ЛАБОРАТОРНАЯ РАБОТА №4

Цель работы:

приобрести умения и практические навыки для работы с директивами управления сегментами и процедурами.

Теоретическая часть:

Выделяют две группы директив управления сегментом: *стандартные* и *упрощенные*.

Стандартные директивы позволяют обеспечивать полное управление сегментами программы. Каждый сегмент описывается двумя директивами: *segment* и *ends*.

```
имя_сегмента segment выравнивание тип размерность класс
```

– *имя_сегмента* – метка, используемая для получения сегментного адреса и для объединения сегментов в группы;

– *выравнивание* – *byte* (может начинаться с любого адреса), *word* (адрес начала кратен 2), *dword* (адрес начала кратен 4), *para* (адрес начала сегмента кратен 16, используется по умолчанию), *page* (адрес начала сегмента кратен 256);

– *тип* показывает, могут ли сегменты объединяться с другими сегментами – *public*, *stack*, *common*, *private*, *at ...* . Тип *private* – сегмент не будет объединяться с другими сегментами с тем же именем вне данного модуля, *public* – все сегменты с одинаковым именем объединяются. Тип *common* – все сегменты с одним и тем же именем будут расположены по одному адресу, при этом все сегменты с данным именем будут перекрываться и совместно использовать память – размер полученного в результате сегмента будет равен размеру самого большого сегмента. Тип *at ...* – сегмент будет расположен по абсолютному адресу параграфа (*параграф* – объем памяти, кратный 16, поэтому последняя шестнадцатеричная цифра адреса параграфа равна 0). Тип *stack* – определение сегмента стека – все одноименные сегменты соединяются, адреса в этих сегментах вычисляются относительно регистра *ss*. Комбинированный тип *stack* (стек) аналогичен комбинированному типу

public, за исключением того, что регистр *ss* является стандартным сегментным регистром для сегментов стека;

– *размерность* – *use16* (по умолчанию) – адреса и команды считаются 16-разрядными, размер сегмента не превышает 64 Кбайт. *use32* – адреса и данные считаются 32-разрядными, сегмент может иметь размер до 4 Гбайт;

– *класс* – заключенная в одиночные кавычки строка, помогающая компоновщику определить соответствующий порядок следования сегментов при сборке программы из сегментов нескольких модулей. Сегменты одного класса физически размещаются в памяти друг за другом.

Любой из параметров может быть пропущен, например:

```
code segment
…
code ends
prim segment word
…
prim ends
ABC segment public ‘a’
…
ABC ends
```

Директива *ends* завершает сегментное описание – *имя_сегмента ends*.

Директива *assume сегментный регистр: <имя сегмента, nothing>* *обязательно* используется со стандартными директивами – она показывает, с какими сегментами связаны сегментные регистры. Не загружает значения в сегментные регистры. Обычно (но не обязательно) указывается при описании кода программы сразу после директивы *segment*, например:

```
; описание сегмента данных
data segment
;   директивы определения данных
data ends
; описание кодового сегмента
cseg segment
assume cs:cseg, ds:data
;   команды
cseg ends
```

При использовании *упрощенных директив* следует определить модель памяти, задаваемую при помощи следующей директивы:

```
.model модель, язык, модификатор
```

– *модель* – модели памяти могут быть следующими:

- *tiny (крошечная)* – код, данные и стек находятся в одном сегменте, размер не превышает 64 Кбайт;
- *small (малая)* – код находится в одном сегменте, данные и стек – в другом;
- *compact (компактная)* – код находится в одном сегменте, данные – в нескольких сегментах;
- *medium (средняя)* – код находится в нескольких сегментах, данные – в одном;
- *large (большая)* – и код, и данные могут находиться в нескольких сегментах;
- *huge (огромная)* – и код, и данные могут находиться в нескольких сегментах, размер может превышать 64 Кбайта;
- *FLAT* – то же самое, что *tiny*, сегмент имеет 32 разряда, линейная.

– *язык* – необязательный параметр, означающий, что программа рассчитана на вызов из другой программы на языке высокого уровня;

– *модификатор* – *NEARSTACK* (по умолчанию), *FARSTACK* – первый модификатор используется в случае, когда сегмент стека и данные образуют один сегмент, второй – во всех иных случаях. Необязательный параметр.

Директива *.code* описывает кодовый сегмент программы, *.data* – сегмент данных, *.stack* – стек. Например:

```
.code имя сегмента
<команды>
.data
<данные>
.stack размер
```

В модели *FLAT* все объединяется в одну группу *FLAT*. При использовании *tiny* – в группу *DGROUP*. В других моделях в группу *DGROUP* объединяются *.data* и *.stack*. Сегменты *ds*, *ss* (если не используется *FARSTACK*), *cs* (*tiny*) настраиваются на эту группу как при *assume*.

Сегментный регистр *cs* никогда не загружается явно – его значение загружает операционная система. Если бы кодовый сегмент не был загружен при запуске программы, система не знала бы, где найти инструкции по загрузке сегмента. Регистр *ss* также загружает операционная система. Возможность установки регистра *ds* в любые значения связана с тем, что данные могут находиться в нескольких сегментах, а также может понадобиться доступ к системным областям памяти.

Загрузка значения в сегментный регистр *ds* при использовании стандартных директив управления сегментом:

```
  ; описание сегмента данных
Data1 segment word 'data'
  …
Data1 ends
  ; описание сегмента данных
Data2 segment word 'data'
 ...
Data2 ends
  ; описание кодового сегмента
Code segment word 'code'
Assume cs: code, ds:data1, es:data2
S:   mov ax, data1       ; адрес сегмента data1 занести в ax
mov ds, ax               ;скопировать в ds адрес сегмента
mov ax, data2
mov es,ax
. . .
code ends
end S
```

Загрузка значения в сегментный регистр *ds* при использовании упрощенных директив управления сегментом (допустим, была использована модель *small* – код в одном сегменте, стек и данные – в другом):

```
.model small
.code
begin: mov ax, @data     ; адрес сегмента data занести в ax
                         ; mov ax, dgroup – другой способ
загрузки адреса сегмента данных в ax
mov ds, ax               ; скопировать в ds адрес сегмента
  . . .
.data
…
```

```
end begin
```

Процедуры – отдельная часть кодового сегмента, которая воспринимает входные данные, выполняет определенные действия и, возможно, возвращает результаты в вызывающую программу. Процедуры описываются при помощи директив *proc* и *endp*.

```
имя proc язык тип uses регистр
```

– *имя* – метка для обращения к процедуре;

– *язык* – этот параметр означает, что процедура рассчитана на вызов из программы на языке высокого уровня;

– *uses* – применяется вместе с языком, указывает, что в начале должны стоять директива *push*, а в конце – *pop*;

– *тип* – *NEAR* (вызываемая процедура и вызывающая программа находятся в одном сегменте, то есть, происходит внутрисегментный вызов – меняется только значение *ip*), *FAR* (вызываемая процедура и вызывающая программа находятся в разных сегментах, меняются значения *cs* и *ip*).

По умолчанию описание процедуры выглядит следующим образом:

```
имя proc NEAR (если используется small или compact) / FAR (если используется medium, large или huge)
        ; команды для выполнения при вызове процедуры
endp
```

Директивы не вызывают появления кода, они управляют способом вызова процедур и возвратом в вызывающую программу.

Директива *call имя / операнд* применяется для вызова процедур. Имя – метка, регистр, переменная с адресом процедуры. Заносит в стек адреса следующих команд и загружает в один или пару регистров *cs*, *ip* адрес процедуры (*NEAR* – в регистр *ip*, *FAR* – в пару регистров *cs:ip*).

Возврат выполняется с помощью команды *ret <число>*. Если был использован тип *NEAR* – используется команда *retn* – считывает с вершины стека *слово* в *ip*, если *FAR* – используется команда *retf*, считывает с вершины стека *двойное слово* в *cs:ip*. Число – необязательный параметр, показывающий, сколько байтов будет удалено из вершины стека после считывания адреса возврата, используется при передаче параметров процедуры через стек.

Существует два способа передачи параметров в процедуры: *через регистр* и *через стек*.

При *передаче параметров через регистр* перед вызовом процедуры в регистры загружаются нужные значения.

Передача параметров через стек выполняется, если количество параметров превышает размер доступного регистрового пространства. Параметры при этом помещаются в стек перед использованием директивы *call*. Для обращения к параметрам используется регистр *bp* как указатель области стека (*bp* – адресный регистр, в качестве сегментного регистра для него используется регистр *ss*). Одной из первых команд в процедуре должна быть команда, которая загружает регистр *bp* текущим значением указателя стека *sp*.

Если процедура в ходе работы использует локальные переменные, то память под эти переменные выделяется в стеке. Образ стека, который отводится под локальные переменные, называется *стековым кадром*. Вызываемая подпрограмма сначала сохраняет в стеке значение регистра *bp*. Это необходимо в тех случаях, когда процедура, в свою очередь, вызывает другую процедуру или сама вызывается из другой процедуры. Далее в стеке выделяется область под локальные переменные. Для этого значение указателя стека *sp* уменьшается на количество байтов, необходимое для размещения локальных переменных. Перед возвратом в вызывающую программу значение *sp* увеличивается на ту же величину, то есть локальные переменные удаляются из стека.

```
p proc near
  push lp
  mov bp,sp
  sub sp, N
  mov ax, [bp+4]
  mov [bp-2], ax    ; обращение к локальным переменным
  mov sp, bp        ; восстановление указателя стека
  pop bp
  ret K
p endp
```

Пример:

Написать подпрограмму, которая вычисляет сумму элементов массива. Тип элементов массива – слово. Тип процедуры – дальний. Код программы находится в одном сегменте, а стек и данные – в другом.

Будем передавать параметры через стек. Для этого в основной программе занесем в стек адрес начала массива и количество элементов массива. Заполнение стека в момент вызова процедуры будет таким:

bp=sp	*ip*
bp+2	*cs*
bp+4	количество элементов
bp+6	адрес начала массива
	…

так как тип процедуры FAR, то сохраняются значения регистров ip и cs

В процедуре скопируем в регистр *bp* значение указателя *sp*. Теперь для обращения к параметру в процедуре можно использовать адресацию по базе со сдвигом. Сдвиг – число, соответствующее расстоянию в байтах от параметра до вершины стека. Сумму элементов массива будем хранить в регистре *ax*, предварительно обнулив его. Для доступа к элементу массива будем применять адресацию по базе с индексированием. Элементы массива находятся в сегменте данных. По умолчанию обращение к этому сегменту выполняется для базового регистра *bx*. Поэтому надо скопировать адрес начала массива из стека в регистр *bx*. Для обращения к текущему элементу массива будем использовать регистр *si*.

```
.model small
.code
;описание процедуры
pp   proc far
     mov bp, sp
     mov cx, [bp+4] ; в cx – количество элементов массива
     mov bx, [bp+6] ; в bx – адрес начала массива
     xor ax, ax     ; обнуление суммы
     xor si, si     ; в si – индекс текущего элемента
m:   add ax, [bx+si];добавление элемента массива к сумме
```

```
          add  si,2      ; увеличение индекса на длину элемента
массива
          loop m
          ret 4          ; удаление параметров из стека и возврат
управления в вызывающую программу
     pp   endp
     n:   mov ax, @data
          mov ds, ax
          lea dx, mas    ; занесем в регистр dx адрес начала
массива
          push dx        ; адрес массива – в стек
          mov ax,10      ; занесем в регистр ax количество
элементов массива
          push ax        ; количество элементов массива – в стек
          call pp        ; вызов процедуры
          mov sum, ax
          mov ax,4c00h
          int 21h

     .data
     mas dw 1,2,3,4,5,6,7,8,9,10
     sum dw 0
          end n
```

Задание. Написать программу, реализующую процедуру, которая выполняет действия в соответствии с выданным вариантом. Параметры в процедуру передаются через стек. Вызвать процедуру два раза для различных массивов.

1. Директивы описания сегментов – стандартные. Выравнивание – по словам. Удвоить положительные элементы массива. Тип элементов массива – *word*. Тип процедуры – дальний.

2. Директивы описания сегментов – стандартные. Выравнивание – по двойным словам. Обнулить положительные элементы массива. Тип элементов массива – *word*. Тип процедуры – дальний.

3. Директивы описания сегментов – стандартные. Выравнивание – по страницам. Уменьшить в 2 раза положительные элементы массива. Тип элементов массива – *word*. Тип процедуры – дальний.

4. Директивы описания сегментов – стандартные. Выравнивание – по байтам. Обнулить отрицательные элементы массива. Тип элементов массива – *word*. Тип процедуры – дальний.

5. Директивы описания сегментов – стандартные. Выравнивание – по словам. Посчитать количество положительных. Обнулить положительные, установить отрицательные в единицу. Тип элементов массива – *word*. Тип процедуры – дальний.

6. Директивы описания сегментов – стандартные. Выравнивание – по двойным словам. Посчитать количество отрицательных. Обнулить отрицательные, установить положительные в единицу. Тип элементов массива – *word*. Тип процедуры – дальний.

7. Директивы описания сегментов – стандартные. Выравнивание – по страницам. Найти сумму элементов массива и вернуть результат через аккумулятор. Тип элементов массива – *word*. Тип процедуры – дальний.

8. Директивы описания сегментов – стандартные. Выравнивание – по параграфам. Найти произведение элементов массива и вернуть результат через аккумулятор. Тип элементов массива – *word*. Тип процедуры – дальний.

9. Директивы описания сегментов – стандартные. Выравнивание – по словам. Посчитать количество положительных и вернуть результат через аккумулятор. Тип элементов массива – *word*. Тип процедуры – дальний.

10. Директивы описания сегментов – стандартные. Выравнивание – по двойным словам. Посчитать количество отрицательных и вернуть результат через аккумулятор. Тип элементов массива – *word*. Тип процедуры – дальний.

11. Директивы описания сегментов – стандартные. Выравнивание – по страницам. Найти сумму положительных элементов массива и вернуть результат через аккумулятор. Тип элементов массива – *word*. Тип процедуры – дальний.

12. Директивы описания сегментов – стандартные. Выравнивание – по байтам. Найти произведение положительных элементов массива и вернуть

результат через аккумулятор. Тип элементов массива – *word*. Тип процедуры – дальний.

13. Директивы описания сегментов – стандартные. Выравнивание – по словам. Обнулить положительные, установить отрицательные в единицу. Тип элементов массива – *word*. Тип процедуры – дальний.

14. Директивы описания сегментов – стандартные. Выравнивание – по двойным словам. Обнулить отрицательные, установить положительные в единицу. Тип элементов массива – *word*. Тип процедуры – дальний.

15. Директивы описания сегментов – стандартные. Выравнивание – по страницам. Найти сумму отрицательных элементов массива и вернуть ее через аккумулятор. Тип элементов массива – *word*. Тип процедуры – дальний.

16. Директивы описания сегментов – упрощенные. Стек, данные и код – в одном сегменте. Удвоить отрицательные элементы массива. Тип элементов массива – *word*. Тип процедуры – ближний.

17. Директивы описания сегментов – упрощенные. Стек и данные – в одном сегменте, код – в другом сегменте. Заменить положительные элементы массива единицами. Тип элементов массива – *word*. Тип процедуры – ближний.

18. Директивы описания сегментов – упрощенные. Стек, данные и код – в одном сегменте. Заменить отрицательные элементы нулями. Тип элементов массива – *word*. Тип процедуры – ближний.

19. Директивы описания сегментов – упрощенные. Стек и данные – в одном сегменте, код – в другом сегменте. Найти количество положительных элементов массива. Результат вернуть через аккумулятор. Тип элементов массива – *word*. Тип процедуры – дальний.

20. Директивы описания сегментов – упрощенные. Стек, данные и код – в одном сегменте. Найти количество отрицательных элементов массива. Результат вернуть через аккумулятор. Тип элементов массива – *word*. Тип процедуры – ближний.

21. Директивы описания сегментов – упрощенные. Стек и данные – в одном сегменте, код – в другом сегменте. Найти сумму положительных элементов массива. Результат вернуть через аккумулятор. Тип элементов массива – *word*. Тип процедуры – дальний.

22. Директивы описания сегментов – упрощенные. Стек, данные и код – в одном сегменте. Найти сумму отрицательных элементов массива. Результат вернуть через аккумулятор. Тип элементов массива – *word*. Тип процедуры – ближний.

23. Директивы описания сегментов – упрощенные. Стек и данные – в одном сегменте, код – в другом сегменте. Найти максимальный элемент массива. Результат вернуть через аккумулятор. Тип элементов массива – *word*. Тип процедуры – дальний.

24. Директивы описания сегментов – упрощенные. Стек и данные – в одном сегменте, код – в другом сегменте. Найти минимальный элемент массива. Результат вернуть через аккумулятор. Тип элементов массива – *word*. Тип процедуры – дальний.

25. Директивы описания сегментов – упрощенные. Стек и данные – в одном сегменте, код – в другом сегменте. Найти количество элементов массива, которые больше заданного значения. Результат вернуть через аккумулятор. Тип элементов массива – *word*. Тип процедуры – дальний.

26. Директивы описания сегментов – упрощенные. Стек, данные и код – в одном сегменте. Найти количество элементов массива, которые меньше заданного значения. Результат вернуть через аккумулятор. Тип элементов массива – *word*. Тип процедуры – ближний.

27. Директивы описания сегментов – упрощенные. Стек и данные – в одном сегменте, код – в другом сегменте. Посчитать количество положительных элементов массива. Тип элементов массива – *word*. Тип процедуры – дальний.

28. Директивы описания сегментов – упрощенные. Стек, данные и код – в одном сегменте. Посчитать количество отрицательных элементов массива. Тип элементов массива – *word*. Тип процедуры – дальний.

29. Директивы описания сегментов – упрощенные. Стек и данные – в одном сегменте, код – в другом сегменте. Посчитать количество элементов массива, кратных 2. Тип элементов массива – *word*. Тип процедуры – ближний.

30. Директивы описания сегментов – упрощенные. Стек, данные и код – в одном сегменте. Посчитать количество элементов массива, не кратных 2. Тип элементов массива – *word*. Тип процедуры – дальний.

Порядок работы:

1. Открыть программу, указанную преподавателем, после объяснения принципов работы с ней.
2. Создать файл с расширением, указанным преподавателем.
3. Ввести текст программы.
4. Сохранить программу.
5. Выполнить компиляцию.
6. Если ошибок нет, то запустить эмуляцию программы и пошагово выполнить ее. Подготовить отчет о проделанной работе.

Вопросы к теоретическому материалу

1. Какие две группы директив управления сегментом выделяют?

2. Что позволяют стандартные директивы управления сегментом?

3. Укажите две директивы, при помощи которых описываются сегменты при использовании стандартных директив управления сегментом.

4. Укажите синтаксис директивы *segment*.

5. Что называется именем сегмента?

6. Каким может быть выравнивание сегмента?

7. Каким может быть тип сегмента?

8. Какой может быть размерность сегмента?

9. Что называется классом сегмента?

10. Укажите синтаксис директивы *ends*.

11. Какая директива обязательно используется со стандартными директивами управления сегментом? Для чего она нужна?

12. Что следует определить при использовании упрощенных директив управления сегментом?

13. Укажите синтаксис директивы определения модели памяти.

14. Какие модели памяти существуют?

15. Поясните, для чего при определении модели памяти нужен параметр язык при использовании упрощенных директив описания сегментов.

16. Каким может быть модификатор при использовании упрощенных директив описания сегментов?

17. Какие сегментные регистры загружаются операционной системой, а какие необходимо загружать самостоятельно?

18. Что такое процедура?

19. Укажите, при помощи каких двух директив описываются процедуры.

20. Укажите синтаксис директивы *proc*.

21. Для чего применяется имя процедуры?

22. Поясните, для чего при описании процедуры нужен параметр язык.

23. Укажите, какие типы может иметь процедура.

24. Охарактеризуйте тип процедуры *NEAR*.
25. Охарактеризуйте тип процедуры *FAR*.
26. Укажите способы передачи параметров в процедуры.
27. Охарактеризуйте передачу параметров в процедуру через регистр.
28. Охарактеризуйте передачу параметров в процедуру через стек.
29. Что называется стековым кадром?
30. Какие регистры используются при работе со стеком?

ПРОЦЕСС СДАЧИ ЛАБОРАТОРНОЙ РАБОТЫ

По итогам выполнения каждой лабораторной работы студент:

1. демонстрирует преподавателю правильно работающие программы;
2. демонстрирует приобретенные знания и навыки, отвечая на несколько небольших вопросов преподавателя по составленной программе, возможностям ее доработки и теме лабораторной работы в целом;
3. демонстрирует отчет по выполненной лабораторной работе.

Итоговая оценка складывается из оценок по трем указанным составляющим.

Отчет по лабораторной работе оформляется по шаблону, представленному в приложении 1. Требования к отчету представлены в приложении 2.

ПРИЛОЖЕНИЕ 1. ШАБЛОН ОТЧЕТА

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Казанский национальный исследовательский технический университет им. А.Н. Туполева – КАИ»

Институт компьютерных технологий и защиты информации

Отделение СПО ИКТЗИ (Колледж информационных технологий)

ЛАБОРАТОРНАЯ РАБОТА №__

по дисциплине

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

Работу выполнил

Студент гр.43__

Фамилия И.О.

Принял

Преподаватель Григорьева В.В.

Казань 2024

ВАРИАНТ __

1. **Цель работы**
2. **Задание на лабораторную работу** – вставляется задание на лабораторную работу, соответствующее индивидуальному варианту студента.
3. **Результат выполнения работы** – формируется описание хода выполнения работы (разработанных подпрограмм, классов, переменных, структур данных и т.п.) и вставляются скриншоты с результатами работы разработанных программ (скриншоты должны быть подписаны).
4. **Листинг программы** – вставляется код разработанной программы **с комментариями**.

ПРИЛОЖЕНИЕ 2. ТРЕБОВАНИЯ К ОФОРМЛЕНИЮ ОТЧЕТА

Лист документа должен иметь книжную ориентацию, поля документа должны составлять: левое – 3 см, правое – 1,5 см, верхнее – 2 см, нижнее 2 см.

Нумерация страниц – внизу страницы, по центру, особый колонтитул для первой страницы.

Междустрочный интервал – 1,5 (полуторный), отступ первой строки – 1,25.

Текст документа должен быть выполнен с использованием шрифта Times New Roman, размер – 14, выравнивание – по ширине. Заголовки выполняются тем же шрифтом, но размера 16, полужирное начертание, размещение – по центру.

Рисунки должны размещаться по центру, они нумеруются по порядку. Перед рисунком в тексте на него должна быть ссылка. Подпись рисунка должна располагаться по центру и быть выполнена шрифтом Times New Roman, размер – 12. Сначала происходит нумерация рисунка, а затем пишется его название.