Ростов-на-Дону, 2014

Аннотация

Приводятся теоретические сведения, необходимые для выполнения лабораторных работ, порядок выполнения и индивидуальные варианты.

Автор

к.т.н., доцент
Цветкова О.Л.

Оглавление

ЛАБОРАТОРНАЯ РАБОТА № 1
СОСТАВЛЕНИЕ, ТРАНСЛЯЦИЯ И КОМПОНОВКА ПРОГРАММ НА АССЕМБЛЕРЕ

Цель работы: изучение типовой структуры программы на языке ассемблера ЭВМ IBM PC и освоение методов набора текста программы, ее трансляции и компоновки.

Теоретические сведения

Типовая структура программы на языке ассемблера

Рассмотрим программу, выполняющую следующие действия: размещение в памяти двух переменных; увеличение первой из них на 5 с сохранением предыдущего состояния аккумулятора, присвоение второй переменной значения первой переменной, восстановление первоначального состояния аккумулятора состояния аккумулятора, вывод на экран сообщения "OK" при нормальном завершении программы с последующим выходом в операционную систему при нажатия любой клавиши.

Текст исходной программы на языке ассемблера имеет следующий вид:

```
title   summa                          ; присвоение модулю имени
summa
assume  cs:code, ds:data, ss:stack     ; установка базовых адресов
                                       ; сегментных регистров
data segment para 'data'               ; начало сегмента данных
var_1    dw    0                       ; определение и инициализация
двух
var_2   dw    0                        ; переменных
data    ends                           ; конец сегмента данных
stack segment para 'stack'             ; начало сегмента стека
dw     10 dup(?)                       ; резервироватние 10 слов
stk_top label word                     ; вешина стека
stack  ends                            ; конец сегмента стека
code segment para 'code'               ; начало сегмента кода
start: mov ax,data                     ; метка входа в программу и
mov ds,ax                              ; инициализация регистра ds
mov ax,stack                           ; инициализация регистров
mov ss,ax                              ; гистров ss и sp
mov sp,offset stk_top
prog:  push ax                         ;занесение      содержимого
регистрах
```

```
                        ; в стек
mov ax,var_1            ; занесение в ;ax первой переменной
add ax,5                ; увеличение var_1 на 5
mov var_2,ax            ; присвоение var_2 значения (ax)
pop ax                  ; восстановить ax (вызов из стека)
 mov ah,2               ; задание ОС режима вывода на экран
 mov dl,'O'             ; занесение в регистр dl и вывод на
 int 21h                ; экран символа "O "
 mov dl,'K'             ; занесение в регистр dl и вывод на
 int 21h                ; на экран символа "K"
 mov ah,08              ; задание ОС режима ожидания нажа-
 int 21h                ; тия клавиши
 mov ah,4ch             ; возврат в операционную систему
 int 21h
 code ends              ; конец сегмента кода
 end start              ; конец исходного модуля и указание
                        ; :пускового адреса start
```

В первой строке программы находится директива title (наименование), которая присваивает внутреннее имя объектному модулю, генерируемому ассемблером. Имя модуля summa нельзя путать с именем файла – оно хранится внутри объектного модуля.

Одним из принципиальных моментов является то, что программа состоит из трех различных логических сегментов: DATA. STACK и CODE. Каждый сегмент состоит из директивы SEGMENT и заканчивается директивой ENDS, причем обе директивы для каждого сегмента имеют одинаковые имена. Логические сегменты, естественно, соответствуют физическим сегментам памяти, но привязки их к физическим адресам в исходном модуле нет.

Директива ASSUME (предположить, считать) сообщает ассемблеру, что регистр CS будет содержать базовый адрес сегмента CODE, а регистр DS – базовый адрес сегмента DATA.

Сегмент DATA содержит всего две переменные (VAR_1 и VAR_2), которые определены и инициализированы (установлены в нуль) с помощью директивы DW. Обе переменные имеют тип WORD, поэтому в области данных рассматриваемой программы имеются два слова.

Первая строка в сегменте STACK (стек) содержит директиву DW с операндом 10 DUP (?). Эта директива резервирует 10 слов памяти, но не инициализирует их (не присваивает определенных значений). Таким образом, в программе для стека резервируется 10 слов, иначе говоря, стек имеет глубину 10 слов.

Следующая строка в сегменте STACK содержит директиву LABEL (отметить):

STK_TOP LABEL WORD

Данная директива определяет имя STK_TOP , которое относится к слову, находящемуся после 10 слов, и идентифицирует вершину пока пустого стека. Значение смещения STK_TOP от начала сегмента STACK должно находиться в указателе стека SP, когда стел пустой. Содержимое SP будет уменьшаться на 2 при выполнении каждой команды, включающей слово в стек. Когда стек полностью заполнен (в нем находится 10 слов), содержимое SP равно нулю. Дальнейшее включение слова в стек приведет к неуправляемому поведению системы.

Сегмент кода начинается пятью командами пересылки данных MOV. Они выполняют обязательную инициализацию сегментных регистров DS, SS и указателя стека SP. Имена DATA и STACK идентифицируют базовые адреса сегментов данных и стека. Выражение OFFSET STK_TOP представляет собой значение смещения метки STK_TOP от начала содержащего ее сегмента STACK.

Собственно программа начинается с метки PROG. Программа имеет иллюстрационный характер и выполняет простые действия. Сначала содержимое аккумулятора AX включается в стек. Затем в AX загружается значение переменной VAR_1. Оно увеличивантся на 5, и результат запоминается как значение переменноц VAR_2. После этого в аккумулятор из стека возвращается его старое содержимое. Очевидно, что действия данной части программы описываются простым оператором

VAR_2: = VAR_1 + 5.

Затем идут команды обеспечения режимов взаимодействия прикладной программы с операционной системой при выводе двух символов "O" и "K" на экран с последующим выходом в операционную систему при нажатии любой клавиши.

Последняя строка программы с директивой END, содержащей операнд START, сообщает ассемблеру о достижении конца исходного модуля и необходимости начать выполнение программы с команды, отмеченной меткой START.

Технология создания программ

Используя возможности ОС и системных обрабатывающих программ (текстовых редакторов, трансляторов и др.) можно поместить приведенный в п 1.1 исходный текст программы на языке ассемблера в записанный на диск файл. Такой файл называется **исходными модулем** программы. Файл, содержащий програм-

му, переведенную на машинный язык и готовую к выполнению, называется **загрузочным**, или **исполняемым**, модулем.

Преобразование исходного ассемблерного модуля в загрузочный делается в два этапа:

* исходный модуль обрабатывается программой-транслятором (MASM.EXE или TASM.EXE) и получается объектный модуль с расширением OBJ;

* объектный модуль обрабатывается компоновщиком (одной из программ LINK.EXE, TLINK.EXE), в результате работы которого на диск записывается загрузочный модуль нашей программы.

Чтобы запустить выполнение программы трансляции или компоновки из MS DOS, достаточно ввести с клавиатуры путь к ней и нажать клавишу Enter. Чтобы программа знала, какие данные необходимо обработать, ей при запуске необходимо передать параметры прикладной программы на ассемлере. В данном случае это путь к обрабатываемому файлу, имя файла (например, ABC.ACM), куда нужно записать результат обработки и другие данные. Параметры вводятся при пуске программы как показано ниже.

```
S:\TASM\TASM.EXE S:\WORK\ABC.ASM  /l  /zi
S:\TASM\TLINK.EXE S:\WORK\ABC.OBJ  /v
```

Первая директива запустит транслятор TASM.EXE из подкаталога, который тоже называется TASM, и передаст ему путь к исходному модулю.

Транслятор сформирует модуль ABC.OBJ. Ключ /l в директиве (он может отсутствовать) указывает, что кроме объектного модуля нужен текстовый файл ABC.LST с **листингом** программы, в котором показаны полученные при трансляции машинные коды. В Norton Commander предусмотрена возможность запустить программу обработки файла, установив на его имя курсор и нажав клавишу Enter. Обычно Norton Commander настроен так, что при установке курсора на файл с расширением ASM и нажатии Enter выполняется трансляция программы, а при установке на файл OBJ - компоновка загрузочного модуля. Если это не сделано, для настройки нужно нажать клавишу F9. В верхней строке появится меню, в котором надо выбрать пункт Commands, а в появившемся после этого вертикальном меню - пункт Extension file edit. Далее станет понятно, как связать нужные директивы оператора с файлами, имеющими расширения ASM и OBJ.

Порядок выполнения лабораторной работы

1. Создать на диске каталог, в названии которого указать номер своей группы и бригады (например, R3_1_3).

2. Подготовить в нем файл с исходным текстом программы, приведенным в п.1. Присвоить файлу имя PR1.ASM.

3. Определить, в каком каталоге находятся программы TASM.EXE,TLINK.EXE, и получить с их помощью загрузочный модуль PR1.EXE с листингом программы.

4. Проверить работу модуля PR1.EXE. Представить преподавателю результат работы программы (выдачи сообщения "OK").

Содержание отчета

1. Наименование и цель выполняемой работы.

2. Формулировка и порядок выполнения работы.

3. Формулировка задачи, решаемой программой, и листинг программы с подробными комментариями.

4. Выводы по проделанной работе.

Контрольные вопросы

1. Основные сегменты программы на ассемблере и их взаимосвязь с сегментными регистрами процессора.

2. Директивы определения сегментов и инициализации сегментных регистров.

3. Особенности создания сегмента стека и используемые при этом директивы.

4. Назначение и порядок применения транслятора и компоновщика программы на языке ассемблера.

ЛАБОРАТОРНАЯ РАБОТА № 2
ВЫПОЛНЕНИЕ ПРОГРАММ НА АССЕМБЛЕРЕ С ИСПОЛЬЗОВАНИЕМ ТУРБООТЛАДЧИКА

Цель работы: освоение методов отладки программ на языке ассемблера с пошаговым и автоматическим выполнением ее в среде Турбоотладчика

Теоретические сведения

Для детального изучения работы программы можно использовать Турбо отладчик - программу, хранящуюся в файле TD.EXE. Отладчик позволяет выполнять программу в автоматическом режиме с остановкой при достижении заданной программистом команды, а также в шаговом режиме - с остановкой после каждой команды.

Выполнение программы по шагам позволяет после каждой команды посмотреть, как изменяется содержимое памяти и регистров процессора.

Рассмотрим выполнение под отладчиком программы, выполняющей следующие действия: размещение в памяти строки текста; вывод на экран 18 символов из этой строки, начиная с десятого; ожидание нажатия любой клавиши; по которому завершается работа программы. Ниже приведен исходный текст программы на ассемблере.

```
title stroka
s          segment stack 'stack'
st         db 4 dup ('stack')
s          ends
;          данные
datag      segment para 'data'
n          dw 9
k          dw 18
pr1        db 'Пример1: вывод час'
n1         db 'ти строки на экран '
datag      ends

codeg segment para 'code'
                assume  ds:datag, cs:codeg,ss:s
;                       команды                 другая запись
тех же  действий
```

```
start:    mov  ax,datag            ;   mov ax,seg pr1
mov  ds,ax
 mov  bx,n                 ;   mov bx,[n]
mov  cx,k
mov  ah,2
;  циклический вывод символов
g:          mov  dl,pr1[bx]          ;   mov dl,[bx+offset pr1]
int   21h
inc   bx
dec   cx
jne   g
;  ждать нажатия клавиши
mov  ah,08
int   21h
;  возврат в операционную систему
mov  ax,4c00h
int   21h
codeg     ends
end       start          ;start - метка пусковой точки
```

Запуск отладчика можно выполнить, задав в директиве путь к отладчику и прикладной программе (например, с именем PR2) следующим образом:

S:\TC\TD\TD.EXE S:\R3_1_3\PR2.EXE

После запуска на экране должно появиться изображение, структура которого показана на рис.4.

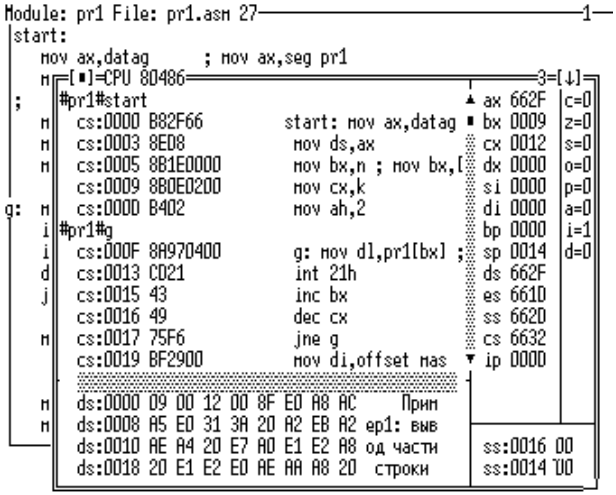

Рис.4.

В верхней строке выводится меню (на рисунке не показано) с перечнем выполняемых отладчиком функций (а точнее, групп функций).

Для выбора пункта из меню следует нажать клавишу с надписью F10. Один из пунктов меню при этом выделится курсором - изменением цвета фона. После этого клавишами со стрелками курсор переводится на нужный пункт и нажимается Enter.

Остальное поле экрана занято перекрывающими друг друга окнами. Окно 1 содержит исходный текст программы. Оно закрыто окном 3, в котором дополнительно изображены поля с содержимым различных компонентов ЭВМ:

в левом верхнем поле показано содержимое оперативной памяти, рассматриваемое как последовательность команд (выводятся адреса и шестнадцатеричное содержимое ОЗУ, а также условное обозначение команды);

в левом нижнем - адреса и содержимое байтов оперативной памяти, рассматриваемое как данные. Заметим, что содержимое каждого байта отладчик выводит шестнадцатеричным кодом (восемь чисел слева) и в виде символа (справа), если символ с таким кодом существует в стандарте ASCII;

справа вверху находятся два вертикальных поля, отображающих значения регистров (общего назначения, сегментных и регистра IP) и состояние флагов (CF,ZF,SF и других);

в правом нижнем углу показано содержимое сегмента стека.

Данный пример иллюстрирует работу с нашей программой. Действительно, в начале сегмента данных, по адресу ds:0000,находится число 9 (отладчик показывает сначала его младший байт 09, потом старший 00. Далее мы видим число 12h и текст:

"Пример1...".

По содержимому регистров видно, что уже выполнены первые четыре команды программы: числа 9 и 18 уже занесены в регистры BX, CX, а число 2 в регистр AH еще не попало.

Одно из полей является активным, в одной из его строк находится курсор - строка, выделенная цветом фона. Переключается активное поле клавишей Tab.

Данные в активном поле программист может легко изменять. Если это поле регистров или данных, достаточно ввести с клавиатуры какое-нибудь 16-разрядное число, и оно запишется в элемент, отмеченный курсором.

Для изменения строки в поле команд нужно набрать ас-

семблерную команду (например, INC [BX]) и нажать клавишу Enter. Однако при этом нужно следить, чтобы число байтов в новой и изменяемой команде было одинаковым. Если число байтов в новой команде меньше, избыточные байты следует заменить однобайтовой командой NOP (нет операции) с кодом 90h.

Некоторые функции отладчика можно выполнять, нажимая функциональные клавиши F1 - F10 (отдельно или в сочетании с клавишей Alt: Alt/F1 - Alt/F10). В нижней строке указано закрепление функций за клавишами: при нажатии F7 выполнится одна (очередная) команда программы, при нажатии F4 программа будет выполняться автоматически, пока не достигнет команды, отмеченной курсором и т.д.

Обычно в поле данных программист высвечивает содержимое сегмента, на который указывает регистр DS. Чтобы указать область памяти, видимую в этом поле, необходимо нажать клавиши Alt/F10. В ответ появится дополнительное, вертикально расположенное меню, в котором выбирается пункт Goto. После этого появится рамка, в которую необходимо вписать нужный полный (то есть, сегмент:смещение) адрес. В данном случае можно ввести DS:0

Порядок выполнения лабораторной работы

1. Создать на диске каталог, в названии которого указать номер своей группы и бригады (например, R3_1_3), подготовить в нем файл с исходным текстом программы, приведенным в п.1. Присвоить файлу имя PR2.ASM.

2. Выполнить трансляцию и компоновку программы, проверить работу модуля PR2.EXE. Представить преподавателю результат работы программы (выдачи сообщения "Вывод части строки").

3. Проверить работу программы под отладчиком в режиме выполнения по шагам (первые 10 шагов) и зафиксировать изменение состояния регистров и данных.

4. Провести выполнение программы под отладчиком в автоматическом режиме с шагом по 3-4 команды и зафиксировать изменение состояния регистров и данных.

5. Произвести необходимые изменения в программе для вывода фамилии каждого члена группы или бригады и преподавателю.

ЛАБОРАТОРНАЯ РАБОТА № 3
ОБРАБОТКА МАССИВОВ ДВОИЧНЫХ И ДЕСЯТИЧНЫХ ЧИСЕЛ

Цель работы: освоение методов адресации элементов структур данных при их арифметической и логической обработке и приобретение навыков составления программ обработки массивов двоичных и десятичных чисел на языке ассемблера.

Теоретические сведения

Основные методы адресации элементов различных структур данных

При микропроцессорном управлении робототехническими системами и другим технологическим оборудованием часто необходимо выполнять арифметическую и / или логическую обработку элементов различных структур данных, отражающих протекание технологического процесса.

Распространенными типами такой обработки являются: определение максимального или минимального из чисел; нахождение среднего значения массива чисел; фильтрация отсчетов контролируемых технологических сигналов от помех и другие виды статистической обработки данных.

Широкий набор методов адресации на языке ассемблера позволяет организовать эффективный доступ к элементам данных при их обработке, а также при выводе результатов обработки данных на экран дисплея или внешнее устройство ввода-вывода для контроля и документирования.

При доступе к регулярным структурам данных применима **косвенная регистровая** адресация, когда эффективный адрес EA операнда находится в одном из базовых или индексных регистров BX, SI или DI. Косвенная адресация через базовый регистр BP моделируется при помощи базовой адресации с нулевым смещением. Косвенный регистровый режим адресации позволяет вычислять адреса во время выполнения программы, что часто требуется при обращении к регулярным структурам данных. При модификации содержимого регистра одна и та же команда оперирует с различными ячейками памяти. Для изменения содержимого регистра применяется команда загрузки эффективного адреса LEA, а также все арифметические и логические команды.

Команда LEA работает аналогично команде MOV, но имеет следующее основное отличие: *она пересылает в регистр - при-*

емник эффективный адрес той ячейки памяти, которую переслала бы команда MOV (с тем же методом адресации источника данных). Так, в следующем примере

MOV DX,TXT

LEA DX,TXT

первая команда занесет в регистр DX содержимое ячейки TXT, а вторая ее адрес. Того же результата можно достичь командой MOV DX, offset TXT , но следует заметить, что в команде пересылки константа offset TXT вычисляется во время трансляции и заносится в код команды, а команда LEA вычисляет адрес в процессе выполнения команды. Поэтому команда

MOV DI, offset [BX+7]

недопустима - транслятор не может знать, какое значение будет занесено в регистр BX во время выполнения программы, а команда

LEA DI, [BX+7]

во время выполнения программы вычислит адрес ячейки памяти, равный сумме содержимого регистра BX и числа 7 и эту сумму (а не содержимое ячейки памяти с адресом BX+7), занесет в регистр DI.

Еще раз: команда LEA *DI, [BX+7] занесет в регистр DI то же, что две команды*

MOV DI,BX

ADD DI,7 .

Вторым операндом команды LEA, естественно, не может быть объект, не имеющий адреса, т.е. регистр или константа. Недопустимы команды

LEA DI, BX

LEA DI, offset TXT .

Базовая адресация с использованием регистра BP является удобным средством обращения к данным, находящимся в стеке, что требуется, например, при передаче подпрограмме параметров через стек. При базовой адресации эффективный адрес операнда EA является суммой значения смещения и содержимого регистра BX или BP. При определении BP в качестве базового адреса шинный интерфейс процессора обращается к операнду в текущем сегменте стека.

Основное применение базовой адресации связано с доступом к элементам структур данных, когда смещение (номер) элемента известно при ассемблировании программы, а базовый (начальный) адрес структуры должен вычисляться при выполнении программы. Таким образом, структура данных может размещать-

ся в различных областях памяти, а модификация содержимого базового регистра обеспечивает доступ к этим областям. Базовый регистр указывает на начало структуры данных, а требуемый элемент адресуется с помощью смещения (расстояния) от базы (см. рис.1).

Смещения, содержащиеся в команде, могут иметь длину 8 или 16 бит и интерпретируются как знаковые числа.

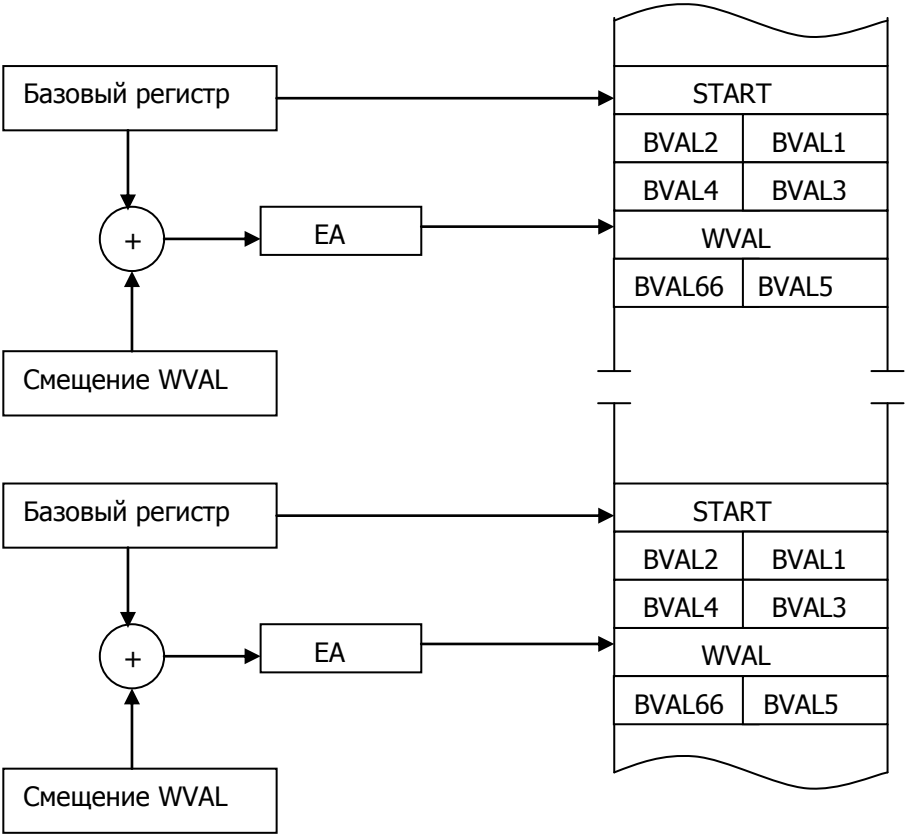

Рис. 1.Применение базовой адресации

В языке ассемблера используются два обозначения базовой адресации, например, обе команды

MOV AX, [BP]10

MOV AX, [BP + 10]

передают шестое слово массива, адресуемого BP.

Для обращения к элементам однородного массива применяется **индексная** адресация, при которой эффективный адрес EA вычисляется как сумма смещения, находящегося в команде, и

содержимого регистров SI или DI. Смещение определяет фиксированный при ассемблировании начальный адрес массива, а значение в индексном регистре определяет нужный элемент. Поскольку все элементы однородного массива имеют одинаковый размер, простые манипуляции над содержимым индексного регистра позволяют обращаться к любому элементу масссива.

Например, команда

MOV ARRAY[SI], AX

осуществляет передачу содержимого AX в элемент массива ARRAY.

Базовая индексная адресация является удобным инструментом доступа к элементам матрицы (двумерного массива) или к элементам массива, находящегося в стеке. При базовой индексной адресации эффективный адрес равен сумме содержимого базового регистра (BP или BX), индексного регистра (SI или DI) и смещения, находящегося в команде. Два варьируемых при выполнении программы компонента адреса делают базовую индексную адресацию наиболее гибким режимом доступа к данным.

При адресации массива, находящегося в стеке (см. рис.2) регистр BP обычно адресует некоторую отсчетную точку в стеке после того, как подпрограмма включила в стек содержимое внутренних регистров микропроцессора и выделила область стека для локальных переменных.

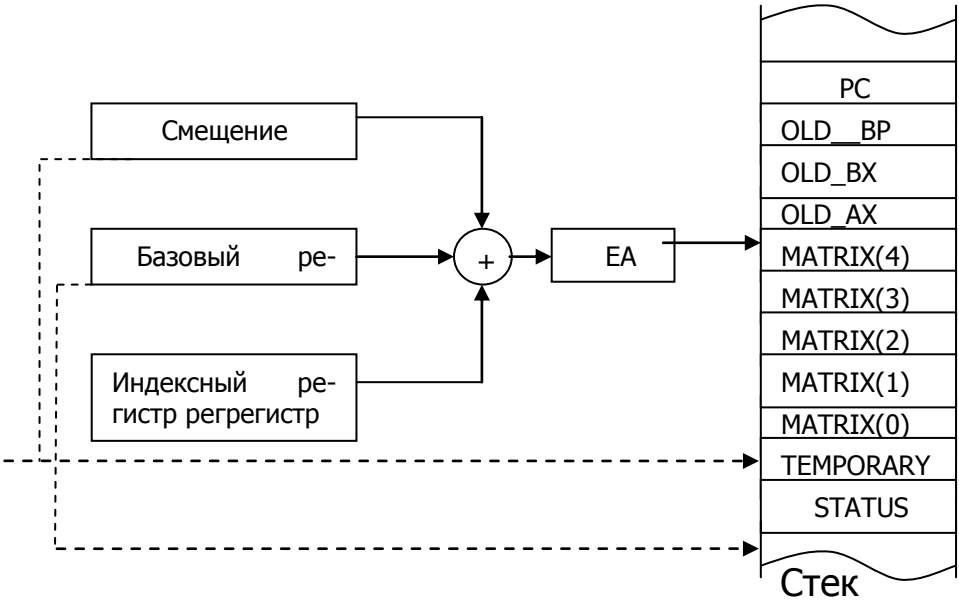

Рис. 2. Обращение к массиву в стеке

Примером обращения к элементу данных в стеке является команда

ADD [BP]MATRIX[SI], DI,

выполняющая сложение элемента массива с содержимым регистра DI.

Арифметическая и логическая обработка массивов

Рассмотрим фрагмент программы нахождения максимального 8-битного беззнакового числа в массиве, адресуемом регистром SI. При этом регистр CX содержит число элементов массива. По окончании программы максимальное значение заносится в регистр AL, а его адрес в массиве размещается в регистре DX.

```
START:   MOV   AL, [SI]    ; загрузка максимума
         MOV   DX.,  SI    ; индекс максимума
NEXT:    DEC   CX          ; проверка окончания поиска
         JZ    DONE
```

```
        INC     SI          ; продвижение указателя
        CMP     AL,[SI]     ; сравнение с максимумом
        JB      START       ; новый максимум
        JMP     NEXT        ; продолжение поиска
DONE:
```

В этом фрагменте программы сначала в качестве максимального принимается первый элемент массива, а затем с ним сравниваются следующие элементы. Если текущий элемент больше ранее найденного максимума, то он замещает его в регистре AL.

Предположим, что в области памяти, адресумой регистром SI, находится цепочка 7- битных кодов символов, заканчивающихся символом NULL. Старшие биты всех кодов символов нулевые. Пусть необходимо записать в старший бит каждого символа контрольный разряд четности, а на место символа NULL – байт продольной четности всего блока символов. Требуемые действия выполняются следующей программой:

```
        XOR   AH,AH       ; сбросить регистр AH
START:  MOV   AL,[SI]     ; передать в AL код символа
        AND   AL,AL       ; установить флажки
        JZ    EXIT        ; символ NULL – конец цепочки
        JPE   NON         ; четное число единиц ?
        OR    AL, 80h     ; нет-записать 1 в старший бит
NON:    MOV   [SI],AL     ; вернуть символ в цепочку
        XOR   AH,AL       ; учесть в контрольном байте
        INC   SI          ; продвинуть указатель
        JMP   START       ; повторить цикл
EXIT:   MOV   [SI],AH     ; записать контрольный байт
```

Рассмотрим программу вычисления средних значений для пяти пар десятичных чисел, размещенных в массиве.

```
title mod_1   ; вычисление массива средних
      assume cs:p_code,ds:p_data
p_data  segment
      means  dw  5 dup (0)   ;массив средних
      pairs    dw    200, 80
               dw  12h, 14h
               dw   65, 1408
               dw  1831, 7
               dw  39, 2508
      count    equ  5
```

```
p_data ends
p_stack segment
       dw 8 dup(?)
  stk_top  label word
p_stack ends
p_code segment
  begin:  mov ax,p_data     ;инициализация
          mov ds,ax         ;сегментных
          mov ax,p_stack
          mov ss,ax         ;регистров
          mov sp,offset stk_top
          push cx
          mov cx,count      ;счетчик пар
          mov si,0          ;индексные
          mov di,si         ;регистры
  next:   mov ax, pairs[si]
          mov bx, pairs[si+2]
          mov dx,ax
          add dx,bx
          rcr dx,1
          mov means[di],dx  ;запомнить
          add si,4          ;продвинуть
          add di,2          ;указатели
          loop next         ;зациклить
          mov ax,means[0]
          mov bx,means[2]
          mov cx,means[4]
          mov dx,means[6]
          pop cx
          mov ah,2
          mov dl,'O'
          int 21h
          mov dl,'K'
          int 21h
          mov ah,08
          int 21h
          mov ah,4ch
          int 21h
p_code  ends
end begin
```

Данная программа размещает вычисленные 5 значений

средних величин в массиве MEANS, длина которого определяется константой count. Регистр CX при этом выполняет роль счетчика циклов, а регистры SI и DI используются для индексирования. Регистр SI содержит индекс пары чисел из массива PAIRS, поэтому при прохождении по циклу осуществляется инкремент SI на 4. Выражение PAIRS[SI] адресует первое число пары, а выражение PAIRS[SI + 2] адресует второе число пары. Регистр DI содержит индекс элемента массива MEANS и увеличивается на 2 при прохождении по циклу.

В регистры AX и BX заносятся усредняемые числа, а определение среднего значения выполняется операцией сдвига вправо их суммы на 1 бит, что эквивалентно делению на 2, но выполняется гораздо быстрее, чем при использовании команды деления DIV.

При вычислении среднего значения для всего массива из нескольких чисел требуется делить их сумму на длину массива, которая в общем случае не равна степени числа 2. Однако часто делитель можно представить в виде суммы целых степеней числа 2.

Порядок выполнения лабораторной работы

1. Составить программу вычисления средних значений семи пар чисел, выполнить ее трансляцию и компоновку.

2. В пошаговом режиме выполнения программы под управлением Турбоотладчика проследить и зафиксировать ход вычисления средних значений для двух наборов исходных данных.

3. Модифицировать исходный текст программы для вычисления среднего значения для всего набора из семи пар чисел и проследить ее выполнение под управлением Турбоотладчика.

Содержание отчета

1. Наименование и цель выполняемой работы.

2. Формулировка и порядок выполнения работы.

3. Листинг и результаты выполнения программы вычисления средних значений семи пар чисел с необходимыми комментариями и пояснениями

4. Листинг и результаты выполнения программы вычисления среднего значения для всего набора из семи пар чисел с необходимыми комментариями и пояснениями

5. Выводы по проделанной работе.

Контрольные вопросы

1. Косвенно-регистровый, базовый и индексный методы адресации
2. Обращение к массиву в стеке с использованием базовой индексной адресации данных
3. Обращение к структурам данных с использованием базовой адресации.
4. Арифметическая и логическая обработка массивов данных

ЛАБОРАТОРНАЯ РАБОТА № 4
ОБМЕН С ВНЕШНИМИ УСТРОЙСТВАМИ ЧЕРЕЗ АДРЕСНОЕ ПРОСТРАНСТВО ЭВМ

Цель работы: изучение и практическое освоение методов вывода текстовой и графической информации на экран дисплея с записью в видеопамять.

Теоретические сведения

Организация вывода информации

В ЭВМ используют два основных способа программного обмена процессора с периферийными устройствами:

1) через адресное пространство ЭВМ (этот способ был единственным в машинах с архитектурой PDP11 - СМ4, Электроника, ДВК);

2) при помощи специальных команд ввода-вывода (так осуществлялся обмен в ЭВМ фирмы "Хьюлетт - Паккард" HP21MX, отечественной СМ2 и др.).

В IBM PC для обмена со стандартной периферией применены оба варианта.

Вывод текстовой и графической информации: **через адресное пространство ЭВМ** предполагает, что за периферийным устройством закреплен определенный участок адресного пространства. Данные, записываемые в такой участок, управляют работой устройства, а чтение информации из этого участка позволяет определить состояние устройства. Например, за экраном дисплея в стандартном текстовом режиме (25 строк,80 символов в строке) закреплена область адресного пространства, начиная с адреса B800:0000. Эта область называется **видеопамятью** (см. рис.1).

На экране дисплея отображаются те символы, коды которых хранятся в видеопамяти. В ней за каждым знакоместом экрана (местом, куда можно вывести символ) закреплено одно слово. Слова с последовательно нарастающими адресами управляют соседними знакоместами экрана, как показано на рис.1

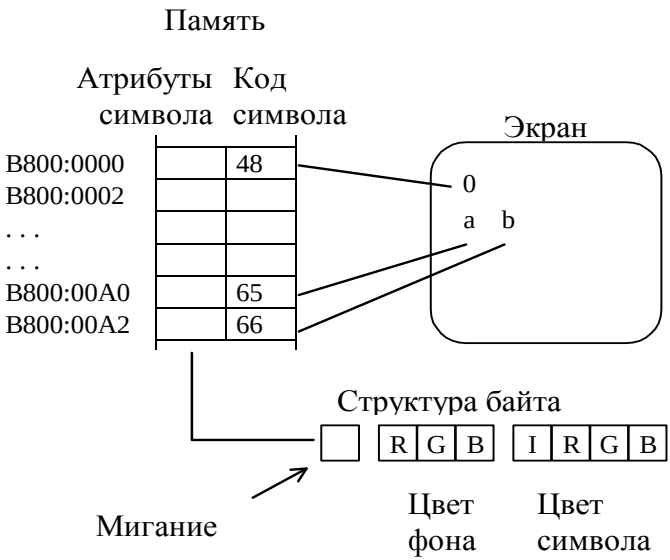

Рис. 1. Соответствие слов в видеопамяти иизображения символов на экране дисплея

Младший байт слова с адресом B800:0000 содержит код того символа, который мы видим в левом верхнем углу экрана. Байт с адресом B800:0002 управляет вторым символом первой строки, байт с адресом B800:00A0 (десятичное число 160) - первым символом второй строки и т.д. Биты старшего байта каждого слова определяют **атрибуты символа**: цвет символа, фона, на котором он нарисован, а также признак мигания символа. Цвет символа задается по системе I R G B –соответственно: интенсивность, красный, зеленый и синий цвета. Единица в битах R, G или B включает соответствующий цветовой луч. Единица в бите I увеличивает яркость цветовых лучей. При включении нескольких лучей цвета смешиваются. Например, при установке в единицу всех четырех битов символ рисуется белым цветом. При задании цвета фона управление интенсивностью не предусмотрено, поскольку при этом бит 7 используется для обеспечения возможности вывода на экран мигающих символов.

Из изложенного ясно, что если по адресу 0B800:0000 записать слово 4741h, то в первой позиции первой строки появится белая буква **A** на красном фоне. Действительно, младший байт слова содержит число 41h (десятичное число 65) - это код буквы **A**. Содержимое старшего байта в двоичном коде имеет вид 0100 0111. Единицы в трех его младших битах, которые отвечают за

цвет символа, указывают, что включены все три луча: красный, зеленый и синий. Код 100 в разрядах 4 - 6 управляет цветом фона. Здесь установлен бит, включающий красный цвет. Рассуждая аналогичным образом, увидим, что код 1020h отобразится в синий прямоугольник.

Закраска экрана цветом

На экране имеется 2000 знакомест, поэтому занесение кода 1020h в 2000 слов, начиная с адреса 0B800:0000, обеспечивает закраску синим цветом всего экрана. Программа закраски экрана имеет следующий вид.

```
title FON_S
code      segment para 'code'
assume cs:code
start:  push di
push cx
mov cx,2000
mov ax,0B800h
mov es,ax
mov di,0
pro:      mov word ptr es:[di],1020h
add di,2
loop pro
pop cx
pop di
code  ends
end  start
```

Для того, чтобы данные попадали в видеопамять, в сегментный регистр ES в начале процедуры заносится код 0B800h. Команда записи в видеопамять mov word ptr es:[di],1020h использует сегментный регистр ES, чтобы не изменять значение DS, который в основной программе обычно указывает на сегмент данных.

Вывод символов в заданную позицию экрана

Рассмотрим программу, которая выводит один символ '0' в заданную позицию экрана: столбец с номером 55h и строку с номером 22h.

```
title SIMWOL
code      segment para 'code'
assume cs:code
```

```
Cd equ  030h
Stb equ 055h
Str equ 022h
Start:  push bp
mov bp,sp
push di
mov ax,0B800h
mov es,ax
mov di,Str
add di,Stb
add di,Stb
mov byte ptr es:[di+1],7
mov ax,Cd
mov es:[di],al
pop di
pop bp
code      ends
end start
```

Процедура получает в стеке три параметра, имеющих обозначения:

Cd - код символа;

Str- адрес в видеопамяти первого символа (начала) строки, в которую выводится символ;

Stb - номер столбца в который выводится символ.

Процедура формирует в регистрах ES:SI адрес слова, соответствующего выбранной позиции экрана. В младший байт этого слова заносится код символа, а в старший - число 7, задающее атрибуты символа.

Рассмотрим приведенную на рис.2 таблицу из 16 строк и 16 столбцов, содержащую символы, коды которых возрастают (построчно) от 0 до 255

По такой таблице легко определить шестнадцатеричный код любого символа, а именно: надо записать (слева направо) сначала шестнадцатеричный номер строки, потом столбца, в которой стоит символ. Так, символ 0 имеет код 30h.

Конфигурация символа, соответствующего на экране заданному коду, не задана в дисплее жестко, а может настраиваться специальной программой.

Поэтому полезно иметь программу, которая выводит на экран такую таблицу. Такая программа с именем Tabl_Simw для решения этой задачи очищает экран процедурой FON_S, а потом

16 раз вызывает процедуру **string**, заполняющую одну строку таблицы. При каждом вызове смещение **Stroka** первого символа строки от начала видеопамяти увеличивается на 160 (длину строки экрана в байтах). Процедура **string** 16 раз обращается к процедуре SIMWOL, наращивая каждый раз код символа Cod и номер столбца Stolb.

Рис 2. Таблица символов

Разработка программы Tabl_Simw является примером применения технологии "снизу вверх", когда сначала пишутся и от-

лаживаются процедуры, а потом вызывающая их программа.

```
title Tabl_Simw
extrn SIMWOL:far,FON_S:far
s       segment stack 'stack'
  db 50 dup ('stack')
s       ends
d       segment para 'data'
Stroka  dw ?
Stolb   dw ?
Cod     dw ?
d       ends
DeltaS  equ 80*2
codeg   segment para 'code'
  assume ds:d,cs:codeg
start:      call far ptr FON_S
  mov cx,16
  mov Stroka,DeltaS*4    ;позиция на экране первой строки
  mov Cod,0
prod:       call string        ;вывод 16-ти символов строки
  add Stroka,DeltaS        ;увеличение адреса начала строки
  loop prod
  mov ah,8
  int 21h
; возврат в операционную систему
k:      mov ah,4ch
  int 21h
string  proc near
  mov dx,16
  mov Stolb,25    ;позиция левого края на экране
pro1:   push Stroka
  push Stolb
  push Cod
  call far ptr SIMWOL
  inc Cod
  add Stolb,2
  dec dx
  jne pro1
  ret
string  endp
codeg   ends
end     start ;start - метка пусковой точки
```

Если записать основную программу и процедуры в файлы

OSN.ASM, VIDEO2.ASM, VIDEO3.ASM, а потом транслировать, то процедуры VIDEO2 и VIDEO3 можно объединить в библиотеку VIDEO.LST директивой

TLIB VIDEO+VIDEO2+VIDEO3,VIDEO.

Содержание библиотеки:

Publics by module
VIDEO2 size = 31
SIMWOL
VIDEO3 size = 26
FON_S

будет записано в файле VIDEO.LST.

Порядок выполнения лабораторной работы

1. Закрасьте правую половину экрана голубым цветом, используя запись в видеопамять кодов "пробел". Предусмотрите в программе гашение экрана и выход в операционную систему по нажатию клавиши.

2. Выведите в заданной преподавателем позиции экрана символ заданного цвета. Предусмотрите в программе гашение экрана и выход в операционную систему по нажатию клавиши.

3. Изучите программу вывода на экран таблицы символов и поясните комментариями работу.

Содержание отчета

1. Наименование и цель выполняемой работы.
2. Формулировка и порядок выполнения работы.
3. Листинг и результаты выполнения программы закраски экрана с необходимыми пояснениями по внесенным изменениям в соответствии с заданием преподавателя.
4. Листинг и результаты выполнения программы вывода символа на экран с необходимыми пояснениями по внесенным изменениям в соответствии с заданием преподавателя.
5. Выводы по проделанной работе.

Контрольные вопросы

1. Размещение символов в памяти и на экране дисплея.
2. Управление цветом фона и цветом символа.
3. Порядок очистки и закраски экрана.
4. Порядок вывода символа на экран в заданную позицию.

ЛАБОРАТОРНАЯ РАБОТА № 5
УПРАВЛЕНИЕ РАБОТОЙ СИСТЕМНОГО ДАТЧИКА ВРЕМЕНИ (ТАЙМЕРА)

Цель работы: изучение и практическое освоение методов обмена информацией с внешними устройствами через порты ввода- вывода на примере управления работой таймера.

Теоретические сведения

При использовании способа обмена информацией с внешними устройствами через порты ввода- вывода внешнее устройство доступно программисту через один или несколько регистров, которые называются **портами**. Порты не представлены в адресном пространстве ЭВМ, т.е. за ними не закреплены никакие адреса памяти

За портом разработчиком устройства может быть закреплен номер от 0 до 65535. Для доступа к портам в архитектуре предусмотрены две команды:

чтение из порта IN AL,DX;

записи в порт OUT DX,AL.

В регистр DX заносится номер порта, а регистр AL содержит данные, которые записываются в порт или читаются из него.

В качестве примера рассмотрим управление работой системного датчика времени – таймера при формировании звука на IBM PC. В состав ЭВМ (рис. 1) входит трехканальный таймер - устройство, каждый канал которого представляет собой счетчик-делитель последовательности тактовых импульсов, поступающих от генератора Г с частотой f_0=1.19318 Мгц. Счетчик может работать на вычитание в двоичном или в двоично-десятичном коде. Загрузка всех нулей в счетчик канала (N =0) дает максимальный интервал при счете (2^{16} или 10^4). Звук формируется сигналом частоты $f_з=f_0/N$, представляющим собой последовательность прямоугольных импульсов и поступающим на динамик с выхода канала через ключ К.

Таймер может работать в одном из шести режимов, задаваемых программно.

В режиме 0 осуществляется выдача сигнала прерывания по конечному числу. В режиме 1 таймер представляет собой ждущий мультивибратор. В режиме 2 реализуется генератор тактовых импульсов. В режиме 3 на выходе таймера формируется прямоугольный сигнал скважности 2. В режиме 4 таймер пред-

ставляет собой формирователь программно-управляемого стробирующего сигнала, а в режиме 6 - формирователь схемотехнически-управляемого стробирующего сигнала.

Для вывода звука в ЭВМ предназначен канал 2, работающий, как правило, в режиме 3. (Канал 1 отсчитывает период регенерации ОЗУ, а канал 0 используется операционной системой для организации службы времени).

Перед началом работы командами

```
mov dx,43h
mov al,10110110b
out dx,al
```

в таймер через порт 43h выводится управляющее слово. Назначение битов этого слова показано на рисунке 1. Потом записью единиц в младшие разряды порта 61h включается второй канал таймера и замыкается ключ К.

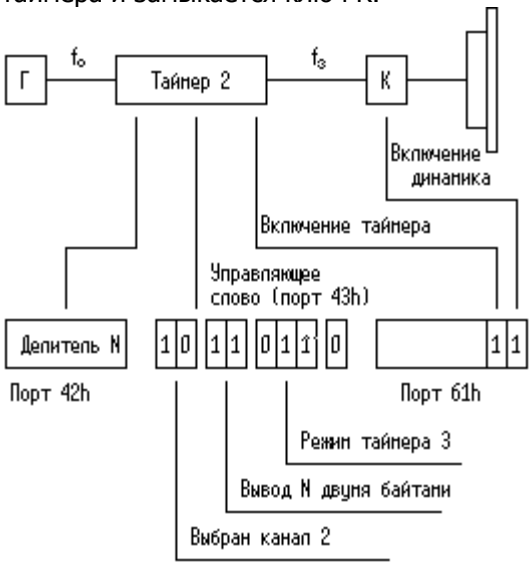

Рис1. Схема канала таймера

Для формирования частоты fs в каждом канале таймера есть два регистра, доступных программисту (в зависимости от номера канала) через порты 40h, 41h, 42h:

* 16-разрядный регистр констант пересчета, который хранит значение коэффициента деления N;
* счетчик, который отсчитывает длину очередного импульса.

Задание требуемого тона звука или его изменение выполняется выводом в закреп- ленный за каналом 2 порт 42h

байта коэффициента деления del = N:

```
mov dx,42h
mov al,byte ptr del
out dx,al
mov al byte ptr del+1
out dx,al
```

Таким образом, коэффициент деления N задается двумя байтами (сначала младший, потом старший), если в битах 4,5 управляющего слова записан код 11. Если эти биты равны 01, выводится только младший, а если 10 - только старший байт (другой байт автоматически обнуляется).

Заметим, что при этом коэффициент деления N занесется в оба указанные выше регистра. После этого в таймере из содержимого счетчика с частотой $2*f_0$ (по заднему фронту или иначе спаду (срезу) импульсов генератора Г) вычитается единица. При обнулении счетчика в него повторно (без участия программиста) записывается коэффициент деления N из регистра констант пересчета и выходной сигнал таймера (который через ключ К подан на динамик) меняется на противоположный (высокий уровень становится низким и наоборот). Таким образом, организована работа таймера в режиме 3. Очевидно, что при этом формируется сигнал, у которого длительность импульса равна длительности паузы (скважность 2).

В режиме 2 единица из содержимого счетчика вычитается с частотой f_0 (только по срезу импульсов генератора Г). Высокий уровень выходного сигнала таймера устанавливается при каждой записи делителя в счетчик и становится низким после первого же вычитания единицы из счетчика. Таким образом, в режиме 2 на выходе таймера формируются короткие импульсы с периодом f_0/N.

Включается звук установкой в единицу двух младших битов порта 61h, без изменения остальных битов. Поэтому необходимо прочитать данные из порта в регистр AL, установить в нем нужные биты, а потом заново записать в порт 61h:

```
in al,dx
or al,00000011b
out dx,al
```

Выключение звука производится сбросом тех же разрядов:

```
mov dx,61h
in al,dx
```

```
and al,11111100b
out dx,al    .
```

Здесь были использованы две логические команды:

* OR - логическое сложение (операция "ИЛИ");
* AND - логическое умножение (операция "И").

Эти команды формируют каждый бит результата в зависимости только от значений одноименных битов операндов. Заметим, что в команде сложения на значение бита суммы влияет также перенос из соседнего разряда. Для логических команд можно указывать (как это сделано на рис. 2) правила вычисления результата только для одного бита. Остальные обрабатываются также.

Как видим, результат операции "ИЛИ" равен единице, если хотя бы один из операндов равен единице. Результат операции "И" единица, только если оба операнда равны единице.

Нетрудно убедиться на числовых примерах, что команда OR AL,11b устанавливает младшие разряды регистра AL, не изменяя остальных, а команда AND AL,11111100b сбрасывает два младших бита и также не изменяет остальные биты регистра AL.

Биты операндов		Бит результата
0	0	0
0	1	0
1	0	0
1	1	1

Команда AND

Биты операндов		Бит результата
0	0	0
0	1	1
1	0	1
1	1	1

Команда OR

Рис 2. Правила выполнения операций OR, AND

Порядок выполнения лабораторной работы

1. Изучить последовательность действий по включению и выключению звука нужного тона, используя Турбоотладчик. Для этого необходимо в среде Турбоотладчика командой Ctrl+I выйти в окно локального меню обмена с портами ввода-вывода и в режиме вывода байта последовательно выдать по адресу 43h байт управляющего слова (B6h), по адресу 42h – два байта ко-

эффициента деления N и по адресу 61h – байт включения звука (3fh) Выключение звука осуществляется выдачей байта 3ch.

2. Составить программу, которая включает звук, ждет нажатия клавиши и по нажатию любой клавиши выключает звук.

3. Составить и продемонстрировать работу программы вывода мелодии, выбранной студентом (например, ми-до, ми-до, фа-ми-ре или до-ре-ми-до-ре-до). Частоты (в герцах) звуков первой октавы, начиная с ноты ДО таковы: 523.3, 587.3, 659.3, 698.5, 784.0, 880.0, 987.7. Следующая октава получается удвоением этих частот.

Содержание отчета

1. Наименование и цель выполняемой работы.
2. Формулировка и порядок выполнения работы.
3. Листинг программы включения и выключения звука с необходимыми комментариями.
4. Листинг программы вывода мелодии с необходимыми пояснениями.
5.
6. строки символов на экран дисплея с необходимыми пояснениями по полученным результатам.
7. Выводы по проделанной работе.

Контрольные вопросы

1. Структурная схема и режимы работы программируемого таймера.
2. Формат управляющего слова задания режима работы таймера.
3. Порядок задания работы таймера в режимах 2 и 3.
4. Порядок измерения времени выполнения программы при чтении содержимого счетчика" на лету".

ЛАБОРАТОРНАЯ РАБОТА № 6
ОРГАНИЗАЦИЯ ПРОЦЕДУР И ТЕХНОЛОГИЯ МОДУЛЬНОГО ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ АССЕМБЛЕРА

Цель работы: освоение методов организации процедур и библиотеки объектных модулей при разработке и отладке больших программ на языке ассемблера.

Теоретические сведения

Организация процедур

При разработке больших программ многие авторы рекомендуют применять последовательность разработки "сверху вниз". При использовании такой технологии вся задача разбивается на укрупненные функции, например ввод строки в память (функция BBOD_STR),вывод строки на экран (функция BYBOD_STR) и т.п. Программа сначала пишется так, как будто в системе команд машины есть возможность выполнить каждую функцию одной командой. Сымитировать потом такую укрупненную команду можно, используя макрокоманды (с ними мы познакомимся позднее) и команды вызова процедуры CALL.

Программы, реализующие выделенные программистом укрупненные функции, называются процедурами, или подпрограммами. Они пишутся и размещаются отдельно от текста основной программы: выше или ниже основного текста, в отдельном сегменте или даже в отдельном файле.

Команда CALL передает управление первой команде процедуры аналогично безусловному переходу JMP, но как только в тексте процедуры встретится команда RET (возврат из процедуры), управление возвращается в основную программу на команду, следующую после CALL.

Например, фрагмент основной программы, который вводит с клавиатуры строку символов, а потом десять раз выводит ее на экран, может иметь следующий вид:

```
cod         segment
start:
                .
                .
                .
                call BBOD_STR
                mov cx,10
```

```
z:          call BYBOD_STR
vv:         dec cx
            jne z
            .
            .
            .
cod       ends
  end     start
```

При вызове процедуры можно использовать такие же методы адресации, как в команде безусловного перехода JMP. Команда JMP с прямым методом адресации может состоять из двух, трех или пяти байтов. В трехбайтовой команде, которая называется ближним (NEAR) переходом, первый байт E9h является кодом операции, а два последних указывают адрес перехода - адрес команды, которая выполнится следующей после JMP. При выполнении команды перехода они занесутся в IP. В пятибайтовой команде, которая называется дальним (FAR) переходом, первый байт EAh является кодом операции, два следующих заносятся при выполнении команды в регистр IP, а еще два - в регистр CS. В языке ассемблера все варианты команды имеют обозначение JMP. Адресная часть команды, кроме имени - метки команды, на которую осуществляется переход, может содержать явное указание типа перехода:

```
JMP FAR PTR M1 дальний переход на M1;
JMP NEAR PTR M1 ближний переход на M1.
```

Безусловный переход называется **косвенным**, если адрес, на который передается управление, не является частью кода команды, а задан (как слово или двойное слово) в отдельной ячейке памяти или формируется с использованием регистров общего назначения. В записи команд косвенного перехода можно использовать *обозначения рассмотренных выше регистрового, прямого и косвенных методов адресации*. Но число, которое в обычной команде (сложения, пересылки и т.п.) было бы операндом, при выполнении команды JMP рассматривается как *адрес* перехода. Примеры косвенных переходов показаны ниже.

Команда JMP AX передает управление на метку, адрес которой занесен в регистр AX. Регистр AX шестнадцатиразрядный, поэтому при регистровой адресации возможны только ближние переходы.

Если адрес передачи управления читается из памяти, тогда переход может быть как ближним, так и дальним. Команды косвенного и прямого перехода одинаковы по форме записи (например, JMP M1), но имеют разные коды операции и выполняются по-разному: если метка M1стоит перед командой - это прямой переход, если M1 - объявление слова или двойного слова, формируется ближний или дальний косвенный переход (т.е. ячейка M1 содержит адрес перехода, и при выполнении команды содержимое переменной M1 заносится в IP или в CS:IP).

В команде JMP WORD PTR [BX] регистр BX хранит *адрес* слова, в котором записан *адрес* ближнего перехода. Выполнение команды дальнего перехода JMP DWORD PTR [BX] отличается тем, что BX хранит адрес двойного слова. Итак, в данном случае регистр хранит **адрес адреса** перехода.

Процедура BYBOD_STR, выводящая на экран символы, начиная с метки PR1 и до появления кода клавиши Enter (числа 13), может быть реализована так:

```
                BYBOD_STR:
                 mov bx,0
                 mov ah,2
        ; вывод символов
        g:       mov dl,pr1[bx]
                 int 21h
                 inc bx
                 cmp dl,13
                 jne g
                 ret
```

Рассмотрим выполнение команд CALL, RET подробнее. Они также обращаются к стеку. Команда CALL BYBOD_STR может быть эквивалентна следующим трем командам:

```
  push seg vv            ;текущее CS - в стек
  push offset vv         ;текущее IP - в стек
  jmp far ptr BYBOD_STR
        vv: . . .
```

или двум командам

```
  push offset vv         ;текущее IP - в стек
  jmp near ptr BYBOD_STR
        vv: . . .
```

Таким образом., команда CALL очень похожа на команду JMP и осуществляет безусловный переход на указанную в команде метку. Аналогично командам перехода предусмотрены вызовы дальней и ближней процедуры. В первом случае адресная часть

команды задает двумя словами новые значения регистровой пары CS:IP, во втором - только IP.

Но есть и важное **отличие**. При дальнем вызове перед выполнением перехода в стек записываются, в указанном выше порядке, сегмент и смещение следующей после CALL команды (в данном примере она отмечена меткой vv), а при ближнем - только смещение (значение регистра IP).

Выполнение команды RETF (возврата из дальней процедуры) заключается в чтении двух слов из стека и занесении их в регистры CS:IP, а возврат из ближней процедуры RETN заключается в чтении слова из стека и занесении его в регистр IP. Очевидно, что дальнему вызову процедуры должна соответствовать команда RETF, а ближнему - RETN. Тип вызова и возврата может указываться программистом явно (CALL FAR PTR, CALL NEAR PTR, RETF, RETN), но в языке предусмотрены средства правильного выбора типа по умолчанию.

Каждая процедура начинается строчкой, содержащей имя процедуры, служебное слово PROC и тип процедуры (NEAR, FAR), а заканчивается строчкой, содержащей то же имя и слово ENDP.

```
BYBOD_STR  PROC  NEAR
. . .
BYBOD_STR  ENDP
```

В этом случае в тексте процедуры можно использовать команду RET, которая автоматически, в зависимости от типа процедуры, заменится кодом команды RETN или RETF. Правильный тип вызова такой процедуры также установится транслятором по умолчанию.

Заметим, что имя процедуры играет роль обычной метки (аналогично метке, заданной директивой LABEL), поэтому перед первой командой процедуры она повторно не ставится.

Исходные тексты основной программы и процедур могут находиться в разных файлах. В этом случае каждый файл транслируется отдельно и получается несколько объектных OBJ-файлов. Полученные объектные модули объединяются в один исполняемый модуль компоновщиком (TLINK или LINK). Для этого директиве нужно указать имена всех объединяемых модулей, например, так

```
TLINK OSN+BBOD_STR+BYBOD_STR;
```

Если программа получается объединением нескольких модулей, в исходном тексте появляются обращения к переменным и

переходы к меткам, отсутствующим в данном модуле, они называются **внешними**.

Метка BBOD_STR отсутствует в основной программе, переменная PR1 не объявлена в процедуре и т. п. Для того, чтобы транслятор и компоновщик не воспринимали это как ошибки, в исходном тексте директивами EXTRN перечисляются имена внешних переменных и меток с указанием их типов: EXTRN BBOD_STR:NEAR или EXTRN PR1:BYTE .

В тех файлах, где эти имена объявлены, они должны быть перечислены как входные точки директивой PUBLIC:

```
PUBLIC BBOD_STR
PUBLIC PR1
```

С учетом этих замечаний исходный текст процедуры BYBOD_STR, хранящейся в отдельном файле PROC.ASM, будет иметь следующий вид:

```
title       BYBOD
            EXTRN pr1:byte
            PUBLIC BYBOD_STR

cod         segment
            assume cs:cod

BYBOD_STR PROC FAR
  mov bx,0
  mov ah,2
; вывод символов
 g:         mov dl,pr1[bx]
  int 21h
  inc bx
  cmp dl,13
  jne g
  ret
BYBOD_STR ENDP
cod       ends
  end
```

Несмотря на то, что сегменты кода в основной программе и процедуре имеют одинаковое имя COD, они являются разными адресными элементами сегментами: смещение в основной программе отсчитывается от метки start, а в процедуре - от метки BYBOD_STR. Поэтому процедура должна запускаться дальним

вызовом, что обеспечивается указанием ее типа FAR в процедуре (BYBOD_STR PROC FAR) и основной программе (EXTRN BYBOD_STR:FAR).

Если при объявлении сегмента указывать служебное слово PUBLIC (COD SEGMENT PUBLIC), то при компоновке сегменты разных модулей с одинаковыми именами объединятся в один сегмент. При этом для обращения к процедурам можно будет воспользоваться ближним вызовом.

При объявлении сегмента можно дополнительно указывать его класс: словом 'data' сегмент данных, 'code' - сегмент кода. В этом случае компоновщик сгруппирует сегменты из разных файлов так, чтобы сегменты одного класса размещались в памяти подряд, а сегменты класса 'stack', даже с разными именами, будут объединены в один сегмент, т.к. существует только один указатель SS:SP на вершину стека.

Передача параметров процедурам

Процедура BYBOD_STR имеет существенный недостаток - она выводит только ту строку, которая отмечена именем PR1. *Если в основной программе строка обозначена другим именем или необходимо выводить тексты из разных строк, процедурой воспользоваться не удастся.*

Адрес начала выводимой строки можно передать процедуре из основной программы в качестве параметра, например, в регистре BX. Вызов процедуры при этом будет иметь вид: call BYBOD_STR , а в процедуре надо удалить директиву EXTRN и заменить команду

mov dl,pr1[bx] командой mov dl,[bx] .

Параметры (значения переменных и их адреса), можно также передавать процедуре через стек или в ячейках памяти, следующих за командой CALL.

Рассмотрим типичные варианты на простейшем примере вычисления суммы двух чисел: SUM=A+B. Ниже приведен текст программы и рисунки, показывающие состояние стека и содержимое регистров sp, bp после обращения к процедуре - (рис.1,а), запоминания второй командой процедуры указателя стека в регистре bp - (рис.1,б), запоминания регистра bx (рис.1,в).

```
b   equ  [bp+8]
d   segment 'data'
s1   dw     ?
s2   dw     ?
sum dw     ?
d   ends
c   segment 'code'
.
.
.
push s2
push s1
push offset sum
call summa
add sp,6
.
.
.
mov ah,4ch
int 21h

summa proc near
push bp
mov bp,sp
push bx
mov ax,b
add ax,[bp+6]
;получение адреса sum
mov bx,[bp+4]
mov [bx],ax
pop bx
pop bp
ret
summa endp
```

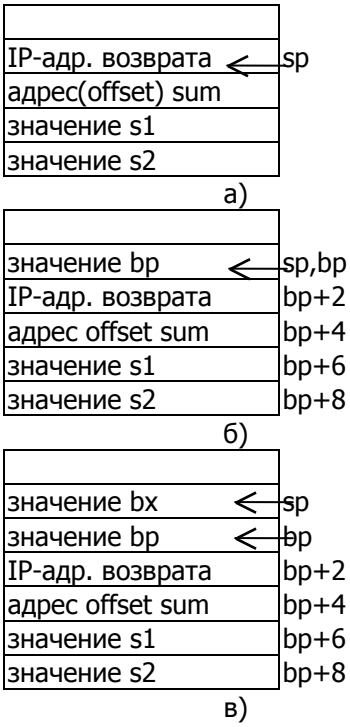

Рис.1 Размещение данных в стеке

Перед обращением к процедуре в стек записываются значения слагаемых и адрес ячейки SUM, в которую процедура должна записать сумму. При выполнении команды CALL в стек попадает также текущее значение регистра IP - адрес возврата из процедуры. (Так как тип процедуры NEAR, содержимое сегментного регистра в стек не пишется). Содержимое регистра SP в этот момент указывает на ячейку стека с адресом возврата из

процедуры в основную программу.

Во время работы процедур возникает необходимость использовать регистры процессора, в нашей процедуре это AX,BX и BP. Но основную программу писать и отлаживать значительно удобнее, если процедура не изменяет значения регистров. Поэтому принято в начале процедуры значения тех регистров, которые будут изменяться, запоминать в стеке, а перед возвратом из процедуры восстанавливать их из стека.

Первыми двумя командами процедуры содержимое регистра BP запоминается в стеке, и в этот регистр копируется указатель стека. Состояние стека и содержимое SP, BP в этот момент отображается рис.1,б. При дальнейших обращениях к стеку содержимое SP изменяется, а BP - нет. Это позволяет, как видно из рис.1,в, в любом месте процедуры прочитать переданные процедуре параметры по адресам [BP+4], [BP+6], [BP+8]. (Напомним, что при косвенной адресации через регистр BP физический адрес определяется парой SS:BP.)

Обращение к параметрам процедуры по их адресам в стеке может стать источником ошибок в программе, так как в большой и разветвленной процедуре легко перепутать, скажем, [BX+4] и [BX+6]. В этом случае за параметрами можно закрепить любые удобные для запоминания имена директивой EQU. Строка **b equ [bp+8]** в примере программы требует, чтобы везде в тексте программы транслятор автоматически вместо имени **b** подставил перед трансляцией [bp+8].

Отметим, что основная программа передает процедуре адрес переменной sum, процедура читает его в регистр BX и, используя косвенную адресацию, записывает в sum командой mov [bx],ax результат сложения. Передача процедуре слагаемых производится записью в стек их значений, а не адресов, поэтому чтение данных из стека не позволяет процедуре изменять содержимое переменных s1,s2. Первый вариант называют передачей параметров **по ссылке**, а второй - **по значению**.

Перед выходом из процедуры содержимое регистров BX, BP восстанавливается из стека, команда RET читает из стека адрес возврата, но после перехода в основную программу параметры процедуры еще остаются в стеке. Команда add sp,6 удаляет их из стека, так как переставляет указатель sp аналогично трем последовательным командам POP. Команда RET может иметь непосредственно заданный операнд, который указывает количество байтов, которое будет дополнительно прочитано из стека при выходе из процедуры. Если процедуру SUMMA закончить коман-

дой RET 6, то при ее выполнении из стека прочитается не только адрес возврата, но и хранящиеся в стеке параметры процедуры. При этом в основной программе следует удалить команду очистки стека add sp,6.

Заметим, что при реализации процедур язык Си использует первый вариант очистки стека от параметров (в основной программе), а язык Паскаль - второй (в процедуре).

Порядок выполнения лабораторной работы

1. Составить (без использования процедур) программу ввода строки символов в область памяти (буфер) из двадцати байтов, зарезервированную директивами

```
STR1            db 20          ; размер буфера
   db 20 dup (?)  ; резервирование 20 байтов
```

и вывода ее на экран. Ввод должен заканчиваться по нажатию клавиши Enter (код клавиши 13) или по заполнению буфера. Размер выделенной области памяти задан значением ее первого байта.

2. Оформить ввод и вывод строки в виде процедур BBOD_STR, BYBOD_STR. Для того, чтобы процедуры ввода и вывода работали, в зависимости от обращения к ним, с разными строками, организовать передачу процедуре адреса строки через стек.

3. Зарезервировать в памяти два буфера для хранения строк STR1, STR2, имеющих длину по 15 байтов. Используя процедуры BBOD_STR, BYBOD_STR ввести с клавиатуры строку STR1, потом STR2, ожидать нажатия любой клавиши, после чего вывести на экран введенные строки в обратном порядке: сначала STR1, потом STR2. (Замечание: чтобы строка STR2 при выводе не затирала на экране ранее выведенную строку STR1, в начале или конце процедуры BYB_STR следует вывести на экран управляющий код 10:"перевод строки".). При вводе данных попытаться ввести строку более, чем из 15 символов и показать, что процедура правильно ограничивает длину строки. Разделить основную программу и процедуры на три файла: основная программа, процедура ввода, процедура вывода.

Содержание отчета

1. Наименование и цель выполняемой работы.
2. Формулировка и порядок выполнения работы.

3. Листинг и результаты выполнения программы ввода строки с необходимыми пояснениями
4. Листинг и результаты выполнения программы вывода строки на экран с необходимыми пояснениями
5. Листинг и результаты выполнения основной программы ввода и вывода на экран строки с необходимыми комментариями
6. Выводы по проделанной работе.

Контрольные вопросы

1. Процедуры (подпрограммы), команды вызова подпрограммы и возврата из подпрограммы
2. Изменение состояния стека при вызове подпрограммы.
3. Порядок создания библиотеки объектных модулей.

Передача параметров процедурам по ссылке и по значению.