**Федеральное государственное образовательное учреждение   
высшего образования**

**«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИМЕНИ И.С. ТУРГЕНЕВА»**

**Институт приборостроения, автоматизации и информационных технологий**

**Кафедра информационных систем и цифровых технологий**

Отчет к лабораторной работе № 5

«Использование подпрограмм. Сортировка массива чисел»

Выполнила:

Студентка группы 21ПГ

Банных Мария Алексеевна

Приняла:

доцент кафедры   
информационных систем   
и цифровых технологий  
Конюхова Оксана Владимировна  
  
доцент кафедры   
информационных систем   
и цифровых технологий  
Амелина Ольга Викторовна

Орёл 2022 год

Ответы на контрольные вопросы:

*1. Описание процедур. Варианты размещения процедур в программе.*

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

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

*2. Процедуры и сопрограммы. Особенности передачи управления при вызове процедур и при вызове сопрограмм.*

Процедура (подпрограмма) – это группа команд для решения конкретной подзадачи, обладающая средствами получения управления из точки вызова задачи более высокого уровня и возврата управления в эту точку. Это правильным образом оформленная совокупность команд, которая, будучи однократно описана, может быть вызвана в любом месте программы. Но, в отличие от команд перехода, после выполнения процедуры управление возвращается к команде, следующей за командой вызова процедуры.

Структура — это тип данных, состоящий из фиксированного числа элементов разного типа.

При передаче управления процедуре процессор автоматически записывает в вершину стека два (для ближнего вызова) или четыре (для дальнего вызова) байта – адрес возврата в вызывающую программу. Если предварительно в стек были записаны переданные процедуре параметры или указатели на них, то они окажутся под адресом возврата.

*3. Команды вызова процедуры и возврата из неё.*

Вызов процедуры осуществляется командой CALL, за которой следует имя процедуры. Формат команды: CALL: CALL Модификатор имя\_процедуры

Командой RET осуществляется возврат из данной процедуры в вызвавшую её программу или другую процедуру на команду, следующую за командой последнего вызова процедуры.

*4. Механизмы обработки процедур ближнего и дальнего вызовов. Что представляет собой «адрес возврата» и где он размещается?*

Вызов процедуры, или переход с возвратом: CALL [модификатор] <имя процедуры>

Команда записывает адрес следующей за ней команды в стек, а затем осуществляет переход по метке <имя процедуры>. Адрес следующей выполняемой команды задается парой CS:IP. Если процедура описана как:

* дальняя, то в стек по очереди заносятся два значения CS, IP.
* короткая – только значение IP

Адрес возврата в вызывающую программу хранится в стеке.

Команда RET [число] возвращает управление вызывающей программе. Она считывает адрес возврата из вершины стека, загружает его в регистры CS и IP (теперь выполняться будет следующая за CALL команда в программе) адрес возврата при этом удаляется из стека, затем стек очищается на указанное число байт и выполняется переход по адресу возврата.

Команда RET соответствует команде RET 0

В зависимости от того, дальняя или ближняя была описана процедура, ассемблер формирует одну из возможных команд RET. В первом случае из стека извлекается два слова, которые загружаются в регистры CS и IP, во втором - одно слово в регистр IP.

*5. Обязательно ли наличие сегмента стека в программе, содержащей процедуры, и почему?*

Если в программе используются процедуры, то в ней описывается сегмент стека для резервирования ячеек под дополнительную память. Если сегмент стека в программе отсутствует, то в этом случае операционная система формирует стек самостоятельно.

*6. Сопрограммы. Принципы взаимодействия сопрограмм.*

По определению структура — это тип данных, состоящий из фиксированного числа элементов разного типа. Для использования структур в программе необходимо выполнить три действия:

* Задать шаблон структуры. По смыслу это означает определение нового типа данных, который впоследствии можно использовать для определения переменных этого типа.
* Определить экземпляр структуры. Этот этап подразумевает инициализацию конкретной переменной заранее определенной (с помощью шаблона) структурой.
* Организовать обращение к элементам структуры.

Описать структуру в программе означает лишь указать ее схему или шаблон; память при этом не выделяется. Этот шаблон можно рассматривать лишь как информацию для транслятора о расположении полей и их значении по умолчанию.

Определить структуру — значит, дать указание транслятору выделить память и присвоить этой области памяти символическое имя. Описать структуру в программе можно только один раз, а определить — любое количество раз.

Практическая часть:

.model small

steck segment

dw 5 dup (' ')

steck ends

data segment

massiv db 5,3,1,8,9,4,2,6

nyli db 8 dup (0)

data ends

kod segment

assume DS: data, CS: kod

begin:

mov ax, data

mov ds, ax

xor ax, ax

CALL sortirovka

mov ah, massiv[si] ;вывели элементы массива в регистры по порядку, чтобы проверить

mov al, massiv[si+1]

mov bh, massiv[si+2]

mov bl, massiv[si+3]

mov ch, massiv[si+4]

mov cl, massiv[si+5]

mov dh, massiv[si+6]

mov dl, massiv[si+7]

mov ah, 4ch ;конец кода

int 21h

sortirovka proc near

povtor:

mov bx, 7 ;записываем 7 для проверки сортировки

mov cx, 7 ;цикл выполнится 7 раз, т.к. сравниваем 8 чисел между собой

nachalo:

mov ah, massiv[si]

cmp ah, massiv[si+1]

JBE konez ;если OPR1 <= OPR2, то идём в конец и вычитаем единицу из 7

mov bx, 10 ;присваиваем 10, чтобы bl != bh

mov al, [si+1]

xchg ah, al ;если OPR1 > OPR2, то меняем их местами и записываем в массив

mov massiv[si], ah

mov massiv[si+1], al

konez:

dec bx

inc si

LOOP nachalo

xor si, si

cmp bl, bh ;сравниваем регистры для проверки

JNE povtor ;если не равны, начинаем сначала

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

;Работа счётчика bx: изначально помещаем туда цифру 7, она уменьшается на "1" каждый раз, когда OPR1 <= OPR2 (т.е. каждый раз, когда нам не приходится менять местами элементы массива, перезаписывая их.)

;Если мы 7 раз подряд пройдём проверку (все 7 раз OPR1 <= OPR2), то регистр bx будет равен 0 это означает конец повторений.

;Если хотя бы 1 раз из 7 мы не пройдём проверку, то регистру bx будет присвоено значение 10, в таком случае мы не сможем за 7 повторений получить в регистре bx "0" и цикл начнётся сначала.

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ret

sortirovka endp

kod ends

end begin

end