

Московский государственный технический университет
имени Н. Э. Баумана
Факультет Информатика и системы управления
Кафедра Компьютерные системы и сети

«УТВЕРЖДАЮ»

Заведующий кафедрой ИУ-6

_____ Сюзев В.В.

Г. С. Иванова, Т.Н. Ничушкина

Методические указания по выполнению лабораторных работ № 3 - 5
по дисциплине Машинно-зависимые языки и основы компиляции

Москва 2013

Содержание

Лабораторная работа № 3.....	3
Лабораторная работа № 4.....	7
Лабораторная работа № 5.....	13

Лабораторная работа № 3

Программирование ветвлений и циклов

Цель работы: изучение средств и приемов программирования ветвлений и циклов на языке ассемблера

Объем работы: 4 часа

Теоретическая часть

1 Программирование ветвлений

Ветвления на языке ассемблера программируются с использованием команд условной и безусловной передачи управления.

Сначала выполняют сравнение. В результате будут установлены флаги. Затем, если условие не выполняется, то переходим на метку ELSE. Если условие выполняется, то переход не осуществляется, и управление переходит к следующей команде, т.е. выполнению команд, помеченных как Операции1. По завершению Операций1 передаем управление на команду, следующую за ветвлением, иначе будут выполняться команды, помеченные как Операции2, переход на которые был обозначен меткой ELSE. Если переход был осуществлен, то после Операций 2 переходим на команду, следующую за ветвлением:

```
cmp ...
j<условие> else
Операции1
jmp com
else: Операции2
com: ...
```

Пример. Написать фрагмент вычисления $X=\max(A,B)$:

```
mov ax, A
cmp ax, B ; сравнение A и B
jl less ; переход по меньше
mov X, ax
jmp continue ; переход на конец ветвления
less: mov ax, B
      mov X, ax
continue: ...
```

2 Программирование итерационных циклов (цикл-пока)

Программирование циклических процессов осуществляется с использованием либо команд переходов, либо – в случае счетных циклов – с использованием команд организации циклов.

Так, чтобы реализовать цикл-пока необходим один условный и один безусловный переходы:

```
сycl: cmp ... ; проверка условия выхода  
      jne com ; выход из цикла  
      Операции ; тело цикла  
      jmp cycl ; возврат в цикл  
com: ...
```

Пример. Написать фрагмент суммирования чисел от 1 до 10, используя итерационный цикл.

```
mov ax,0 ; обнуление суммы  
mov bx,1 ; первое слагаемое  
cycl: cmp bx, 10 ; слагаемое больше 10  
      jb continue ; выход из цикла  
      add ax, bx ; суммирование  
      inc bx ; следующее число  
      jmp cycl ; возврат в цикл  
continue: ... ; выход, сумма - в ax
```

3 Организация счетного цикла

Для организации счетного цикла с использованием команды LOOP необходимо записать количество повторений в регистр счетчика ECX. Тогда команда LOOP будет отсчитывать повторения, вычитая 1 из счетчика.

Примечание. Если перед началом цикла в регистр ECX загружен 0, то цикл выполняется 2^{32} раз. Такая ситуация называется «зацикливанием», поскольку программа надолго «зависает».

```
mov ECX,n ; загрузка счетчика  
begin_loop: Операции ; тело цикла  
loop begin_loop
```

Пример. Написать фрагмент суммирования чисел от 1 до 10, используя счетный цикл.

```
mov AX,0 ; обнуление суммы
```

```

mov BX,1 ; первое слагаемое
mov ECX,10 ; загрузка счетчика
cycl: add AX,BX ; суммирование
inc BX ; следующее число
loop cycl ; возврат в цикл
continue: ... ; выход, сумма – в ax

```

Порядок выполнения работы

1. Прочитать и проанализировать задание в соответствии со своим вариантом.
2. Разработать схему алгоритма решения задачи.
3. Написать программу на языке ассемблера.
4. Вызвать среду программирования RadAsm, создать новый проект консольного приложения и ввести текст программы в среду программирования.
5. Подобрать тестовые данные (не менее 3-х вариантов) и заранее просчитать результат.
6. Отладить программу на выбранных тестовых данных.
7. Продемонстрировать работу программы преподавателю.
8. Составить отчет по лабораторной работе.
9. Защитить лабораторную работу преподавателю.

Требования к отчету

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

Каждый отчет должен иметь титульный лист, на котором указывается:

- а) наименование факультета и кафедры;
- б) название дисциплины;
- в) номер и тема лабораторной работы;
- г) фамилия преподавателя, ведущего занятия;
- д) фамилия, имя и номер группы студента;
- е) номер варианта задания.

Кроме того отчет по лабораторной работе должен содержать:

- 1) схему алгоритма, выполненную вручную или в соответствующем пакете;
- 2) текст программы;
- 3) результаты тестирования, которые должны быть оформлены в виде таблицы вида:

Исходные данные	Ожидаемый результат	Полученный результат

- 4) выводы.

Контрольные вопросы

1. Какие машинные команды участвуют при программировании ветвления и циклов?
2. Выделите в своей программе фрагмент, реализующий ветвление. Каково назначение каждой машинной команды фрагмента?
3. Чем вызвана необходимость использования команд безусловной передачи управления?
4. Поясните последовательность команд, выполняющих операции ввода-вывода в вашей программе. Чем вызвана сложность преобразований данных при выполнении операций ввода-вывода?

Лабораторная работа № 4

Программирование обработки массивов и матриц

Цель работы: изучение приемов моделирования обработки массивов и матриц в языке ассемблера

Объем работы: 4 часа

Теоретическая часть

1 Моделирование одномерных массивов

Массив во внутреннем представлении – это последовательность элементов в памяти. В ассемблере такую последовательность можно определить, например, так:

а **WORD 10, 13, 28, 67, 0, -1** ; массив из 6 чисел длиной слово.

Программирование обработки выполняется с использованием адресного регистра, в котором хранится либо смещение текущего элемента относительно начала сегмента данных, либо его смещение относительно начала массива. При переходе к следующему элементу и то и то смещение увеличиваются на длину элемента. Если длина элемента отлична от единицы, то можно использовать масштаб.

Для работы с массивами часто используется специальная команда – команда загрузки исполнительного адреса:

LEA r32, mem

Команда вычисляет исполнительный адрес второго операнда, находящегося в памяти, и помещает его в регистр первого операнда.

Примеры:

lea EBX, Exword ; в регистр **EBX** загружается исполнительный адрес **Exword**

lea EBX, [EDI+10] ; в регистр **EBX** загружается адрес 10-го байта относительно точки, на которую указывает адрес в регистре **EDI**.

Пример. Написать процедуру, выполняющую суммирование массива из 10 чисел размером слово.

Вариант 1 (используется адрес):

```
mov AX, 0
```

```
lea EBX, MAS
```

```
mov ECX, 10
```

```
cycl: add AX, [EBX]
```

```
add EBX, 2
```

```
loop cycl
```

Вариант 2 (используется смещение):

```
mov AX, 0
```

```
mov EBX, 0
```

```
mov ECX, 10
```

```
cycl: add AX, MAS[EBX*2]
```

```
add EBX, 1
```

```
loop cycl
```

Второй вариант позволяет получать более наглядный код и потому является предпочтительным.

В том случае, если элементы просматриваются не подряд, адрес элемента может рассчитываться по его номеру (числа нумерованы с единицы):

$A_{\text{исп}} = A_{\text{начала}} + (<\text{Номер}> - 1) * <\text{длина элемента}>$.

Полученный по формуле адрес записывается в 32-х разрядный регистр и используется для доступа к элементу.

Пример. Написать фрагмент, который извлекает из массива, включающего 10 чисел размером слово, число с номером n ($n \leq 10$).

mov EBX, N ; номер числа

dec EBX ; вычитаем 1

mov AX, MAS [EBX*2] ; результат в AX

2 Моделирование матриц

Значения матрицы могут располагаться в памяти по строкам и по столбцам. Для определенности будем считать, что матрица расположена в памяти построчно, как в Паскале и C++.

При обработке элементов матрицы следует различать просмотр по строкам, просмотр по столбцам, просмотр по диагоналям и произвольный доступ.

Если матрица расположена в памяти по строкам и просмотр выполняется по строкам, то обработка может выполняться так, как в одномерном массиве, без учета перехода от одной строки к другой.

Пример. Написать фрагмент определения максимального элемента матрицы A(3,5).

```
mov EBX, 0 ; номер элемента 0
mov ECX, 14 ; счетчик цикла
mov AX, A ; заносим первое число
cyc1: cmp AX, A[EBX*2+2] ; сравниваем числа
      jge next ; если больше, то перейти к следующему
      mov AX, A[EBX*2+2] ; если меньше, то запомнить
next: add EBX, 1 ; переходим к следующему числу
      loop cyc1
```

Просмотр по строкам при необходимости фиксировать завершение строки и просмотр по столбцам при построчном расположении в памяти выполняются в двойном цикле.

Пример. Определить сумму максимальных элементов столбцов матрицы A(3,5).

```
mov AX, 0 ; обнуляем сумму
mov EBX, 0 ; смещение элемента столбца в строке
mov ECX, 5 ; количество столбцов
cyc1: push ECX ; сохраняем счетчик
      mov ECX, 2 ; счетчик элементов в столбце
      mov DX, A[EBX] ; заносим первый элемент столбца
      mov ESI, 10 ; смещение второго элемента столбца
cyc2: cmp DX, A[EBX]+[ESI] ; сравниваем
      jge next ; если больше или равно - к следующему
      mov DX, A[EBX]+[ESI] ; если меньше, то сохранили
next: add ESI, 10 ; переходим к следующему элементу
      loop cyc2 ; цикл по элементам столбца
```

add AX,DX ; просуммировали максимальный элемент

pop ECX ; восстановили счетчик

add EBX,2 ; перешли к следующему столбцу

loop cyc11 ; цикл по столбцам

При просмотре по диагонали обычно используют один цикл, через переменную которого рассчитываются смещения элементов массива. Однако проще использовать специальный регистр смещения, который должен соответствующим образом переадресовываться.

Порядок выполнения работы

1. Прочитать и проанализировать задание в соответствии со своим вариантом.
2. Разработать схему алгоритма решения задачи.
3. Написать программу.
4. Вызвать среду программирования, создать новый проект консольного приложения и ввести текст программы в среду программирования.
5. Подобрать тестовые данные (не менее 3-х вариантов).
6. Отладить программу на выбранных тестовых данных.
7. Продемонстрировать работу программы преподавателю.
8. Составить отчет по лабораторной работе.
9. Защитить лабораторную работу преподавателю.

Требования к отчету

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

Каждый отчет должен иметь титульный лист, на котором указывается:

- а) наименование факультета и кафедры;
- б) название дисциплины;
- в) номер и тема лабораторной работы;
- г) фамилия преподавателя, ведущего занятия;
- д) фамилия, имя и номер группы студента;
- е) номер варианта задания.

Кроме того отчет по лабораторной работе должен содержать:

- 1) схему алгоритма, выполненную вручную или в соответствующем пакете;
- 2) текст программы;
- 3) результаты тестирования, которые должны быть оформлены в виде таблицы вида:

Исходные данные	Ожидаемый результат	Полученный результат

- 4) выводы.

Контрольные вопросы

1. Почему в ассемблере не определены понятия «массив», «матрица»?
2. Как в ассемблере моделируются массивы?
3. Поясните фрагмент последовательной адресации элементов массива? Почему при этом для хранения частей адреса используют регистры?
4. Как в памяти компьютера размещаются элементы матриц?
5. Чем моделирование матриц отличается от моделирования массивов? В каких случаях при выполнении операций для адресации матриц используется один регистр, а в каких – два?

Лабораторная работа № 5

Программирование с использованием разноязыковых модулей

Цель работы: изучение конвенций о способах передачи управления и данных при вызове из программы, написанной на языке высокого уровня, подпрограмм, написанных на ассемблере

Объем работы: 5 часов

Теоретическая часть

Основные проблемы связи разноязыковых модулей:

- осуществление совместной компоновки модулей;
- организация передачи и возврата управления;
- передача параметров:
 - с использованием глобальных переменных,
 - с использованием стека (по значению и по ссылке),
- обеспечение возврата результата функции;
- обеспечение корректного использования регистров процессора.

Корректное обращение к процедурам, написанным на ассемблере, из приложений Windows, и наоборот, предполагает соблюдение определенных правил. Эти правила определяют способ передачи параметров, закономерности формирования внутренних имен подпрограмм и глобальных данных и применяемую модель памяти.

Правила, декларирующие способы передачи параметров при организации связи модулей, получили название «конвенции». Поскольку первоначально основные правила передачи управления и параметров определялись языком программирования, названия основных конвенций связано с именами двух основных универсальных языков программирования: Паскаль и Си. Остальные получили свои имена в соответствии с основными свойствами: стандартная Windows, защищенная и регистровая.

Конвенция **Паскаль** предполагает, что параметры помещаются в стек в том порядке, в котором они встречаются в списке формальных параметров подпрограммы. Причем все параметры передаются через стек, регистры для передачи параметров не используются. Завершаясь, подпрограмма удаляет параметры из стека, а потом возвращает управление.

Конвенция **Си** предполагает обратный порядок помещения параметров в стек, регистры также не используются, и параметры из стека удаляет вызывающая программа.

Стандартная и **Защищенная** конвенции используют обратный порядок занесения параметров в стек, но очистку стека вызываемой процедурой. Эти конвенции очень похожи. Отличие только в том, что Защищенная конвенция формирует исключение при обнаружении ошибок, связанных с передачей параметров.

Регистровая конвенция означает передачу до трех параметров в регистрах. Обычно этого хватает, но если параметров больше, то остальные передаются через стек.

Конвенции реализованы в основных средах программирования (таблица 1).

Таблица 1 – Конвенции по передаче параметров

	Конвенция	Delphi	C++ Builder и Visual C++	Порядок параметров в стеке	Очистка стека	Использование регистров
1	Паскаль	pascal	__pascal	Слева направо	Вызываемая процедура	Нет
2	Си	cdecl	__cdecl	Справа налево	Вызывающая программа	Нет
3	Стандартная	stdcall	__stdcall	Справа налево	Вызываемая процедура	Нет
4	Защищенная	savecall	–	Справа налево	Вызываемая процедура	Нет
5	Регистровая	register	__fastcall	Справа налево	Вызываемая процедура	Три регистра EAX, EDX, ECX (VC – до 2-х), остальные параметры – в стеке

Компилятор Delphi Pascal изменяет внутренние имена подпрограмм и глобальных переменных, заменяя строчные буквы на прописные. Это позволяет не учитывать регистр при записи программы на этом языке.

Компиляторы C изменяют имена всех глобальных («extern») переменных программы, добавляя перед ними символ подчеркивания «_». При этом строчные и прописные буквы в C различаются. Компиляторы C++ дописывают к именам функций специальные комбинации символов, отражающие используемый способ передачи параметров и их тип. В таблице 2 приведены основные особенности согласования имен в перечисленных средах.

Таблица 2 – Особенности формирования внутренних имен

	Delphi Pascal	Borland C++	Visual C++
Чувствительность к регистру клавиатуры	Не различает строчных и прописных букв	Различает строчные и прописные буквы	Различает строчные и прописные буквы
Преобразование внешних имен	Преобразует все строчные буквы имен в прописные	Помещает символ «_» перед внешними именами	Помещает символ «_» перед внешними именами
Преобразование имен подпрограмм	Преобразует все строчные буквы имен в прописные	Изменяет внутреннее имя подпрограммы: @<Имя>\$q<описание параметров>	Изменяет внутреннее имя подпрограммы: @_<имя>@<число параметров* 4>

Во всех рассматриваемых средах необходимо сохранять регистры: EBX, EBP, ESI, EDI, регистры EAX, EDX, ECX нигде сохранять не надо.

Согласование типа вызова не выполняется, поскольку во всех случаях используется модель памяти Flat, для которой все вызовы ближние near, но смещение имеет размер 32 бита.

Программы на Delphi Pascal используют модель FLAT, а потому модули ассемблера должны быть разработаны применительно к той же модели. Следовательно, все адреса в ассемблере должны быть ближними и состоять только из смещения в сегменте (4 байта).

Параметры передаются в вызываемую подпрограмму через стек, и там же размещаются локальные переменные. Вызов подпрограммы реализуется по варианту:

```

push  <Параметр 1> ; занесение параметров в стек
...
push  <Параметр n>
call  <Имя подпрограммы> ; вызов подпрограммы

```

Вызываемые подпрограммы должны иметь стандартно оформленные вход – *пролог* и выход – *эпилог*.

Пролог:

```

<Имя> proc near
  push  EBP      ; сохранить старое EBP в стеке
  mov   EBP, ESP ; установить базу для параметров в стеке
  sub   ESP, <Объем памяти локальных переменных>
  <Сохранение используемых регистров>
  ...

```

Эпилог:

```

...
<Восстановление используемых регистров>
mov   ESP, EBP ; удалить область локальных переменных
pop   EBP      ; восстановить значение EBP
ret   <Размер области параметров>

```

В момент получения управления подпрограммой в стеке находятся параметры в виде значений или адресов и 4-х байтовый адрес возврата в вызывающую программу (рисунок 1, *а*). Затем вызываемая подпрограмма размещает в стеке старое значение EBP, область локальных переменных и использует стек для своих надобностей (рисунок 1, *б*).

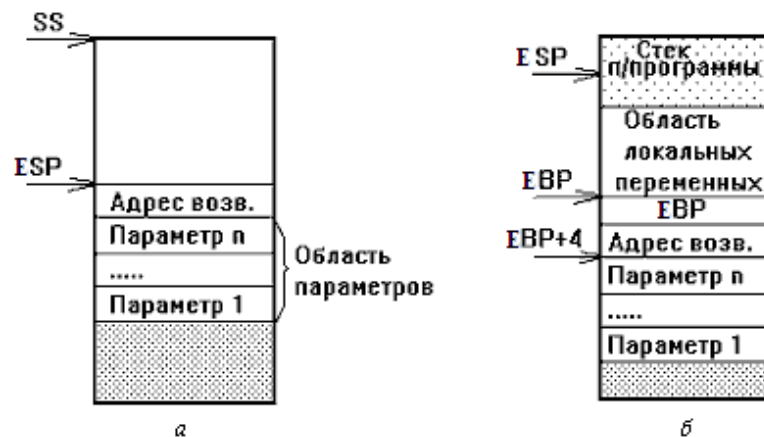


Рисунок 1 – Содержимое стека:

а – в момент передачи управления подпрограмме; *б* – во время работы подпрограммы

Адрес области параметров в этом случае определяется относительно содержимого регистра EBP. Так [EBP+8] – адрес последнего параметра. Адреса остальных параметров определяются аналогично с учетом длины каждого параметра в стеке (см. далее).

При выходе из подпрограммы команда **ret** должна удалить из стека всю область параметров, в противном случае произойдет нарушение работы вызывающей программы.

Delphi Pascal использует следующие внутренние представления данных.

Целое –

shortint: -128..127 – байт со знаком;
byte: 0..255 – байт без знака;
smallint: -32768..32767 – слово со знаком;
word: 0..65535 – слово без знака;
integer, longint: – двойное слово со знаком.

Символ – **char** – код ANSI байт без знака.

Булевский тип – **boolean** – 0(false) и 1(true) – байт без знака.

Указатель – **pointer** – 32-х разрядное смещение.

Строка – **shortstring** – символьный вектор указанной при определении длины, содержащий текущую длину в первом байте.

Массив – **array** – последовательность элементов указанного типа, расположенных в памяти таким образом, что правый индекс возрастает быстрее левого (для матрицы - построчно).

Для обращения к данным этих типов в программе на ассемблере необходимо использовать вполне определенные типы переменных. Соответствие типов представлено в таблице 3.

Таблица 0 – Соответствие типов ассемблера и Delphi Pascal

№	Тип данных ассемблера	Размер памяти	Соответствующий тип данных Delphi Pascal
1	BYTE	1 байт	Shortint, byte, char, boolean
2	WORD	2 байта	SmallInt, word
3	FWORD	6 байт	Real
4	DWORD	4 байта	Single, указатель, integer
5	QWORD	8 байт	Double, comp
6	TBYTE	10 байт	Extended

Сложные структурные типы описывают, указав тип только первого элемента и используя затем его для обработки всей структуры.

В Delphi Pascal параметры могут передаваться двумя способами: по значению и по ссылке (с указанием **var** или **const**). В первом случае подпрограмме передаются копии значений параметров, и, соответственно, она не имеет возможности менять значения передаваемых параметров в вызывающей программе. Во втором случае подпрограмма получает адреса передаваемых значений и может не только читать значения, но и менять их. И в том и в другом случае параметры или их адреса заносятся в стек. Причем, параметры всегда помещаются в стек в том порядке, в котором они описаны при объявлении процедуры, то есть слева направо. Исключение составляет только конвенция register, при которой до трех параметров помещается в регистры, а остальные помещаются в стек в порядке, обратном их описанию в списке параметров.

Параметры – значения скалярного типа (char, Boolean, smallint, word, shortint, byte, integer и перечисляемые типы) непосредственно помещаются в стек. Если размер параметра составляет 1 байт, то он помещается в стек в виде целого слова. Сам параметр

располагается в первом (младшем) байте этого слова, старший байт при этом не инициализируется. Параметры размером 2 и 4 байта помещаются в стек в виде слова и двойного слова соответственно.

Адреса всех типов помещаются в стек в виде 32-х разрядных смещений.

Строковые параметры, переданные по значению, независимо от их размера вызывающей программой в стек не записываются. Вместо этого в стек помещается адрес копии строки (4 байта). Сама копия размещается в отведенной для этого памяти стека, предназначенной для локальных данных.

Порядок выполнения работы

1. Прочитать и проанализировать задание в соответствии со своим вариантом.
2. Выполнить декомпозицию программы в соответствии с заданием.
3. Разработать схемы алгоритмов всех частей.
4. Написать программу на языке высокого уровня.
5. Вызвать среду программирования и ввести текст программы в среду программирования.
6. Написать программу на языке ассемблера и выполнить ее трансляцию.
7. Компоновать части в единую программу средствами среды языка программирования высокого уровня.
8. Отладить программу на выбранных тестовых данных.
9. Продемонстрировать работу программы преподавателю.
10. Составить отчет по лабораторной работе.
11. Защитить лабораторную работу преподавателю.

Требования к отчету

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

Каждый отчет должен иметь титульный лист, на котором указывается:

- а) наименование факультета и кафедры;
- б) название дисциплины;
- в) номер и тема лабораторной работы;
- г) фамилия преподавателя, ведущего занятия;
- д) фамилия, имя и номер группы студента;
- е) номер варианта задания.

Кроме того отчет по лабораторной работе должен содержать:

- 1) схемы алгоритма всех частей программы, выполненным вручную или в соответствующем пакете;
- 2) схематическое изображение содержимого стека в момент передачи управления;
- 3) текст подпрограммы на ассемблере и фрагменты текста программы на языке высокого уровня, которые отвечают за передачу управления подпрограмме;
- 4) результаты тестирования, которые должны быть оформлены в виде таблицы вида:

Исходные данные	Ожидаемый результат	Полученный результат

5) выводы.

Контрольные вопросы

1. Что такое «конвенции о связи»? Перечислите конвенции, которые вы знаете.
2. Какие конвенции вы использовали при создании своей программы?
3. Как связана структура данных стека в момент передачи управления и текст программы и подпрограмм?
4. С какой целью применяют разноязыковые модули в одном проекте?