МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФГБОУ ВО «ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ И.С.ТУРГЕНЕВА»

Кафедра «Программная инженерия»

**Отчет**

По лабораторной работе №5  
на тему:  
“Использование подпрограмм. Сортировка массива чисел”  
по дисциплине: “Архитектура ЭВМ и систем”

Работу выполнил:  
Студент группы 01-ИТ

Колчев Д.Н.

Проверил:  
Доцент кафедры  
программной инженерии

Конюхова О.В.

Орёл, 2020

1. **Структура программы**

s\_s segment  
 dw 5 dup(0)  
s\_s ends

d\_s segment  
 mus1 db 3, 5, 1, 8, 9, 4, 2, 7, '$'  
 mus2 db 8 dup(0)  
d\_s ends

c\_s segment  
assume ds:d\_s, cs:c\_s, ss:s\_s

; Процедура ссортирует массив  
sort proc near

;Очищаем все регистры  
 xor ax, ax  
 xor bx, bx  
 xor cx, cx  
 xor dx, dx

;Передаём число эл-в в массиве (N-1)  
 mov cl, 8  
 sub cl, 1   
  
 ;Производим перемещение всех пар элементов  
 i:  
 mov si, 0  
 mov al, cl

;Сравниваем текущую пару, перемещаем 1 эл.  
 j:  
 xor dx, dx  
 xor bx, bx  
  
 ; Ищем большее число среди bl и dl  
 mov bl, mus2[si]  
 mov dl, mus2[si+1]  
 cmp bl,dl  
 ja yes  
 jmp no

;Элемент больше, меняем местами  
 yes:  
 ;Меняем местами через доп. регистр bh  
 mov bh, bl  
 mov bl, dl  
 mov dl, bh  
 mov bh, 0

;Помещаем отсортированные эл-ты в mas2  
 mov mus2[si],bl  
 mov mus2[si+1],dl  
 jmp jExit

;Элемент меньше, оставляем на местах  
 no:  
 ;Но всё равно помещаем в mas2  
 mov mus2[si],bl  
 mov mus2[si+1],dl  
 jmp jExit

;Рассматриваем следующую пару  
 jExit:   
 add si, 1  
 Loop j

xor cx, cx  
 mov cl, al  
 ;sub cl, 1  
 xor ax, ax  
 Loop i

ret  
sort endp

check proc near

;Очищаем все регистры  
 xor ax, ax  
 xor bx, bx  
 xor cx, cx  
 xor dx, dx

;Создаём цикл для просмотра 4 пар эл-в  
 mov cl, 4  
 mov si, 0  
 m1:  
 ;Передаём в 2 регистры пару элементов mas  
 mov al,mus2[si]  
 mov bl,mus2[si+1]

;Нулируем для просмотра следующей пары  
 xor ax, ax  
 xor bx, bx  
 add si, 2  
 Loop m1

Ret  
check endp

copy proc near

;Очищаем все регистры  
 xor ax, ax  
 xor bx, bx  
 xor cx, cx  
 xor dx, dx

mov si, 0  
 mov cx, 8

;В цикле, копируем массив  
 cop:  
 ;Копируем черех регистр dx  
 mov dl, mus1[si]  
 mov mus2[si], dl  
 add si, 1  
 Loop cop

Ret  
copy endp

;Вызов функции в начале программы  
begin:  
mov ax, d\_s;   
mov ds, ax  
xor ax, ax

;Копируем старый массив на место нового  
 call copy

;Сортируем числа, занося во 2-й массив  
 call sort

;Прогоняем по регистрам отсортированный массив.  
 call check

mov ah, 4ch  
int 21h  
c\_s ends  
end begin

1. **Контрольные вопросы:**
2. **Описание процедур. Варианты размещения процедур в программе.**

Процедура ограничивается операторами PROC и ENDP, перед которыми указывается имя процедуры. После PROC указывается тип процедуры: процедура ближнего вызова (директива NEAR) или процедура дальнего вызова (директива FAR). В первом случае процедура располагается в том же сегменте кода, что и основная программа, и при вызове такой процедуры выполняется внутрисегментный переход. Во втором случае процедура располагается в другом сегменте кода, и при её вызове выполняется межсегментный переход.  
 Между этими операторами располагается тело процедуры, состоящее из команд и директив языка ассемблера. Последней командой процедуры является команда RET, по которой осуществляется возврат из данной процедуры в вызвавшую её программу или другую процедуру на команду, следующую за командой последнего вызова процедуры.  
 Процедура может размещаться в любом месте программы, но так, чтобы на нее случайным образом не попало управление. В этом случае процессор воспринимает процедуру как часть исполняемого потока команд и начинает их выполнять. Процедура может размещаться:  
 1) В начале программы (до первой исполняемой команды).  
 2) В конце программы (после команды корректного завершения работы и возвращения управления операционной системе – ОС).  
 3) Внутри тела программы или другой процедуры (должен быть предусмотрен обход процедуры с помощью оператора JMP).  
 4) В другом модуле – часто используемые процедуры выносятся в отдельный файл, который оформляется как обычный файл ассемблера, а затем подвергается трансляции для получения объектного кода. Впоследствии этот объектный файл с помощью компоновщика можно объединить с файлом, в котором все или некоторые данные процедуры используются.

1. **Процедуры и сопрограммы. Особенности передачи управления при вызове процедур и при вызове сопрограмм.** В обычной последовательности вызовов существует чёткое различие между вызываемой и вызывающей процедурами. Вызываемая процедура каждый раз начинается сначала, сколько бы раз не происходило обращение к ней. Для выхода из вызываемой процедуры используется команда возврата RET, как описано в предыдущем пункте. Пусть имеются две процедуры: A и B, каждая из которых вызывает другую в качестве процедуры. При возврате из B к A процедура B совершает переход к тому оператору, следующему за командой вызова процедуры B. Когда процедура A передаёт управление процедуре B, она возвращается не к самому началу B (за исключением первого раза), а к тому месту, где произошёл предыдущий вызов A. Процедуры, работающие подобным образом, называются сопрограммами.
2. **Команды вызова процедуры и возврата из неё?** Вызов процедуры осуществляется командой CALL. Пример:

call far ptr quicksort  
 По команде RET осуществляется возврат из данной процедуры в вызвавшую её программу или другую процедуру на команду, следующую за командой последнего вызова процедуры.

1. **Механизмы обработки процедур ближнего и дальнего вызовов. Что представляет собой «адрес возврата» и где он размещается?** Процедурный механизм базируется на командах вызова процедуры, обеспечивающих переход из текущей точки программы к начальной команде процедуры, и командах возврата из процедуры для возврата в точку, непосредственно расположенную за командой вызова. Для работы с процедурами используется стек (дополнительная память, организованная в виде очереди), в который команда вызова помещает текущее значение счётчика команд (IP) при внутрисегментных переходах (или значения регистров IP и CS при межсегментных переходах) – адрес точки возврата. При выходе из процедуры старые значения соответствующих регистров восстанавливаются из стека.  
    При выполнении вызова процедуры PR1 (команда call pr1) в стек помещается адрес возврата – значение счётчика команд IP, содержащего на данный момент адрес команды, которая должна будет выполняться после текущей (mov ah,4ch). Значение регистра IP замещается новым значением – адресом первой команды процедуры. При достижении команды возврата из процедуры (ret) из стека в регистр IP записывается старое значение, что обеспечивает возврат в основную программу на команду mov ah,4ch, которая непосредственно следует за командой вызова процедуры.
2. **Обязательно ли наличие сегмента стека в программе, содержащей процедуры, и почему?** Для работы с процедурами используется стек (дополнительная память, организованная в виде очереди), в который команда вызова помещает текущее значение счётчика команд (IP) при внутрисегментных переходах (или значения регистров IP и CS при межсегментных переходах) – адрес точки возврата. При выходе из процедуры старые значения соответствующих регистров восстанавливаются из стека.  
    Если стека нет, он создаётся системой автоматически.
3. **Сопрограммы. Принципы взаимодействия сопрограмм.** Обычно сопрограммы используются для того, чтобы производить параллельную обработку данных на одном процессоре. Обычные команды CALL и RET не подходят для вызова сопрограмм, поскольку адрес для перехода берётся из стека, как и при возврате, но, в отличие от возврата, при вызове сопрограммы адрес возврата помещается в определённом месте, чтобы в последующем к нему вернуться. Для этого сначала необходимо вытолкнуть старый адрес возврата из стека и поместить его во внутренний регистр, затем поместить счётчик команд IP в стек и, наконец, скопировать содержимое внутреннего регистра в счётчик команд. Поскольку одно слово выталкивается из стека, а другое помещается в него, состояние указателя стека не меняется.