ЛАБОРАТОРНАЯ РАБОТА 7

ИЗУЧЕНИЕ ОДНОМЕРНЫХ МАССИВОВ В МП X86

Рассмотрим решение следующей задачи.

1. **Задание.** Выполнить сортировку массива простым выбором, используя команды управления циклами и режим адресации с масштабированием. Длина массива и сам массив вводятся из файла `in.txt`. Признак конца ввода – длина массива, равная нулю. Файл с исходными данными содержит корректные данные, которые проверять не нужно. Результаты работы программы выводятся на терминал.

Написать программу на С и на ассемблере, которая читает из файла массив, выполняет его сортировку и выводит результаты на дисплей.

2. Определим исходные данные, необходимые для решения поставленной задачи. По условию задания имеется исходный файл с именем in.txt, в котором хранится длина массива и сам массив целых чисел. Поэтому необходимо объявить переменную `len`, в которой будет храниться длина массива. Кроме того, нужно объявить массив, с которым будет работать программа. Поэтому объявим массив длинных целых чисел, имеющий имя `arr`. Длина массива будет равна максимально возможной длине. Поэтому объявим константу MAX_LENGTH и присвоим ей значением 100, которое обозначает максимальную длину массива. Массивы меньшей длины смогут разместиться в этом массиве, а массивы большей дины программа обрабатывать не будет.

Поскольку фрагменты на С и на ассемблере будут обрабатывать данные и записывать результаты в тот же самый массив, то нам потребуется два массива с исходными данными: один для фрагмента на С с именем `arr` и второй для ассемблерной вставки с именем `arr_a`.

Исходные данные.

`in.txt` – текстовый файл с исходными данными;

`MAX_LENGTH` – максимальная длина исходного массива – константа;

`arr, arr_a` – исходный массив длинных целых чисел.

3. Определим имена переменных, в которые будут записаны полученные результаты. По условию задачи отсортированные числа записываются в тот же массив, поэтому мы будем использовать исходные массивы для С-фрагмента и для ассемблерной вставки и определять новые переменные для результата не будем.

Требуемый результат.

`arr, arr_a` – результирующий массив длинных целых чисел.

4. Разработка алгоритма.

Данная программа должна выполнять следующие функции: открывать файл с именем

in.txt, читать из файла длину массива и сам массив, выполнять сортировку исходного массива, записывая данные в исходный файл. Алгоритм решения задачи включает такие шаги:

```
Открыть файл с именем in.txt;
Повторять в цикле
  Прочитать длину массива;
  Если длина массива недопустима, завершить цикл;
  Копировать содержимое файла в исходный массив;
  Сделать копию массива для ассемблера;
  Вывод исходного массива;
  Сортировать массив на C;
  Сортировать массив на ассемблере;
  Вывод исходного массива после сортировки на C;
  Вывод исходного массива после сортировки на ассемблере;
Закрыть файл с именем in.txt;
```

В литературе находим алгоритма сортировки простым выбором либо разрабатываем алгоритм самостоятельно. Окончательная версия алгоритма решения задачи приводится ниже.

Описание алгоритма на псевдокоде.

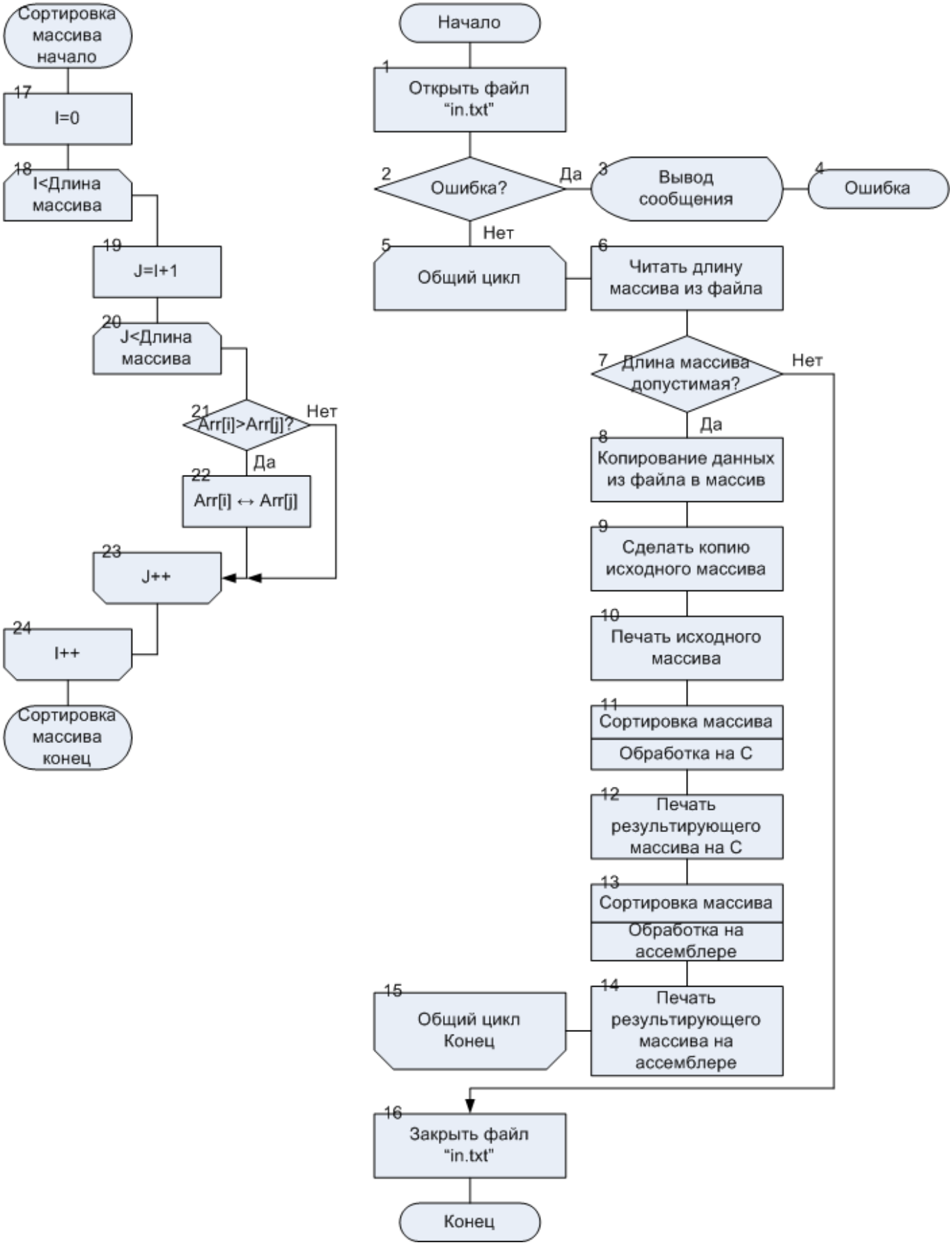

Рис.1. Схема алгоритма сортировки простым выбором и решения задачи в целом.

Детальный алгоритм решения задачи представлен ниже (рис.1). В одном блоке указываются те операции, которые можно выполнять последовательно слева направо. Для выполнения проме-

жуточных вычислений используется отдельная вершина в схеме алгоритма. Чтобы ссылаться на отдельные блоки в схеме алгоритма удобно их пронумеровать, как показано на рисунке.

Описание схемы алгоритма.

Блок 1 – "Открыть фай in.txt". Данный блок открывает файл с именем in.txt для того, чтобы из него можно было читать данные и записать в него данные.

Блок 2 – "Ошибка?". Если при открытии файла возникла ошибка, например файл с таким именем на диске отсутствует, то имеет место ошибка открытия файла. Если ошибки нет, то переходим на блок 5 и продолжаем нормальное выполнение программы. Если же ошибка имеет место, следует завершить выполнение программы и выдать сообщение об ошибке, для чего переходим на блоки 3 и 4.

Блок 3 – "Вывод сообщения об ошибке". Данный блок выдает на дисплей тестовое сообщение об ошибке при открытии файла и переходит на блок 4.

Блок 4 – "Ошибка". Данный блок завершает выполнение программы с кодом ошибки 1.

Блок 5 - "Общий цикл". Данный блок начинает бесконечный цикл выполнения программы. В этом бесконечном цикле выполняются блоки 6-14.

Блок 15 – "Общий цикл. Конец". В этом блоке завершается бесконечный цикл.

Блок 6 – "Читать длину массива из файла". В этом блоке мы читаем очередное число из файла, которое является длиной массива, и записываем его в переменную, хранящую длину массива `len`.

Блок 7 – "Длина массива допустимая?". Если длина массива допустимая, то продолжаем выполнение основного цикла и переходим на блок 8. В противном случае – выходим из цикла и переходим на блок 17, расположенный сразу после общего цикла. Длина массива считается допустимой, если она не равна нулю (это условие окончания вычислений) и меньше предельно возможной (в нашем случае это значение определяется константой `MAX_LENGTH`, равной 100). Т.е. если значение длины массива равно нулю или больше 100, основной цикл завершается переходом на блок 17.

Блок 8 – "Копирование данных из файла в массив". В данном блоке выполняется чтение из файла `len` чисел, и запись их в массив `arr` для последующей обработки в языке С. Данная операция выполняется в цикле.

Блок 9 – "Сделать копию исходного массива". В данном блоке выполняется копирование исходного массива `arr`, который будет обрабатываться в С, в массив `arr_a`, который будет обрабатываться в ассемблере. Данная операция выполняется в цикле. Данный блок нужен потому, что после сортировки С невозможно будет проверить корректность алгоритма на ассемблере, поэтому необходима дополнительная копия исходного необработанного массива.

Блок 10 – "Печать исходного массива". Данный блок в цикле выводит на дисплей значения всех элементом исходного массива. Любопытно все-таки знать, что же мы сортируем.

Блок 11 – "Сортировка массива". Данный блок выполняет сортировку исходного массива на С простым выбором. Детальный алгоритм сортировки простым выбором реализован в блоках 17-24.

Блок 12 – "Печать результирующего массива на С". В этом блоке выполняется печать обработанного массива после его сортировки на С.

Блок 13 – "Сортировка массива". Данный блок выполняет сортировку исходного массива на ассемблере. Детальный алгоритм сортировки простым выбором реализован в блоках 17-24.

Блок 14 – "Печать результирующего массива на ассемблере". В этом блоке выполняется печать обработанного массива после его сортировки на ассемблере.

Блок 16 – "Закрыть файл". Данный блок закрывает исходный файл, с которым мы работали. В принципе этого можно не делать, так как операционная система автоматически закроет файл после завершения программы. Однако методически грамотно закрывать все файлы, которые были открыты в программе.

Блоки 17-24 представляют алгоритм сортировки простым выбором. Данный алгоритм реализован в виде дух вложенных циклов – внешнего и внутреннего. Внешний цикл реализован блоками 17, 18, 24, а внутренний – блоками 19, 20, 23. Операции, которые выполняются во внутреннем цикле, реализованы блоками 21-22. В блоках 19-23 находится минимальный элемент массива `arr[j]`, который заменяет текущий элемент массива `arr[j]`. Данный алгоритм сортировки реализуется как на С, так и на ассемблере.

Блок 17 – "Инициализировать переменную внешнего цикла I (I=0)". Данный блок устанавливает переменную цикла I в ноль.

Блок 18 – "I<Длина массива". Данный блок проверяет условие завершения внешнего цикла – цикл завершается после обработки всех элементов массива.

Блок 24 – "Увеличить переменную внешнего цикла I (I++)". Данный блок выполняет инкремент переменной внешнего цикла I, и управление передается на начало внешнего цикла, т.е. на блок 18.

Блок 19 – "Инициализировать переменную внутреннего цикла J (J=I+1)". Данный блок настраивает переменную внутреннего цикла J на следующий элемент массива. Поиск минимального элемента массива ведется в правой его части после текущего элемента `arr[i]`.

Блок 20 – "J<Длина массива". Данный блок проверяет условие завершения внутреннего цикла – цикл завершается после обработки всех элементов массива.

Блок 21 – "Увеличить переменную внутреннего цикла J (J++)". Данный блок выполняет инкремент переменной внутреннего цикла J, и управление передается на начало внутреннего цикла,

т.е. на блок 20.

Блок 21 – "Сравнить элементы массива (`arr[i]>arr[j]`?)". Данный блок проверяет, не является ли текущий элемент массива из левой его части больше очередного элемента массива из его правой части. Если текущий элемент массива *меньше или равен* элементу из его правой части, то делать ничего не надо и переходим к обработке следующего элемента массива, т.е. на конец внутреннего цикла (на блок 23). Если же текущий элемент массива *больше* элемента из его правой части, то необходимо обменять их местами в блоке 22.

Блок 22 – "Обменять местами текущий элемент массива и очередной элемент из правой его части". Данный блок выполняет обмен элементов массива: меняются местами текущий элемент из левой обработанной части массива и очередной элемент из правой необработанной его части.

5. Кодирование алгоритма.

Выполним кодирование алгоритма решения задачи в виде программы на языке на С и на ассемблере. Программа представляет собой бесконечный цикл, который начинается в строках 26-27, а заканчивается – в строке 87. В этом цикле из файла читается последовательность данных и записывается в массив, затем этот массив сортируется при помощи метода сортировки простым выбором на языке С и на ассемблер, а затем отсортированный массив выводится на печать. Исходные данные являются длинными знаковыми числами, которые будут вводится и выводиться в десятичном виде.

В строках 10-13, а также в строке 18 объявлены исходные переменные и константы. В строке 11 как константа MAX_LENGTH объявлена максимально возможная длина массива равная 100. Исходные массивы `arr` и `arr_a` максимально возможной длины для С и для ассемблера объявлены в строке 12, а в строке 13 объявлена переменная `len`, в которой будет храниться длина массива, введенная из файла. Эти переменные являются глобальными и доступны через их имена и в С и в ассемблере. В строке 18 объявлена переменная `fin`, необходимую для работы с файлами и являющуюся указателем на тип FILE.

Главное при решении данной задачи верное обращение к элементам массива в С и в ассемблере. Для обращения к элементу массива `arr` с индексом `i`, используют такую форму записи `arr[i]`. Для доступа к элементам массива в ассемблере следует использовать режим адресации с масштабированием. Так для обращения к элементу массива `arr` с индексом `i`, который хранится в регистре `esi` служит такая форма записи

```
arr(,%esi,4) – синтаксис AT&T
arr[%esi*4] -  синтаксис Intel.
```

Таким образом, чтобы прочитать элемент массива `arr[i]` в аккумулятор нужны такие команды:

```
movl   arr(,%esi,4),%eax          // синтаксис AT&T
```

```
mov    eax, arr[%esi*4]           // синтаксис Intel.
```

Строка 20 выводит заставку.

Блок 1 – "Открыть фай in.txt". Данный блок реализуют функция `fopen()`, расположенная в строке 22. Функция `fopen()` открывает файл с именем `in.txt`. Файл открывается в текстовом виде (по умолчанию) и в режиме только для чтения (параметр "`r`"). Функция `fopen()` возвращает переменную `fin`, являющуюся указателем на тип FILE. В дальнейшем переменная `fin` используется для всех операций с открытым файлом `in.txt`. Если при открытии файла возникла ошибка, то переменная `fin` получает значение NULL. Такое возможно, например, если файл с указанным именем отсутствует в данном каталоге.

Блоки 2,3,4 – "Обработка ошибок". Данный блок реализуют операторы, расположенные в строках 22-25. В строке 22 проверяется значение файловой переменной `fin`. Если она имеет значение NULL, т.е. имеет место ошибка открытия файла, то в строке 24 с помощью функции `printf()` выдается сообщение об ошибке, и выполнение программы завершается при помощи функции `exit()`, которая устанавливает код ошибки, равный 1. Эта функция автоматически закрывает открытый файл `in.txt`.

Блоки 5,15 - "Общий цикл". Начало общего цикла реализует оператор for в строках 26-27, а его конец закрывающая скобка в строке 86.

Блок 6 – "Читать длину массива из файла". Это блок реализуется в строке 29 функцией `fscanf()`, которая при помощи файловой переменной читает из файла очередное число, являющееся длиной массива, и помещает это значение в переменную `len`.

Блок 7 – "Длина массива допустимая?". Этот блок в строке 31 реализует условный оператор. Этот оператор проверяет значение длины массива в переменной `len`, и, если оно не является допустимым, т.е. равно нулю или больше 100, выполнение основного цикла прерывается оператором break, и управление передается на строку 87.

Блок 8 – "Копирование данных из файла в массив". Данный блок реализует оператор цикла в строке 33, который при помощи функции чтения из файла `fscanf()` читает из файла десятичное число, используя файловую переменную `fin`, и записывает полученное значение в текущий элемент массива (`arr[i]`).

Блок 9 – "Сделать копию исходного массива". Данный блок реализуется в строке 35 при помощи оператора цикла, который тривиально переписывает `len` чисел из массива `arr` в массив `arr_a`.

Блок 10 – "Печать исходного массива". Функция `printf()`, расположенная в строке 39 выводит на печать длину массива `len`, а в строке 40 в цикле при помощи функции `printf()` выводятся на печать элементы исходного массива. Для вывода каждого числа отводится 8 позиций, что позволяет выводить в строке ровно 10 чисел.

Блок 11 – "Сортировка массива". Этот блок реализуют операторы, расположенные в строках 42-51. Детальная реализация блоков 17-24 алгоритма сортировки будет рассмотрена позднее.

Блок 12 – "Печать результирующего массива на С". Реализация печати результата сортировки массива на С приведена в строках 53-54. Функция `printf()`, расположенная в строке 53, выводит на печать сообщение, а такая же функция в строке 40 в цикле выводит на печать элементы исходного массива. Для вывода каждого числа отводится 8 позиций, что позволяет выводить в строке ровно 10 чисел.

Блок 13 – "Сортировка массива". Этот блок реализуют команды, расположенные в строках 57-81. Данный блок выполняет сортировку исходного массива на ассемблере. Детальная реализация алгоритма сортировки простым будет рассмотрена позднее.

Блок 14 – "Печать результирующего массива на ассемблере". Реализация печати результата сортировки массива на ассемблере приведена в строках 83-85. Функция `printf()`, расположенная в строке 83, выводит на печать сообщение, а такая же функция в строке 84 в цикле выводит на печать элементы исходного массива. Для вывода каждого числа отводится 8 позиций, что позволяет выводить в строке ровно 10 чисел.

Блок 16 – "Закрыть файл". Функция `fclose()` закрывает файл, который далее использоваться не будет.

Блоки 17,18,24 – Данные блоки реализует внешний оператор for в строке 42.

Блоки 19,20,23 – Эти блоки реализует внутренний оператор цикла for в строке 43.

Блок 21 – "Сравнить элементы массива (`arr[i]>arr[j]`?)". Данный блок реализует условный оператор в строке 45. Если значение элемента массива `arr[i]` больше элемента `arr[j]`, то переходим на реализацию блока 22, который обменивает местами эти два элемента.

Блок 22 – "Обменять местами текущий элемент массива и очередной элемент из правой его части". Эта операция реализуется тремя операторами в строках 47-49. При этом в качестве промежуточного элемента используется переменная `tmp`.

Рассмотрим реализацию алгоритма сортировки (блоки 17-24) на ассемблере в строках 57-81. Обратите внимание на то, что порядок реализации блоков алгоритма в ассемблере изменен для более эффективной его реализации. В частности увеличение переменной цикла и ее сравнение с пределом выполняется в конце, а не в начале цикла. Это позволяет избежать лишних команд безусловного перехода.

edx		tmp	eax
ecx	len		ebx
esi	i	j	edi

Рис.2. Распределение переменных по регистрам в программе lab5.c

Блок 17 – "Инициализировать переменную внешнего цикла I (I=0)". Данный блок реализуют

команды в строках 59-60. Команда `xor` сбрасывает счетчик цикла в регистре `esi` (переменную i) ноль, а команда `mov` в строке 60 помещает переменную в регистр-счетчик `ecx`. Это сделано для б*о*льшей эффективности программы, поскольку к переменной `len` часто обращаются в цикле.

Блок 19 – "Инициализировать переменную внутреннего цикла J (J=I+1)". Данный блок реализуют команды в строках 63-64. Команда `mov` в строке 63 создает копию переменной `i` регистре `edi` и увеличивает его на единицу в строке 64. Следующим реализуется блок 21.

В строках 65-78 расположены команды, реализующие внутренний цикл алгоритма. Начало внутреннего цикла отмечено меткой L1.

Блок 21 – "Сравнить элементы массива (`arr[i]>arr[j]`?)". Данный блок реализуют команды в строках 67, 69, 70. В строке 67 в аккумулятор (переменная `tmp`) копируется элемент массива с индексом `i`. Для этого используется обращение к элементу массива `arr_a` с индексом, расположенным в регистре `esi` и имеющего длину 4 байта. Команда, реализующая обращение к элементу массива в режиме адресации с масштабированием приводится ниже:

```
movl    arr_a(,%esi,4),%eax // tmp <- arr[i]
```

Следующая команда в строке 69 сравнивает элемент `arr[j]` с элементом а аккумуляторе. Если текущий элемент (`arr[i]`) в аккумуляторе *меньше*, то обмен выполнять не надо, и сразу переходим на метку L2, начиная с которой располагается конец цикла. Если же это не так, то надо выполнить обмен элементами, который реализуется в блоке 22.

Блок 22 – "Обменять местами текущий элемент массива и очередной элемент из правой его части". Данный блок реализуется командами обмена `xchg` в строках 72 и 74.

Блок 24 – "Увеличить переменную внешнего цикла I (I++)". В строке 76 переменная внутреннего цикла увеличивается на единицу.

Блок 18 – "I<Длина массива". Этот блок реализуют команды в строках 77-78. В строке 77 команда сравнения `cmp` сравнивает переменную внутреннего цикла `J` (регистр `edi`) с длиной массива `len` (регистр `ecx`) и, если переменная цикла меньше, управление передается на метку L1, отмечающей начало внутреннего цикла в строке 65.

Блок 21 – "Увеличить переменную внутреннего цикла J (J++)". Данный блок реализован в строке 79 с помощью команды инкремента, которая увеличивает содержимое переменной внешнего цикла I (регистр `esi`).

Блок 20 – "J<Длина массива". Этот блок реализуют команды в строках 80-81. В строке 80 команда сравнения cmp сравнивает переменную внешнего цикла `I` (регистр `esi`) с длиной массива `len` (регистр `ecx`) и, если переменная цикла меньше, управление передается на метку L0, отмечающей начало внешнего цикла в строке 62.

Текст программы с комментариями.

```
 1:  //+================================================================
 2:  // File lab_5.c
 3:  // Циклическая программа, работающая с массивом
 4:  // Вариант 37
 5:  // Программа выполняет сортировку простым выбором
 6:  // Длина масива и сам массив вводятся из файла in.txt
 7:  //
 8:  // (C) Дужий В.И., 2005
 9:  //-================================================================
10:  #include <stdio.h>
11:  #define   MAX_LENGTH  100
12:  long int  arr[MAX_LENGTH], arr_a[MAX_LENGTH];
13:  int len;
14:
15:  int main()
16:  {
17:    int i, j, tmp;
18:    FILE *fin;
19:
20:    printf("\n\t\t(C) Дужий В.И., 2005\n");
21:  // Открыть файл с исходными данными
22:    if((fin = fopen("in.txt","r"))==NULL)

23:    {
24:      printf("\nCan't open file\n"); exit(1);
25:    }
26:    for (;;)
27:    {
28:  // Читать длину масива
29:      fscanf(fin,"%i\n",&len);
30:  // Если массив имеет ненулевую и допустимую длину, читать его
31:      if (len==0 || MAX_LENGTH>100) break;
32:  // Копировать файл в массив
33:      for (i=0;i<len;i++) fscanf(fin,"%i",&arr[i]);
34:  // Сделать копию для ассемблера
35:      for (i=0;i<len;i++) arr_a[i]=arr[i];
36:      printf("\n\tСортировка массива простым выбором");
37:  //============================== C ================================
38:  // Вывод исходного массива
39:      printf("\nИсходный масив имеет длину: %i\n",len);
40:      for (i=0; i<len; i++) printf("%8i",arr[i]);
41:  // Сортировка массива
42:      for (i=0; i<len; i++)
43:        for (j=i+1; j<len; j++)
44:        {
45:          if(arr[i]>arr[j])
46:          {
47:            tmp = arr[i];
48:            arr[i] = arr[j];
49:            arr[j] = tmp;
50:          }
51:        }
52:  // Вывод результирующего массива
53:      printf("\nРезультирующий массив ( C )\n");
54:      for (i=0; i<len; i++) printf("%8i",arr[i]);
55:  //========================== Assembler ============================
56:  // Разделить исходные переменные на знаменатель DENOM
57:  asm(
```

```
58:  //    for (i=0; i<len; i++)
59:  "       xorl    %esi,%esi       ;"
60:  "       movl    len,%ecx        ;"
61:  //      for (j=i+1; j<len; j++) {
62:  "L0:                            ;"
63:  "       movl    %esi,%edi       ;"
64:  "       incl    %edi            ;"
65:  "L1:                            ;"
66:  //        tmp = arr[i];
67:  "       movl    arr_a(,%esi,4),%eax ;"
68:  //        if(arr[i]>arr[j]) {
69:  "       cmpl    arr_a(,%edi,4),%eax ;"
70:  "       jlL2             ;"
71:  //          arr[i] = arr[j];
72:  "       xchgl   %eax,arr_a(,%edi,4) ;"
73:  //          arr[j] = tmp;}
74:  "       xchgl   %eax,arr_a(,%esi,4) ;"
75:  "L2:                            ;"
76:  "       incl    %edi            ;"
77:  "       cmpl    %ecx,%edi       ;"
78:  "       jl      L1              ;"
79:  "       incl    %esi            ;"
80:  "       cmpl    %ecx,%esi       ;"
81:  "       jl      L0              ;");
82:  // Вывод результатов
83:      printf("\nРезультирующий массив (Asm)\n");
84:      for (i=0; i<len; i++) printf("%8i",arr[i]);
85:      printf("\n");
86:  }
87:    fclose(fin);// Закрыть исходный файл
88: return 0;
89:  }
```

6. Тестирование программы.

Составим тестовые примеры, которые позволят найти ошибки в программе. Для правильного выбора тестовых примеров следует проанализировать поставленную задачу и выбрать значения исходных данных, используя следующие соображения:

- исходный массив должен содержать как положительные, так и отрицательные числа.
- длина исходного массива должна быть равна нулю и больше максимально возможной (в данном случае больше 100). Выполнение этого теста должно привести к завершению программы. В приведенной ниже таблице это тесты 1 и 2.
- исходный массив должен содержать числа в порядке возрастания. В приведенной ниже таблице это тест 3.
- исходный массив должен содержать числа в порядке убывания. В приведенной ниже таблице это тест 4.
- исходный массив должен содержать числа в случайном порядке. В приведенной ниже таблице это тесты 5.
- проверить работу программы для длины массива равной 1 и 100. Это наибольшая и

наименьшая длины массива. В приведенной ниже таблице это тесты 6 и 7.

Тестовые примеры.

Номер	Длина	Исходный массив	Ожидаемый результат	Полученный результат	Цель теста
1	0				Признак конца ввода
2	101				Длина больше максимальной
3	5	-20, -10, 0, 30, 50	-20, -10, 0, 30, 50		Элементы по возрастанию
4	7	90,80,70,50,-30,-50,-70	-70,-50,-30,50,70,80,90		Элементы по убыванию
5	6	100,-4, 8,-30,-99,200	-99, -30, -4, 8, 100,200		Элементы в случайном порядке
6	1	1234	1234		Массив из одного элемента
7	100	0,-1,-2,-3,-4,-5,-6, -7,-8,-9,-10,-11…	-99,-98,-97,-96, -95,-94,-93,…		Массив из 100 элементов

Протокол тестирования программы приводится ниже.

```
                (C) Дужий В.И., 2005

        Сортировка массива простым выбором
Исходный массив имеет длину: 5
      10       9       8       7       6
Результирующий массив ( C )
       6       7       8       9      10
Результирующий массив (Asm)
       6       7       8       9      10

        Сортировка массива простым выбором
Исходный массив имеет длину: 3
     444     333     222
Результирующий массив ( C )
     222     333     444
Результирующий массив (Asm)
     222     333     444

        Сортировка массива простым выбором
Исходный массив имеет длину: 7
       7       6       5       4       3       2       1
Результирующий массив ( C )
       1       2       3       4       5       6       7
Результирующий массив (Asm)
       1       2       3       4       5       6       7
```