ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ИНДУСТРИАЛЬНЫЙ УНИВЕРСИТЕТ
ИНСТИТУТ ДИСТАНЦИОННОГО ОБРАЗОВАНИЯ

К.А. Палагута, А.И. Крюков

# МИКРОПРОЦЕССОРЫ И ИНТЕРФЕЙСНЫЕ СРЕДСТВА ТРАНСПОРТНЫХ СРЕДСТВ

*Методические указания*
*по курсовому проектированию*

Москва 2010

Рассмотрены вопросы выполнения арифметических операций над многобайтовыми числами и их сортировки. Особое внимание уделяется тонкостям выполнения логико-арифметических операций над двухбайтовыми числами, используемых при курсовом проектировании. Приводятся примеры выполнения элементов курсовой работы для ассемблера микропроцессора КР580ВМ80, на основе типового варианта задания.

Предназначены для студентов, обучающихся по специальности 220301 (210200) «Автоматизация технологических процессов и производств в машиностроении» и специализации 46 «Автоматические и электронные системы транспортных средств». Могут быть использованы в курсах «Микропроцессорные и интерфейсные средства», «Микропроцессоры и интерфейсные средства транспортных средств».

Рецензент
Кузнецов А.В., к.т.н.,
доцент ГОУ МГИУ

Рекомендованы к изданию на заседании кафедры : Естественнонаучных и инженерно-технических дисциплин
протокол №6 от 07.07.2010 г.

Редактор *Н.А. Киселёва*
Компьютерная верстка *М.А. Махониной*

Санитарно-эпидемиологическое заключение
№ 77.99.60.953.Д.006314.05.07 от 31.05.2007

Подписано в печать 14.10.10
Формат 60x84/16. Изд. № 122/10-м
Усл. печ. л. 5,5. Уч.-изд. л. 4,0. Тираж 300 экз.
Заказ № 312

Издательство МГИУ, 115280, Москва, Автозаводская, 16
www.izdat.msiu.ru; e-mail: izdat@msiu.ru; тел. (495) 620-39-90
**По вопросам приобретения продукции издательства МГИУ обращаться по адресу:**
115280, Москва, Автозаводская, 16
www.izdat.msiu.ru; e-mail: izdat@msiu.ru; тел. (495) 620-39-92
Отпечатано в типографии издательства МГИУ

# ОГЛАВЛЕНИЕ

Введение ..... 4

1. Сортировка по возрастанию всех четных чисел с учетом их модулей ..... 5

1.1. Выборка четных чисел ..... 5

1.2. Определение модулей четных чисел ..... 12

1.3. Сортировка модулей четных чисел ..... 21

1.4. Возврат к знаковым значениям чисел ..... 39

2. Нахождение среднего арифметического всех чисел ..... 48

2.1. Нахождение суммы всех чисел ..... 48

2.2. Выполнение операции деления суммы всех чисел на их количество ..... 56

3. Компиляция программы ..... 66

Заключение ..... 79

Список литературы ..... 80

Приложения ..... 81

# ВВЕДЕНИЕ

Курсовая работа по дисциплине «Микропроцессоры и интерфейсные средства транспортных средств» заключается в написании программы на языке ассемблера КР580ВМ80. Задание формируется в зависимости от сложности для массива из N двухбайтовых или трехбайтовых чисел и включает в простейшем случае два пункта:

1) сортировка массива чисел по убыванию (возрастанию);
2) нахождение среднего арифметического чисел для всего массива или его части.

Задание может усложняться, например, введением требования сортировки чисел с учетом модуля или, например, сортировки только четных (нечетных, положительных либо отрицательных) чисел. Вместо нахождения среднего арифметического может быть предложено вычислить дисперсию для всего массива или его части.

Рассмотрим подробно последовательность действий при разработке программы.

Пусть по заданию требуется для массива, состоящего из 15 двухбайтовых чисел:

1) выполнить сортировку по возрастанию всех четных чисел с учетом их модулей;
2) найти среднее арифметическое всех чисел.

Разобьем данные задачи на более простые подцели:

1. Выполнить сортировку по возрастанию всех четных чисел с учетом их модулей.
   1.1. Выборка четных чисел.
   1.2. Определение модулей четных чисел.
   1.3. Сортировка модулей четных чисел.
   1.4. Возврат к знаковым значениям чисел.
2. Найти среднее арифметическое всех чисел.
   2.1. Нахождение суммы всех чисел.
   2.2. Выполнение операции деления суммы всех чисел на их количество.

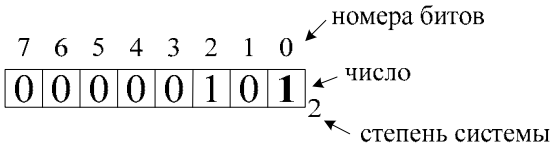
Теперь остановимся на каждой из них более подробно.

# 1. СОРТИРОВКА ПО ВОЗРАСТАНИЮ ВСЕХ ЧЕТНЫХ ЧИСЕЛ С УЧЕТОМ ИХ МОДУЛЕЙ

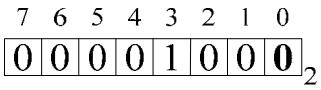
## 1.1. Выборка четных чисел

Для начала рассмотрим анализ четности числа на примере однобайтовых чисел. Четность числа в любой системе счисления можно определить путем деления числа на 2. В этом случае если остаток от деления равен нулю, то число четное, если нет – нечетное. Для чисел, записанных в двоичной системе, четность можно еще определить по наличию 1 в нулевом разряде байта, т.е. если там записана 1, то число нечетное, если там 0 – четное:

число $05_{16}$ = 00000101$_2$ – нечетное, т.к. в младшем разряде записана 1




число $08_{16}$ =00001000$_2$ – четное, т.к. в младшем разряде записан 0




На языке ассемблера проверить число на четность можно путем маскирования битов. В нашем случае для проверки наличия 1 в младшем разряде необходимо выполнить операцию логического умножения проверяемого числа на число $01_{16}$ = 00000001$_2$. Это можно сделать с помощью команды ANI 01, например, для чисел $08_{16}$ и $05_{16}$:

0800) ANI 01

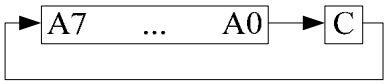
| Адрес/регистр | Код до выполнения операции | Код после выполнения операции |
|---|---|---|
| 0800 | E6 | E6 |
| 0801 | 01 | 01 |
| A | $08_{16}$ =00001000$_2$ | $00_{16}$=00000000$_2$ |
| FL | 02 | 46 (Флаг Tz=1) |
| PC | 0800 | 0802 |

0802) ANI 01

| Адрес/регистр | Код до выполнения операции | Код после выполнения операции |
|---|---|---|
| 0802 | E6 | E6 |
| 0803 | 01 | 01 |
| A | $05_{16}$ =$00000101_2$ | $01_{16}$=$00000001_2$ |
| FL | 02 | 02 (Флаг Tz=0) |
| PC | 0800 | 0804 |

Здесь следует обратить внимание на состояние флага нулевого результата Tz. Если он равен 1, то число четное, если же он равен 0 – нечетное.

Другой способ анализа четности числа состоит сдвиге этого числа вправо на один разряд (с помощью команды RAR, либо RRC) с последующим анализом флага Tc. Если Tc = 1, то число нечетное, если Tc = 0 – четное. Приведем пример для чисел $04_{16}$ и $07_{16}$ с использованием команды RAR:




0804) RAR

| Адрес/регистр | Код до выполнения операции | Код после выполнения операции |
|---|---|---|
| 0804 | 1F | 1F |
| A | $04_{16}$ =$00000100_2$ | $00_{16}$=$00000010_2$ |
| FL | 02 | 03 (Флаг Tc=0) |
| PC | 0804 | 0805 |

0805) RAR

| Адрес/регистр | Код до выполнения операции | Код после выполнения операции |
|---|---|---|
| 0805 | 1F | 1F |
| A | $05_{16}$ =$00000111_2$ | $01_{16}$=$10000011_2$ |
| FL | 03 | 03 (Флаг Tc=1) |
| PC | 0805 | 0806 |

Для обработки массива чисел нужно указать адрес первого числа массива. Это можно сделать, например, с помощью команды lxi b, 0A00. Здесь адрес 0A00 – адрес первого элемента, записанный в регистровой паре BC. Пусть четное число будет выписано в отдельный массив с адресом первого элемента 0A30, для этого используем команду lxi d, 0A30. Поскольку нам предстоит обработать 15 чисел, то разумнее всего будет организовать цикл, который на каждой своей итерации будет проверять одно

число из массива на четность. Условием выхода из цикла будет достижение последнего числа массива (или адреса 0A0F). Одновременно будем вести подсчет количества четных чисел (это нам пригодится позднее). Ниже приведены блок-схема алгоритма (рис 1.1) и листинг программы для массива однобайтовых чисел.

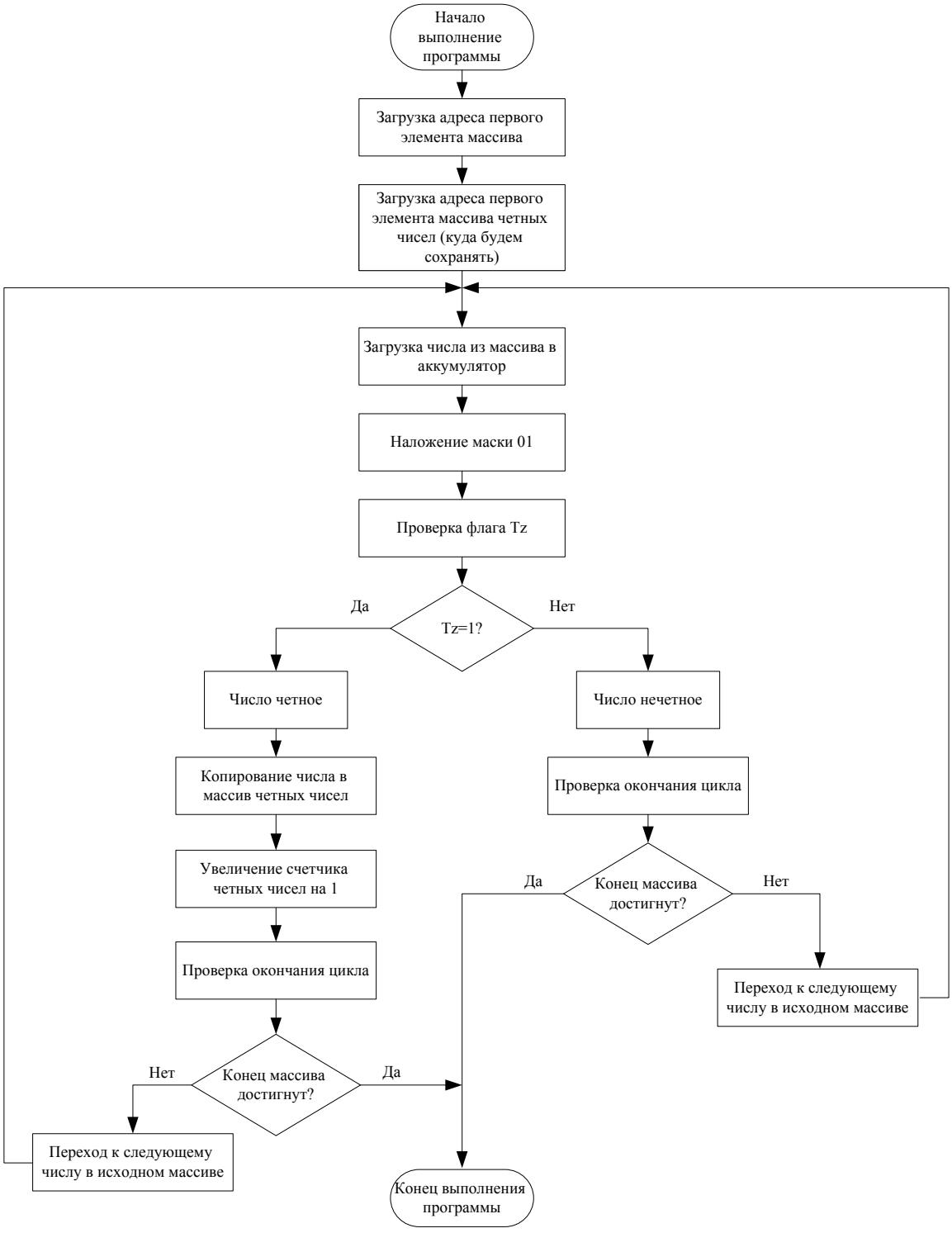



*Рис 1.1. Выборка четных чисел из массива однобайтовых чисел*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | lxi b, 0A00 | Загрузка начальных адресов |
| | lxi d, 0A30 | |
| Label2: | | |
| | ldax b | Загрузка числа из исходного массива в аккумулятор |
| | ani 01 | Проверка на четность (если нечетно, то переходим на Label1, если четно – выполняем следующую комманду) |
| | jnz Label1 | |
| | ldax b | Пересохраняем на новый адрес, если число четное |
| | stax d | |
| | inx d | Увеличиваем адрес для перехода к следующему элементу массива |
| | lda 0AA0 | Загружаем счетчик четных чисел с адреса 0AA0 |
| | adi 01 | Увеличиваем счетчик четных чисел на 1 и сохраняем его по адресу 0AA0 |
| | sta 0AA0 | |
| Label1: | | |
| | inx b | Увеличиваем адрес для перехода к следующему элементу массива |
| | mvi a, 0F | Проверяем окончание цикла, если адрес в регистре C < 0F, то продолжаем выполнение цикла, если C=0F – прекращаем выполнение программы |
| | sub c | |
| | jnz Label2 | |
| | rst1 | |

Теперь расширим этот пример до двухбайтовых чисел, как от нас требуется по заданию. Принципиальным условием работы с такими числами является расположение байтов – старший байт будем располагать на старшем адресе, а младший на младшем, что соответствует идеологии разработчиков компании Intel.

Поскольку каждое число массива содержит не один, а два байта, то и количество адресов занимаемых массивом возрастет в два раза, а следовательно, и условие окончания цикла будет уже не 0F, а 1E. Для определения четности двухбайтовых чисел нужен только младший из двух байтов, его мы и будем проверять на четность, а старший в случае, если число окажется четным, будем просто копировать в массив четных чисел вместе с младшим. Если же число окажется нечетным, то нужно будет просто перейти на обработку младшего байта следующего числа, т.е. увеличить адрес не на один, как это было сделано в случае с однобайтовыми числами, а на два.

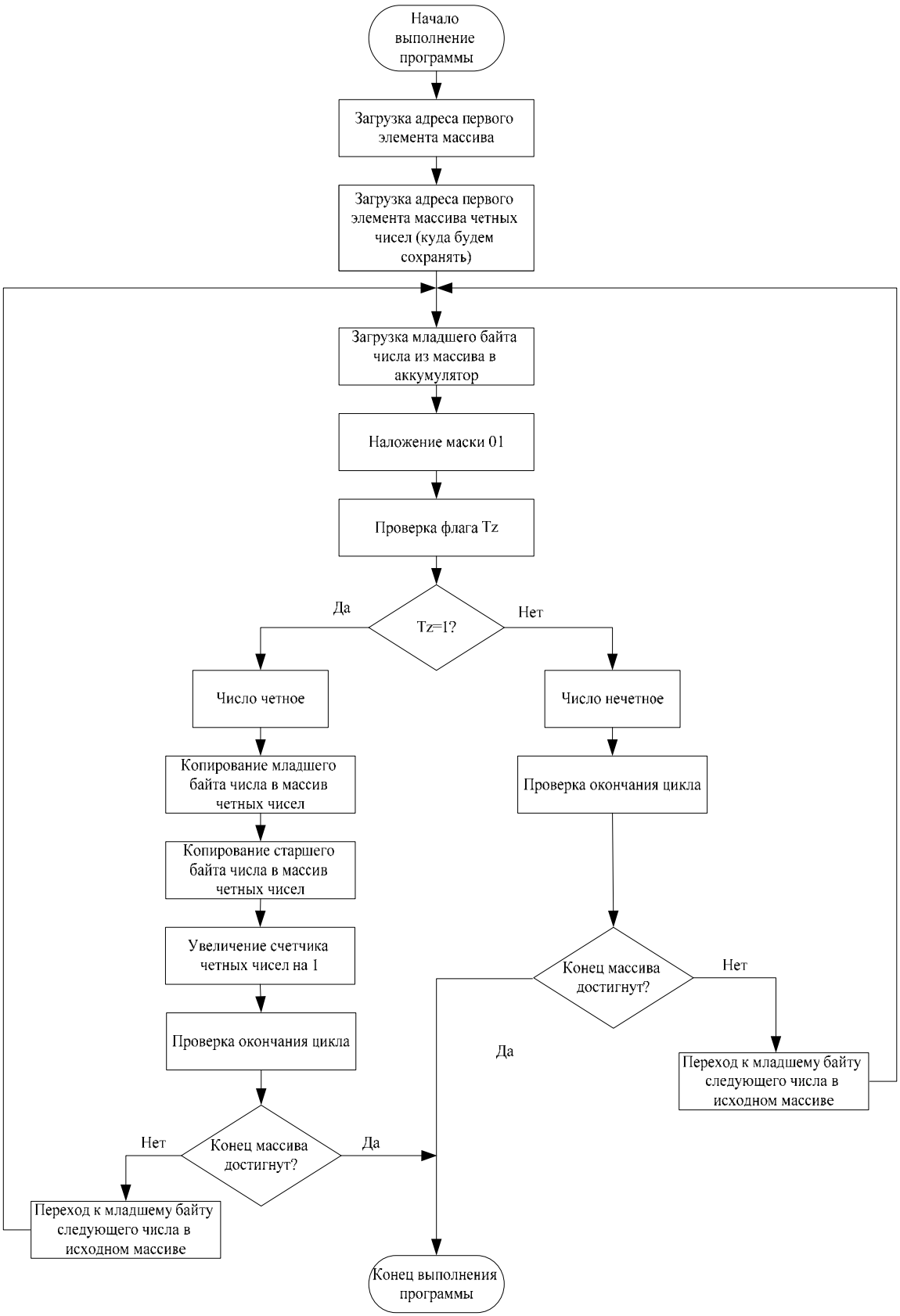



*Рис 1.2. Выборка четных чисел из массива двухбайтовых чисел*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | lxi b, 0A00 | Загрузка начальных адресов |
| | lxi d, 0A30 | |
| Label2: | | |
| | ldax b | Загрузка числа из массива |
| | ani 01 | Проверка на четность (если нечетно, то переходим на Label1, если четно – выполняем следующую комманду) |
| | jnz Label1 | |
| | ldax b | Сохраняем **младший байт** четного числа на новом адресе |
| | stax d | |
| | inx b | Увеличиваем адрес для перехода к следующему байту исходного числа |
| | inx d | Увеличиваем адрес для перехода к следующему байту четного числа |
| | ldax b | Сохраняем **старший байт** четного числа на новом адресе |
| | stax d | |
| | dcx b | Уменьшаем адрес для возврата к младшему байту |
| | inx d | Увеличиваем адрес для массива четных чисел |
| | lda 0AA0 | Загружаем счетчик четных чисел с адреса 0AA0 |
| | adi 01 | Увеличиваем счетчик четных чисел на 1 и сохраняем его по адресу 0AA0 |
| | sta 0AA0 | |
| Label1: | | |
| | inx b | **Дважды увеличиваем адрес** для перехода к младшему байту следующего числа исходного массива |
| | inx b | |
| | mvi a, 1E | Проверяем окончание цикла, если адрес в регистре C < 1E, то продолжаем выполнение цикла, если C=1E – прекращаем выполнение программы |
| | sub c | |
| | jnz Label2 | |
| | rst1 | |

По аналогии с рассмотренным примером можно проверять число на знак, в этом случае нас будет интересовать уже не младший, а старший бит числа, т.к. именно он отвечает за знак. Если в нем записана 1, то число отрицательное, если там 0 – положительное. Так, например, число $A6_{16}$ = **1**$0100110_2$ отрицательное, т.к. в старшем разряде записана 1, а число $2F_{16}$ = **0**$0101111_2$ положительное, т.к. в старшем разряде записан 0. Нетрудно догадаться, что в этом случае в качестве маски будет выступать число $80_{16}$=$10000000_2$.

В случае работы с двухбайтовыми числами нас уже будет интересовать не младший, а старший байт числа. Ниже приведены блок-схема алгоритма и листинг программы для выборки всех положительных из массива 15 двухбайтовых чисел (рис. 1.3).

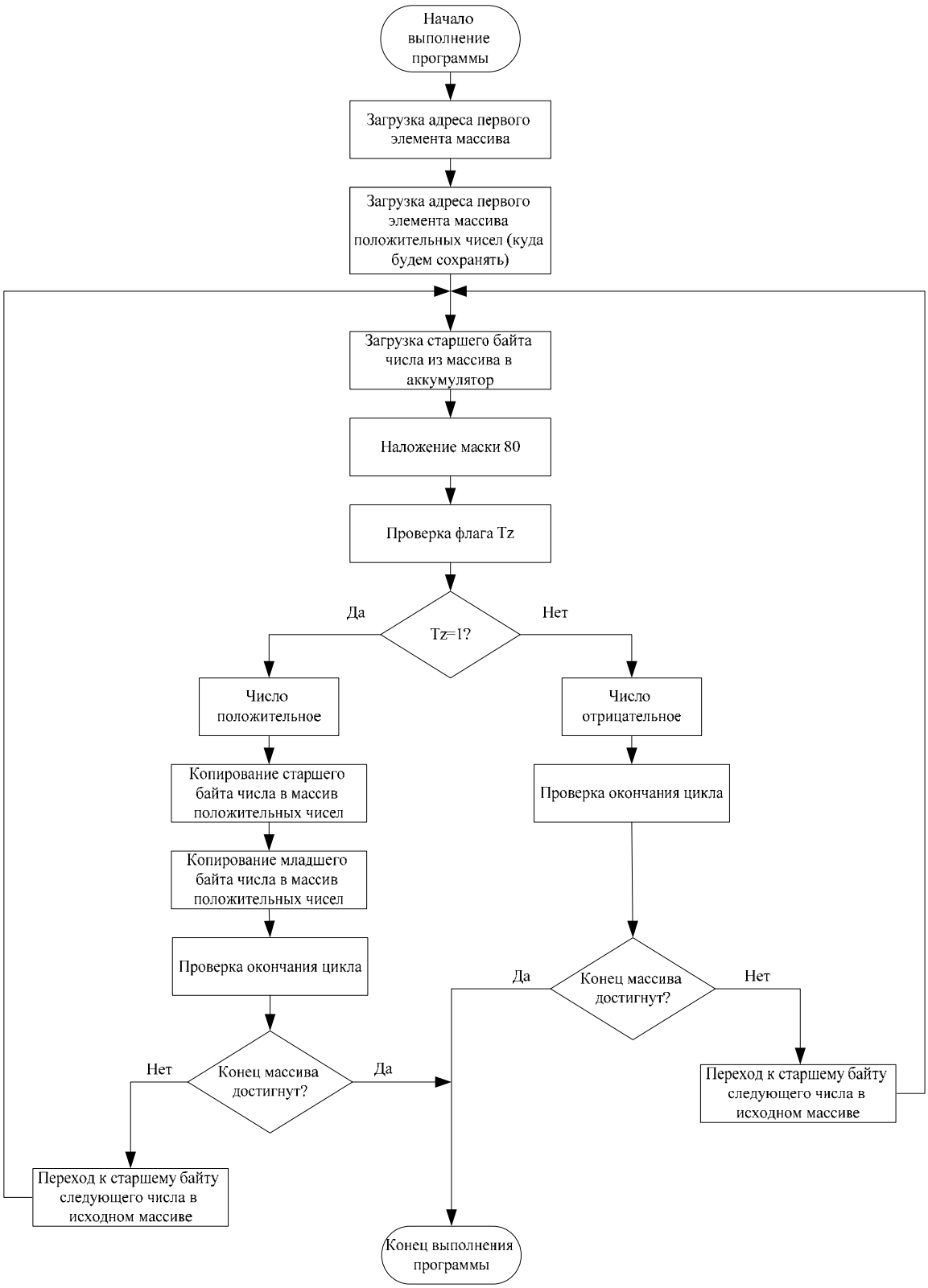



*Рис. 1.3. Выборка положительных чисел из массива двухбайтовых чисел*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | lxi b, 0A00 | Загрузка начальных адресов |
| | lxi d, 0A30 | |
| Label2: | | |
| | inx b | Переходим к **старшему байту** числа |
| | ldax b | Загрузка числа из исходного массива |
| | ani 80 | Проверка на знак (если отрицательно, то переходим на Label1, |
| | jnz Label1 | если положительно – выполняем следующую комманду) |
| | inx d | Переходим к старшему байту в массиве положительных |
| | ldax b | Копируем старший байт в массив положительных чисел |
| | stax d | |
| | dcx b | Переходим к младшему байту числа в исходном массиве |
| | dcx d | Переходим к младшему байту числа в массиве положительных |
| | ldax b | Копируем младший байт в массив положительных чисел |
| | stax d | |
| | lda 0AC0 | Загружаем счетчик четных чисел с адреса 0AA0 |
| | adi 01 | Увеличиваем счетчик четных чисел на 1 и сохраняем его по адресу |
| | sta 0AC0 | 0AA0 |
| | inx d | Переходим к следующему числу в массиве положительных чисел |
| | inx d | |
| | inx b | Переходим к старшему байту следующего числа в исходном |
| Label1: | | массиве |
| | inx b | |
| | mvi a, 1E | Проверяем окончание цикла, если адрес в регистре C < 1E, |
| | sub c | то продолжаем выполнение цикла, если C=1E– прекращаем |
| | jnz Label2 | выполнение программы |
| | rst1 | |

## 1.2. Определение модулей четных чисел

Модуль числа x – неотрицательное число, обозначаемое |x| и определяемое следующим образом:

1) если $x \geq 0$, то $|x| = x$;
2) если $x < 0$, то $|x| = -x$.

Другими словами, модуль числа – это число, взятое без знака. В двоичной системе, как это уже было описано выше, знак определяется наличием 1 в старшем разряде числа. Следовательно, для однобайтовых чисел максимальное положительное число – $7F_{16} = 01111111_2$, а минимальное отрицательное – $81_{16} = 10000001_2$.

Так как модуль – число неотрицательное, то для любого отрицательного числа нужно получить его значение без знака (его прямой код). Для этого необходимо отрицательное число, представленное в обратном дополнительном коде, проинвертировать и прибавить к нему единицу, например:

число $81_{16} = 10000001_2$, после инвертирования получим $01111110_2$, теперь прибавим к результату 1:

$$\begin{array}{l} 01111110_2 \\ +\underline{00000001_2} \\ 01111111_2 \rightarrow \quad 01111111_2 = 7F_{16}, \end{array}$$

т.е. число $7F_{16}$ и есть модуль числа $81_{16}$, или по-другому число $81_{16}$, взятое без знака.

Таким образом, для получения модуля числа требуется выполнить 3 условия:

1) определить знак числа;

2) если число отрицательное, то получить его абсолютное значение (значение без знака);

3) если число положительное, то не выполнять с ним никаких действий.

Как определить знак числа, было рассмотрено выше, поэтому остановимся подробнее на вычислении модуля.

На языке ассемблера инвертирование числа можно осуществить с помощью команды CMA, а прибавление 1 с помощью команды ADI 01. Например, получим модуль числа C4:

0806) CMA

| Адрес/регистр | Код до выполнения операции | Код после выполнения операции |
|---|---|---|
| 0804 | 2F | 2F |
| A | $C4_{16} = 11000100_2$ | $3B_{16} = 00111011_2$ |
| FL | 02 | 02 |
| PC | 0806 | 0807 |

0807) ADI 01

| Адрес/регистр | Код до выполнения операции | Код после выполнения операции |
|---|---|---|
| 0807 | C6 | C6 |
| 0808 | 01 | 01 |
| A | $3B_{16}=00111011_2$ | $3C_{16}=00111100_2$ |
| FL | 02 | 06 |
| PC | 0807 | 0809 |

Особым случаем является вычисление модуля числа 80 – так называемого отрицательного нуля. Это связано с тем, что при инвертировании и прибавлении 1 к результату мы снова получим число 80:

$81_{16} = 10000000_2$, после инвертирования получим $01111111_2$, теперь прибавим к результату 1:

$$\begin{array}{l} \overset{1\,1\,1\,1\,1\,1\,1}{} \\ +\ 01111111_2 \\ \underline{\quad 00000001_2} \\ 10000000_2 \rightarrow \quad 10000000_2 = 80_{16} \end{array}$$

В связи с этим приходится обрабатывать это число отдельно, считая, что его модулем является число $00_{16}$. Проверить на появление отрицательного нуля можно с помощью команды сравнения CPI 80.

Напишем программу, которая будет для массива из 15 однобайтовых чисел, расположенных начиная с 0A30, получать их модули и записывать их по новым адресам, начиная с 0A50. Условием окончания цикла будет достижение адреса последнего элемента – 0A3F. Блок-схема алгоритма этой программы приведена на рис 1.4.

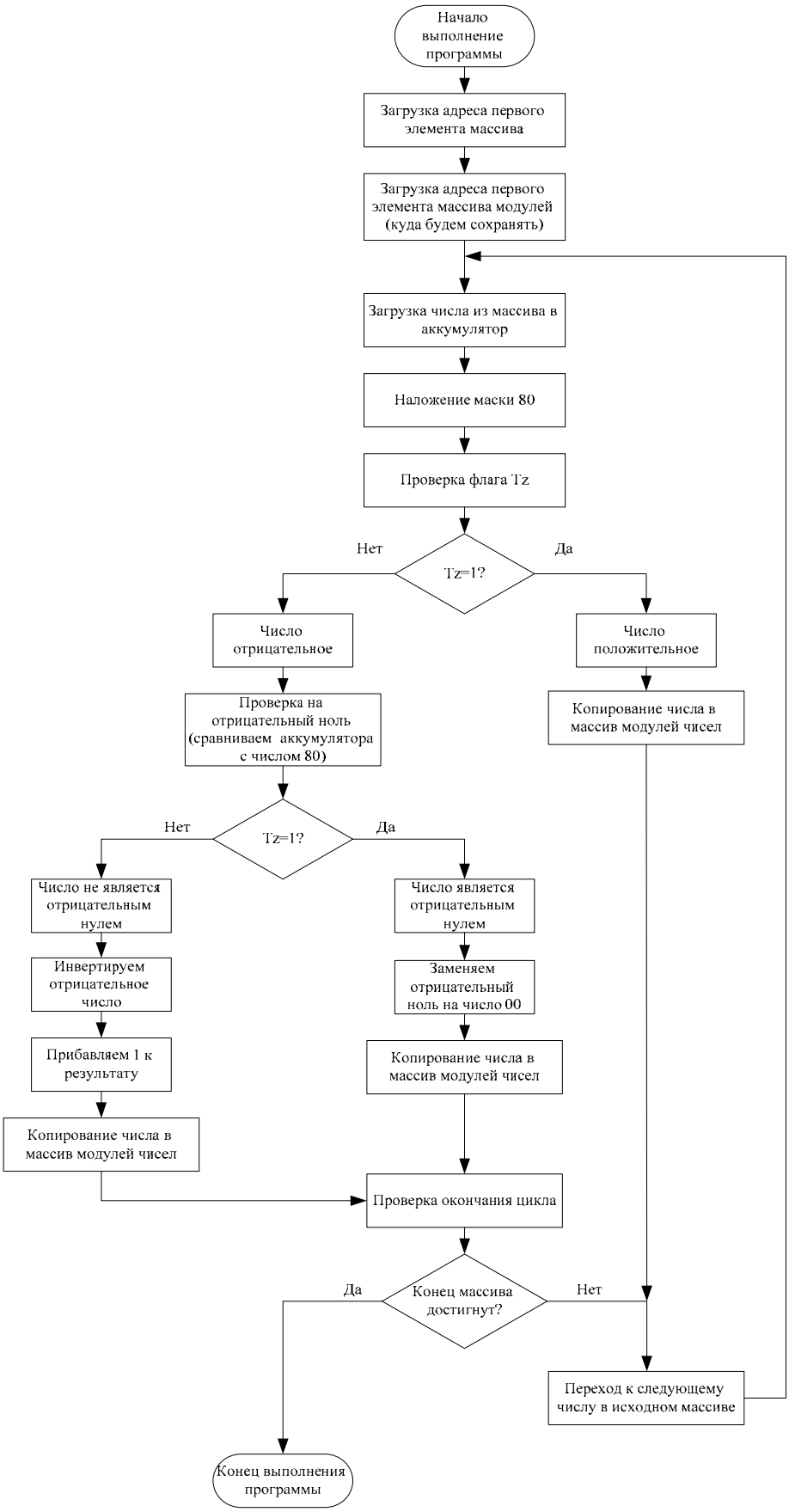



*Рис. 1.4. Блок-схема алгоритма определения модуля однобайтовых чисел*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | lxi b, 0A30 | Загрузка начальных адресов |
| | lxi d, 0A50 | |
| Label2: | | |
| | ldax b | Загрузка числа из исходного массива |
| | ani 80 | Проверка на знак |
| | ldax b | Загрузка числа из исходного массива |
| | cnz Label1 | Если число отрицательно, то вызываем подпрограмму на Label1, если положительно – выполняем следующую команду |
| | stax d | Пересохраняем на новый адрес |
| | inx b | Увеличиваем адрес для перехода к следующему элементу исходного массива |
| | inx d | Увеличиваем адрес для перехода к следующему элементу массива модулей чисел |
| | mvi a, 3F | Проверяем окончание цикла, если адрес в регистрах B и C ≠ (т.е. <) 3F, то продолжаем выполнение цикла, если C = 3F– прекращаем выполнение программы |
| | sub c | |
| | jnz Label2 | |
| | rst1 | |
| | | |
| Label1: | | Подпрограмма получения абсолютного значения числа |
| | cpi 80 | Проверка на возникновение отрицательного нуля |
| | jnz Label3 | Если возник отрицательный ноль, заменяем его на 00 |
| | mvi a, 00 | |
| | ret | Возврат из подпрограммы, если был отрицательный ноль |
| Label3: | | |
| | cma | Инвертирование числа |
| | adi 01 | Прибавление 1 к результату |
| | ret | Возврат из подпрограммы, если отрицательного нуля не было |

Теперь расширим этот пример для работы с двухбайтовыми числами. Здесь возникает трудность получения модуля – при инвертировании числа и прибавлении 1 к результату может возникнуть перенос между байтами, например:

число $A300_{16}$=10100011 $00000000_2$, после инвертирования получим число

01011100 $11111111_2$, теперь прибавим к результату 1:

$$\begin{array}{r} \overset{\curvearrowleft}{1}\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\ 01011100\ 11111111_2 \\ +\ 00000000\ 00000001_2 \\ \hline 01011101\ 00000000_2 \end{array} \quad \rightarrow \quad 01011101\ 00000000_2 = 5D00_{16}$$

Для программного осуществления этого переноса необходимо после прибавления 1 к младшему байту прибавить значение флага переноса Tc к старшему. Это можно сделать с помощью команды ACI 00:

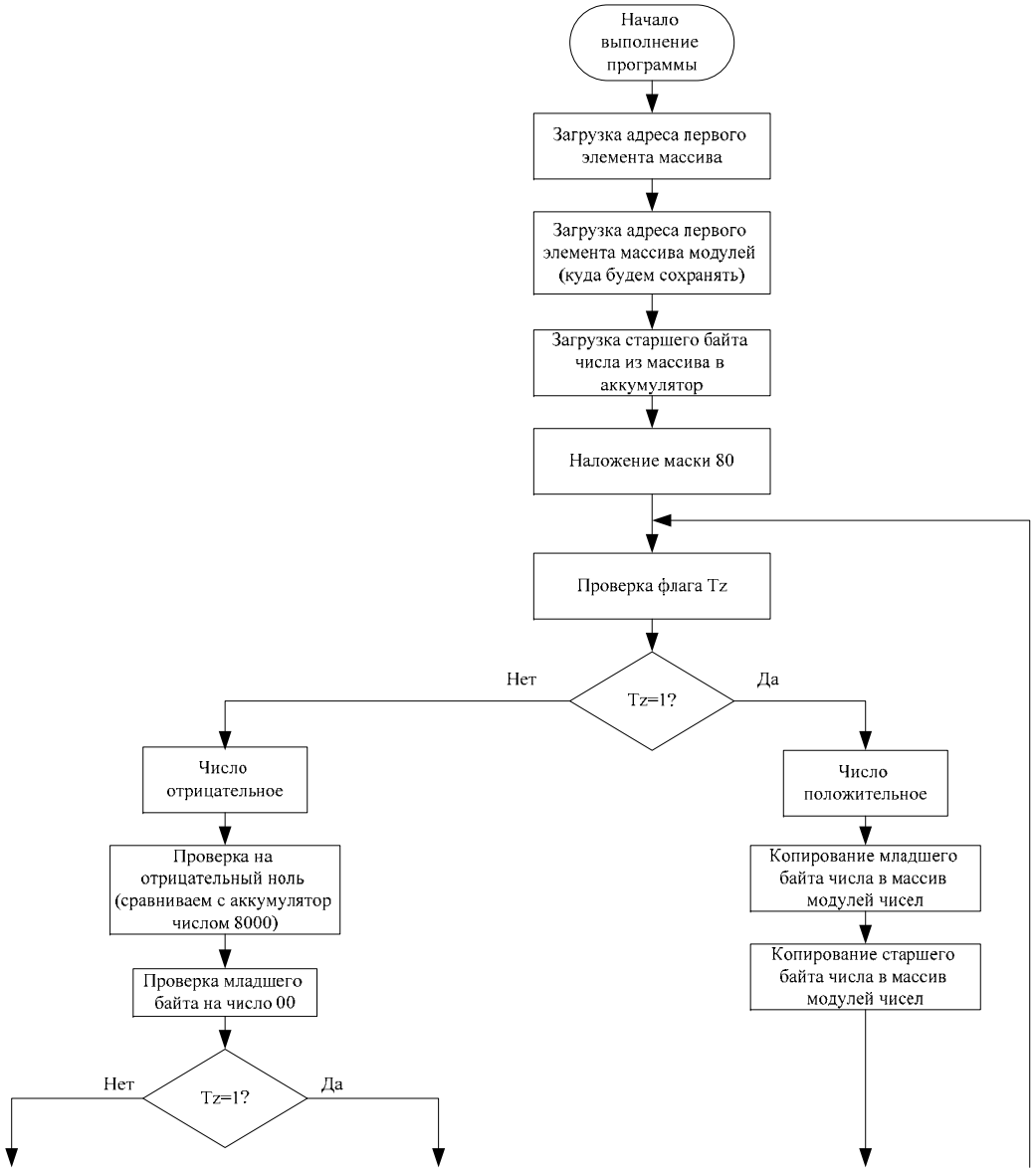
0806) ACI 00

| Адрес/регистр | Код до выполнения операции | Код после выполнения операции |
|---|---|---|
| 0806 | CE | CE |
| 0807 | 00 | 00 |
| A | $5C_{16}=01011100_2$ | $5D_{16}=01011101_2$ |
| FL | 03(Флаг Tc=1) | 02(Флаг Tc=0) |
| PC | 0806 | 0808 |

Таким образом, для прибавления 1 к двухбайтовому числу нам потребуются две команды – сначала ADI 01, а затем ACI 00. Первая из них прибавит 1 к младшему байту, а вторая – значение флага Tc к старшему байту:

| Мнемоника | Комментарий |
|---|---|
| ldax b | Загружаем младший байт числа |
| adi 01 | Прибавление 1 к младшему байту |
| stax b | Пересохраняем младший байт числа |
| inx b | Увеличиваем адрес для перехода к старшему байту |
| ldax b | Загружаем старший байт числа |
| aci 00 | Прибавление флага переноса Tc к старшему байту |
| stax b | Пересохраняем старший байт числа |

Также изменится и условие окончания цикла получения модулей чисел. Им станет достижение адреса 0A4E (рис. 1.5).


Начало выполнение программы
Загрузка адреса первого элемента массива
Загрузка адреса первого элемента массива модулей (куда будем сохранять)
Загрузка старшего байта числа из массива в аккумулятор
Наложение маски 80
Проверка флага Tz
Tz=1?
Нет
Да
Число отрицательное
Проверка на отрицательный ноль (сравниваем с аккумулятор числом 8000)
Проверка младшего байта на число 00
Tz=1?
Нет
Да
Число положительное
Копирование младшего байта числа в массив модулей чисел
Копирование старшего байта числа в массив модулей чисел

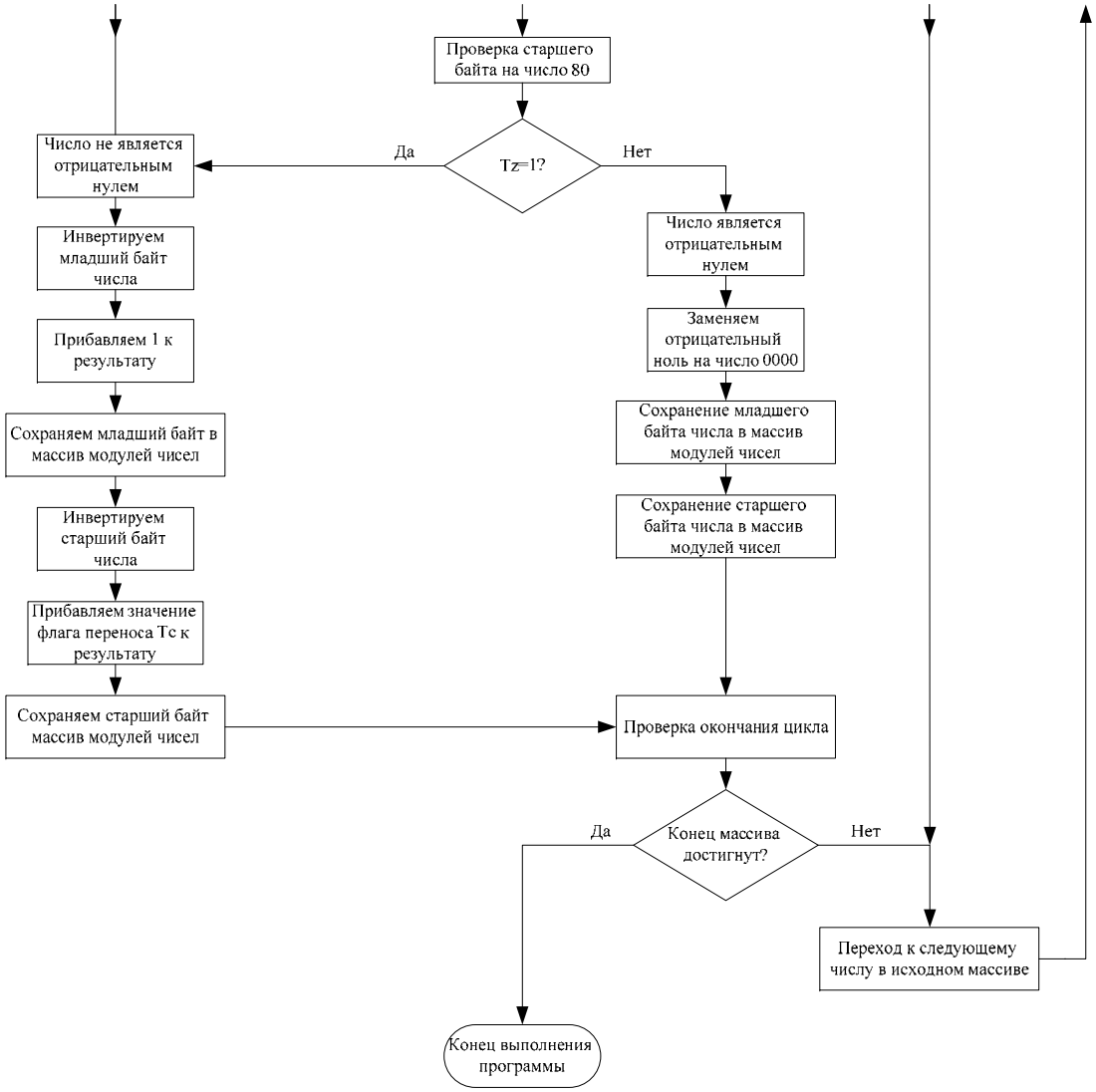



*Рис. 1.5. Блок-схема алгоритма определения модуля двухбайтовых чисел*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | lxi b, 0A30 | Загрузка начальных адресов |
| | lxi d, 0A50 | |
| Label2: | | |
| | inx b | Переходим к **старшему** байту числа в исходном массиве |
| | ldax b | Загрузка старшего байта числа |
| | ani 80 | Проверка на знак |
| | dcx b | Переходим к младшему байту числа в исходном массиве |
| | ldax b | Загрузка младшего байта числа |
| | jnz Label1 | Если число отрицательно, то переходим на подпрограмму на Label1, если положительно – выполняем следующую команду |
| | stax d | Пересохраняем младший байт числа на новый адрес |
| | inx b | Увеличиваем адрес для перехода к старшему байту числа в исходном массиве |
| | inx d | Увеличиваем адрес для перехода к старшему байту числа в массиве модулей чисел |
| | ldax b | Загрузка старшего байта числа |
| Label4: | | |
| | stax d | Пересохраняем старший байт числа на новый адрес |
| | inx b | Увеличиваем адрес для перехода к следующему элементу исходного массива |
| | inx d | Увеличиваем адрес для перехода к следующему элементу массива модулей чисел |
| | mvi a, 4E | Проверяем окончание цикла, если адрес в регистрах B и C ≠ (т.е. <) 4E, то продолжаем выполнение цикла, если C=4E – прекращаем выполнение программы |
| | sub c | |
| | jnz Label2 | |
| | rst1 | |
| | | |
| Label1: | | Подпрограмма получения абсолютного значения числа |
| | cpi 00 | Проверка на возникновение отрицательного нуля |
| | jnz Label3 | Если в младшем байте 00, то переходим к проверке старшего на число 80, если нет – получаем модуль |
| | inx b | |
| | ldax b | Загрузка старшего байта числа |
| | cpi 80 | Проверка на возникновение отрицательного нуля |
| | dcx b | Переходим к младшему байту числа |
| | ldax b | Загружаем младший байт числа |
| | jnz Label3 | Если возник отрицательный ноль, заменяем его на 0000. |
| | mvi a, 00 | |
| | stax d | Сохраняем младший байт числа на новый адрес |
| | inx b | Увеличиваем адрес для перехода к старшему байту числа в исходном массиве |
| | inx d | Увеличиваем адрес для перехода к старшему байту числа в массиве модулей чисел |
| | jmp Label4 | Возврат из подпрограммы, если был отрицательный ноль |
| Label3: | | |
| | cma | Инвертирование младшего байта числа |
| | adi 01 | Прибавление 1 к результату |
| | stax d | Пересохраняем младший байт числа на новый адрес |
| | inx b | Увеличиваем адрес для перехода к старшему байту числа в исходном массиве |
| | inx d | Увеличиваем адрес для перехода к старшему байту числа в массиве модулей чисел |
| | ldax b | Загружаем старший байт числа из исходного массива |
| | cma | Инвертирование старшего байта числа |
| | aci 00 | Прибавление флага переноса Tc к результату |
| | jmp Label4 | Возврат из подпрограммы, если отрицательного нуля не было |

## 1.3. Сортировка модулей четных чисел

Сортировка чисел – это их упорядочение в соответствие с заданным критерием (например, по возрастанию). Существуют различные алгоритмы сортировки чисел: сортировка вставкой, выбором, подсчетом, слиянием и т.п. Мы же воспользуемся одним из самых простых – методом пузырька.

Алгоритм состоит в повторяющихся проходах по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. На рис. 1.6 показан пример для сортировки по возрастанию.

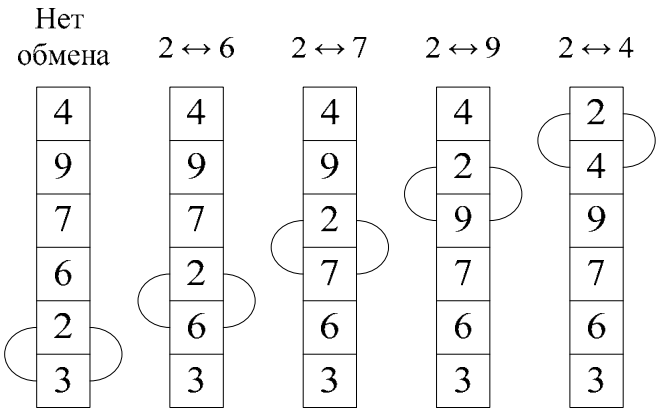



*Рис. 1.6. Сортировка по возрастанию методом пузырька (первый проход, сравниваемые пары выделены)*

Проходы по массиву повторяются до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает – массив отсортирован.

При реализации алгоритма элемент, стоящий не на своём месте, «всплывает» до нужной позиции как пузырёк в воде, отсюда и название алгоритма.

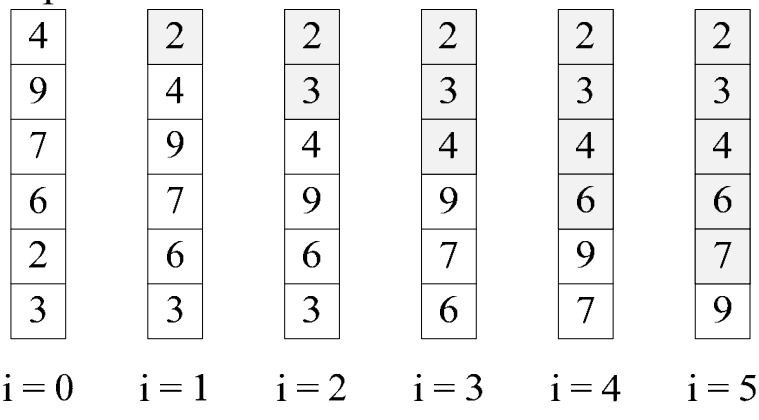



*Рис. 1.7. Сортировка по возрастанию методом пузырька (все проходы)*

Теперь напишем непосредственно программу сортировки для 15 однобайтовых чисел по убыванию. Для этого нам потребуется организовать два цикла – один внешний (будет отвечать за количество проходов), а второй внутренний или вложенный (будет отвечать за сравниваемые пары).

Условием выхода из вложенного цикла будет достижение в процессе сравнения пар чисел последнего числа массива (будем уменьшать счетчик четных чисел из адреса 0AA0), а из внешнего – достижение нулевого значения счетчиком проходов (будем уменьшать счетчик с 0F до нуля).

Числа будем сравнивать с помощью команды CMP, т.е. путем вычитания одного числа из другого. Можно было бы воспользоваться и командой вычитания SUB, но в данном случае удобнее команда сравнения, поскольку нам не требуется сохранять разность чисел, нам важен факт – какое число больше.

0807) CMP D

| Адрес/регистр | Код до выполнения операции | Код после выполнения операции |
|---|---|---|
| 0807 | BA | BA |
| A | $2D_{16}=00101101_2$ | $2D_{16}=00101101_2$ |
| D | $49_{16}=01001001_2$ | $49_{16}=01001001_2$ |
| FL | 02(Флаг Tc=0, Ts=0) | 87(Флаг Tc=1, Ts=1) |
| PC | 0808 | 0809 |

Поскольку мы работаем с модулями чисел, то нас интересует состояние либо флага отрицательного результата (Ts), либо флага заема (Tc). В данном случае если вычитаемое больше уменьшаемого, то оба этих флага будут равны единице, если нет – нулю. Следовательно, если флаги принимают значение 1, то при сортировке по убыванию необходимо произвести обмен чисел местами, если они принимают значение 0 – перейти к следующей паре чисел.

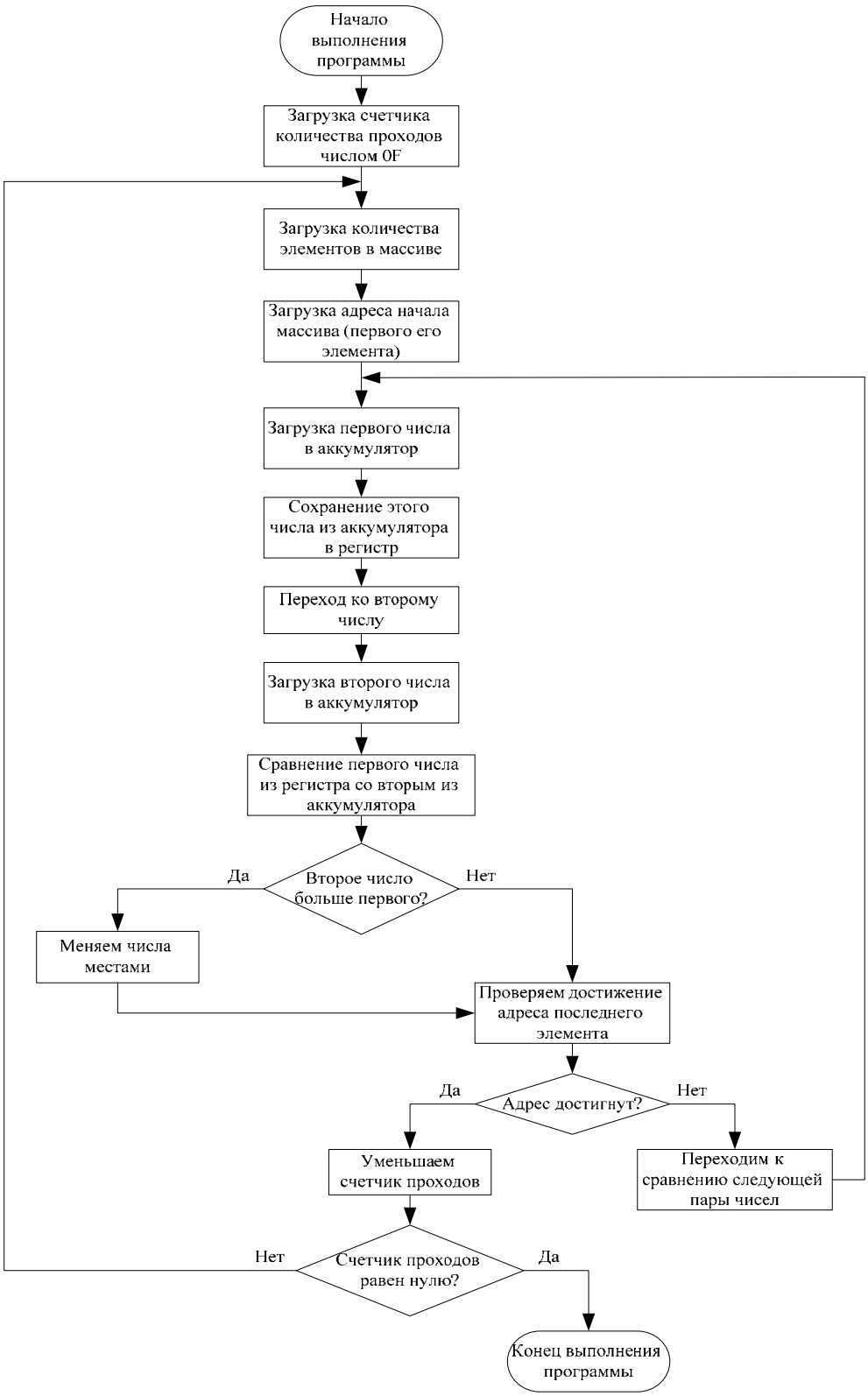



*Рис. 1.8. Сортировка для 15 однобайтовых чисел по убыванию*

Если необходимо сортировать числа по возрастанию, то достаточно изменить условие, стоящее после команды сравнения с JC на JNC. Таким образом, если флаг Tc принимает значение 0,

то необходимо произвести обмен чисел местами, если он принимает значение 1 – перейти к следующей паре чисел. Блок-схема алгоритма для этого случая приведена на рис. 1.9.

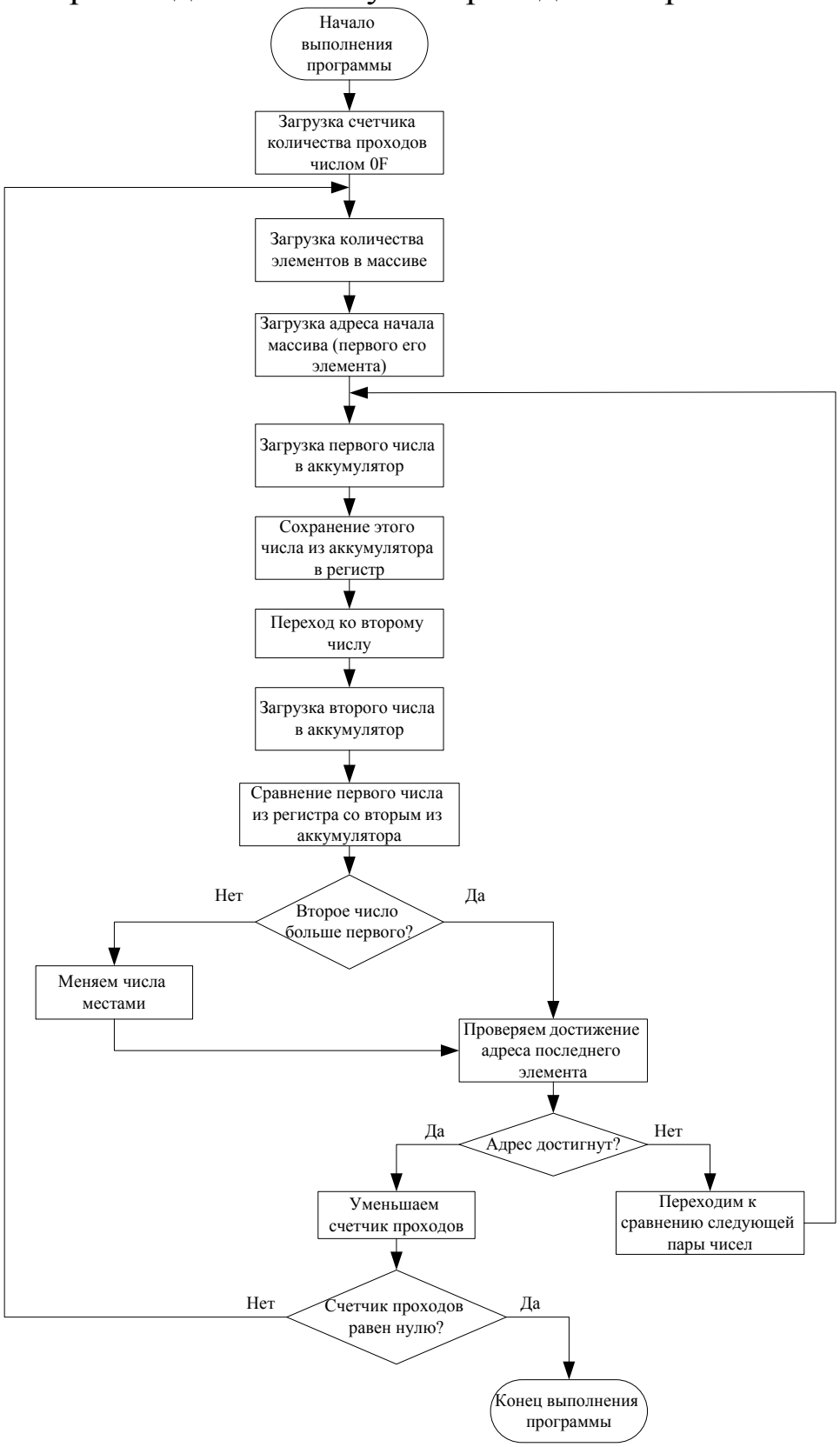
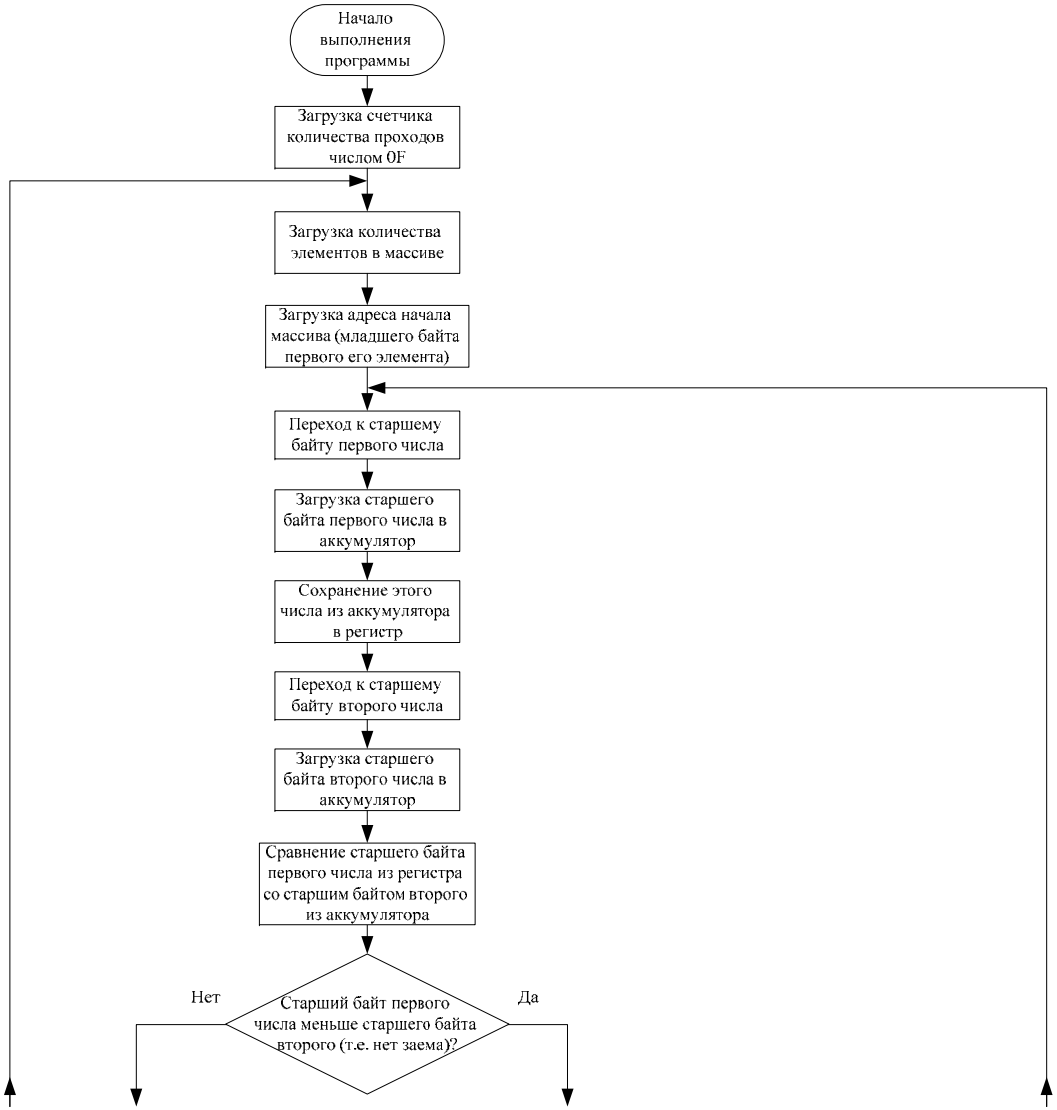



*Рис. 1.9. Сортировка для 15 однобайтовых чисел по возрастанию*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | mvi h, 0F | Загружаем счетчик количества проходов внешнего цикла |
| Label2: | | |
| | lda 0AA0 | Загружаем количество элементов массива |
| | mov l,a | |
| | dcr h | Уменьшаем счетчик количества проходов |
| | jz Label1 | Проверяем условие окончания сортировки |
| | lxi b, 0A50 | Загружаем начальный адрес массива (адрес первого элемента) |
| Label3: | | |
| | mov a, l | Проверяем достижение последнего элемента массива, если он достигнут – переходим на проверку счетчика количества проходов |
| | sui 01 | |
| | mov l,a | |
| | jz Label2 | |
| | ldax b | Загружаем первое число |
| | mov d, a | Сохраняем его в регистре D |
| | inr c | Загружаем второе (следующее) число |
| | ldax b | |
| | cmp d | Сравниваем его с первым числом из регистра D |
| | jnc Label3 | Если есть заем (т.е. первое число оказалось больше второго), то меняем числа местами, если нет заема – переходим к проверке достижения последнего элемента массива |
| | dcr c | |
| | stax b | |
| | inr c | |
| | mov a, d | |
| | stax b | |
| | jmp Label3 | Переходим к проверке достижения последнего элемента массива после обмена чисел |
| Label1: | | |
| | rst1 | |

Перепишем теперь эту программу для работы с двухбайтовыми числами. Здесь сложность будет заключаться в том, что в операции сравнения будут участвовать как старшие, так и младшие байты чисел. Для сравнения чисел будем пользоваться командой сравнения CMP, но сначала будем сравнивать старшие байты, а затем младшие. В случае если расположение старших байт удовлетворяет условию, то сравнивать младшие байты не имеет смысла и сразу переходим к проверке следующей пары. Аналогично, если расположение старших байт не удовлетворяет условию, то сравнивать младшие байты не имеет смысла и нужно переходить к обмену чисел местами. Исключением является случай, когда старшие байты чисел равны, в этом случае необходимо сравнить младшие байты. Блок-схема алгоритма приведена на рис. 1.10.


Начало выполнения программы
Загрузка счетчика количества проходов числом 0F
Загрузка количества элементов в массиве
Загрузка адреса начала массива (младшего байта первого его элемента)
Переход к старшему байту первого числа
Загрузка старшего байта первого числа в аккумулятор
Сохранение этого числа из аккумулятора в регистр
Переход к старшему байту второго числа
Загрузка старшего байта второго числа в аккумулятор
Сравнение старшего байта первого числа из регистра со старшим байтом второго из аккумулятора
Старший байт первого числа меньше старшего байта второго (т.е. нет заема)?
Нет
Да

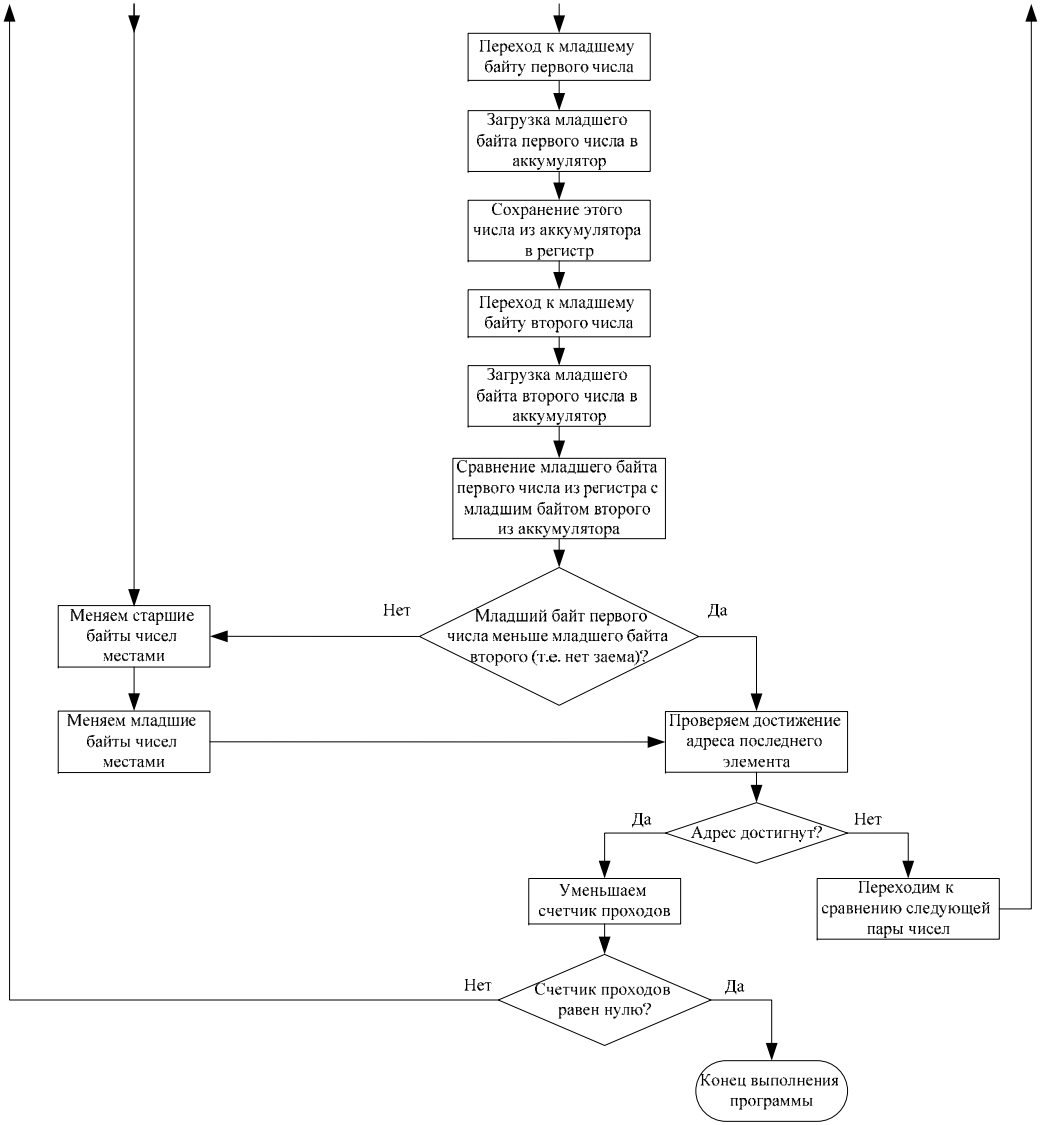



*Рис. 1.10. Сортировка для 15 двухбайтовых чисел по возрастанию*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | mvi h, 0f | Загружаем счетчик количества проходов внешнего цикла |
| Label2: | | |
| | lda 0AA0 | Загружаем количество элементов массива |
| | mov l,a | |
| | dcr h | Уменьшаем счетчик количества проходов |
| | jz Label1 | Проверяем условие окончания сортировки |
| | lxi b, 0A50 | Загружаем начальный адрес массива (адрес первого элемента) |
| Label4: | | |
| | inr c | Переходим к старшему байту первого числа |
| | mov a, l | Проверяем достижение последнего элемента массива, если он достигнут – переходим на проверку счетчика количества проходов |
| | sui 01 | |
| | mov l,a | |
| | jz Label2 | |
| | ldax b | Загружаем старший байт первого числа |
| | mov d, a | Сохраняем его в регистре D |
| | inr c | Переходим к старшему байту второго числа |
| | inr c | |
| | ldax b | Загружаем старший байт второго числа |
| | cmp d | Сравниваем его со старшим байтом первого числа из регистра D |
| | jnc Label3 | Если есть заем (т.е. старший байт первого числа оказался больше старшего байта второго), то меняем числа местами, если нет заема – переходим к сравнению младших байтов чисел |
| Label5: | | |
| | dcr c | Меняем местами **старшие** байты первого и второго числа |
| | dcr c | |
| | stax b | |
| | inr c | |
| | inr c | |
| | mov a, d | |
| | stax b | |
| | dcr c | Переходим к младшему байту первого числа |
| | dcr c | |
| | dcr c | |
| | ldax b | Загружаем младший байт первого числа |
| | mov d, a | Сохраняем его в регистре D |
| | inr c | Переходим к младшему байту второго числа |
| | inr c | |
| | ldax b | Загружаем младший байт второго числа |
| | dcr c | Меняем местами **младшие** байты первого и второго числа |
| | dcr c | |
| | stax b | |
| | inr c | |
| | inr c | |
| | mov a, d | |
| | stax b | |

*Продолжение таблицы*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | jmp Label4 | Переходим к проверке достижения последнего элемента массива после обмена чисел |
| Label3: | | |
| | dcr c | Переходим к младшему байту первого числа |
| | dcr c | |
| | dcr c | |
| | ldax b | Загружаем младший байт первого числа |
| | mov d, a | Сохраняем его в регистре D |
| | inr c | Переходим к младшему байту второго числа |
| | inr c | |
| | ldax b | Загружаем младший байт второго числа |
| | cmp d | Сравниваем его с младшим байтом первого числа из регистра D |
| | jnc Label4 | Если есть заем (т.е. младший байт первого числа оказался больше младшего байта второго), то меняем числа местами, если нет заема – переходим к проверке достижения последнего элемента массива |
| | dcr c | Переходим к старшему байту первого числа |
| | ldax b | Загружаем старший байт первого числа |
| | mov d, a | Сохраняем его в регистре D |
| | inr c | Переходим к старшему байту второго числа |
| | inr c | |
| | ldax b | Загружаем старший байт второго числа |
| | jmp Label5 | Переходим к обмену чисел местами на Label5 |
| Label1: | | |
| | rst1 | |

Эту же программу можно было бы написать и с использованием команд вычитания SUB и SBB. В данном случае будут вычитаться двухбайтовые числа – сначала с помощью команды SUB вычитаются младшие байты чисел, а затем с помощью команды SBB вычитаются старшие байты чисел с учетом флага заема/переноса (TC).

Для примера сравним числа $3716_{16} = 00110111\ 00010110_2$ и $6243_{16} = 01100010\ 01000011_2$. Здесь нам важна не сама разность, а значение флага заема Tc. Если он в результате вычитания равен единице, то уменьшаемое меньше вычитаемого; если он равен нулю – уменьшаемое больше вычитаемого.

Вычтем сначала младшие байты чисел:

0808) SUB D

| Адрес/регистр | Код до выполнения операции | Код после выполнения операции |
|---|---|---|
| 0808 | 92 | 92 |
| A | $16_{16}=00010110_2$ | $D3_{16}=11010011_2$ |
| D | $43_{16}=01000011_2$ | $43_{16}=01000011_2$ |
| FL | 02(Флаг Тс=0) | 83(Флаг Тс=1) |
| PC | 0808 | 0809 |

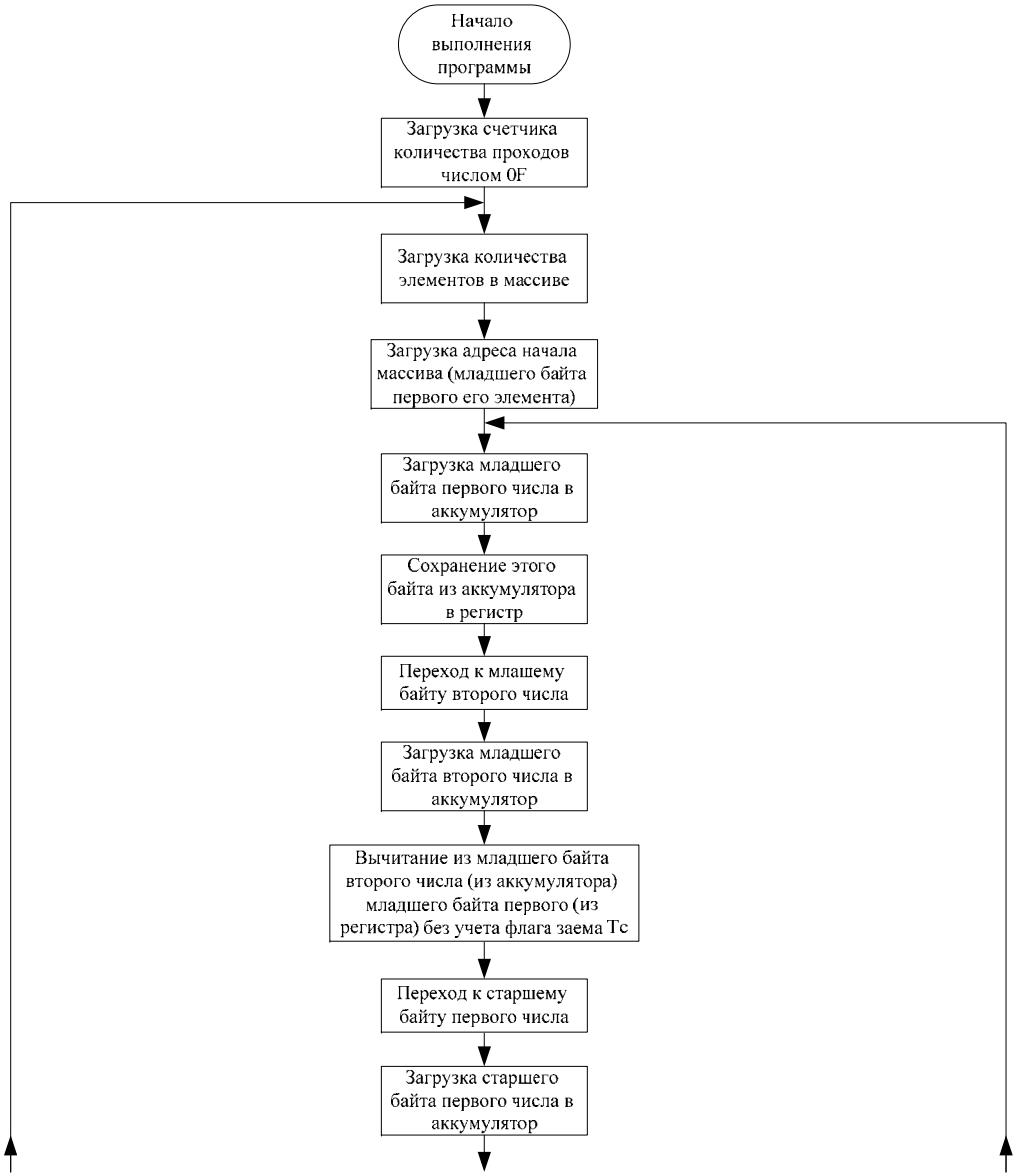
Стоит обратить внимание на активацию флага заема Тс=1 – эта единица должна быть учтена при вычитании старших байтов, что и происходит при использовании команды SBB:

0809) SBB D

| Адрес/регистр | Код до выполнения операции | Код после выполнения операции |
|---|---|---|
| 0809 | 9A | 9A |
| A | $37_{16}=00110111_2$ | $D4_{16}=11010100_2$ |
| D | $62_{16}=01100010_2$ | $62_{16}=01100010_2$ |
| FL | 83(Флаг Тс=1) | 87(Флаг Тс=1) |
| PC | 0809 | 080A |

В данном случае флаг Тс=1. Это говорит о том, что уменьшаемое меньше вычитаемого. Блок-схема алгоритма приведена на рис. 1.11.

Данную программу можно немного сократить в объеме. Если внимательно посмотреть на листинг, то можно заметить, что часть кода, отвечающая за обмен старших байтов местами, аналогична части кода, отвечающего за обмен младших байтов местами.


Начало выполнения программы
Загрузка счетчика количества проходов числом 0F
Загрузка количества элементов в массиве
Загрузка адреса начала массива (младшего байта первого его элемента)
Загрузка младшего байта первого числа в аккумулятор
Сохранение этого байта из аккумулятора в регистр
Переход к млашему байту второго числа
Загрузка младшего байта второго числа в аккумулятор
Вычитание из младшего байта второго числа (из аккумулятора) младшего байта первого (из регистра) без учета флага заема Tc
Переход к старшему байту первого числа
Загрузка старшего байта первого числа в аккумулятор

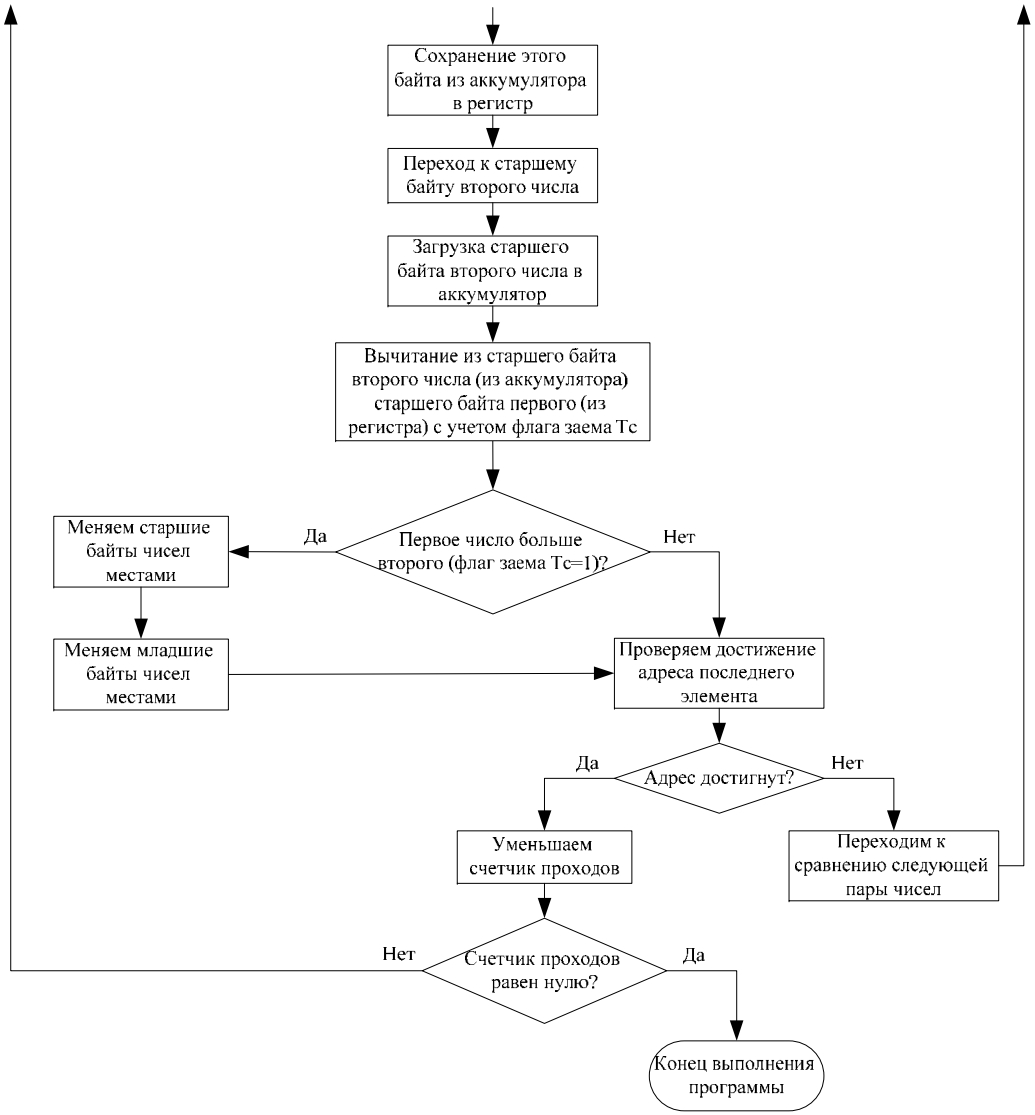



*Рис. 1.11. Сортировка для 15 двухбайтовых чисел по возрастанию*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | mvi h, 0F | Загружаем счетчик количества проходов внешнего цикла |
| Label2: | | |
| | lda 0AA0 | Загружаем количество элементов массива |
| | mov l,a | |
| | dcr h | Уменьшаем счетчик количества проходов |
| | jz Label1 | Проверяем условие окончания сортировки |
| | lxi b, 0A50 | Загружаем начальный адрес массива (адрес первого элемента) |
| Label3: | | |
| | mov a, l | Проверяем достижение последнего элемента массива, если он достигнут – переходим на проверку счетчика количества проходов |
| | sui 01 | |
| | mov l,a | |
| | jz Label2 | |
| | ldax b | Загружаем младший байт первого числа |
| | mov d, a | Сохраняем его в регистре D |
| | inx b | Переходим к младшему байту второго числа |
| | inx b | |
| | ldax b | Загружаем младший байт второго числа |
| | sub d | Вычитаем из него младший байт первого числа из регистра D |
| | dcx b | Переходим к старшему байту первого числа |
| | ldax b | Загружаем старший байт первого числа |
| | mov d, a | Сохраняем его в регистре D |
| | inx b | Переходим к старшему байту второго числа |
| | inx b | |
| | ldax b | Загружаем старший байт второго числа |
| | sbb d | Вычитаем из него старший байт первого числа из регистра D **с учетом флага заема (Tc)** |
| | dcx b | Переходим к младшему байту второго числа |
| | jnc Label3 | Если есть заем (т.е. первое число оказалось больше второго), то меняем числа местами, если нет заема – переходим к проверке достижения последнего элемента массива |
| | inx b | Переходим к старшему байту второго числа |
| | ldax b | Меняем местами **старшие** байты первого и второго числа |
| | dcx b | |
| | dcx b | |
| | stax b | |
| | inx b | |
| | inx b | |
| | mov a, d | |
| | stax b | |
| | dcx b | Переходим к младшему байту первого числа |
| | dcx b | |
| | dcx b | |
| | ldax b | Меняем местами **младшие** байты первого и второго числа |
| | mov d, a | |
| | inx b | |
| | inx b | |

*Продолжение таблицы*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | ldax b | Меняем местами **младшие** байты первого и второго числа |
| | dcx b | |
| | dcx b | |
| | stax b | |
| | inx b | |
| | inx b | |
| | mov a, d | |
| | stax b | |
| | jmp Label3 | Переходим к проверке достижения последнего элемента массива |
| Label1: | | |
| | rst1 | |

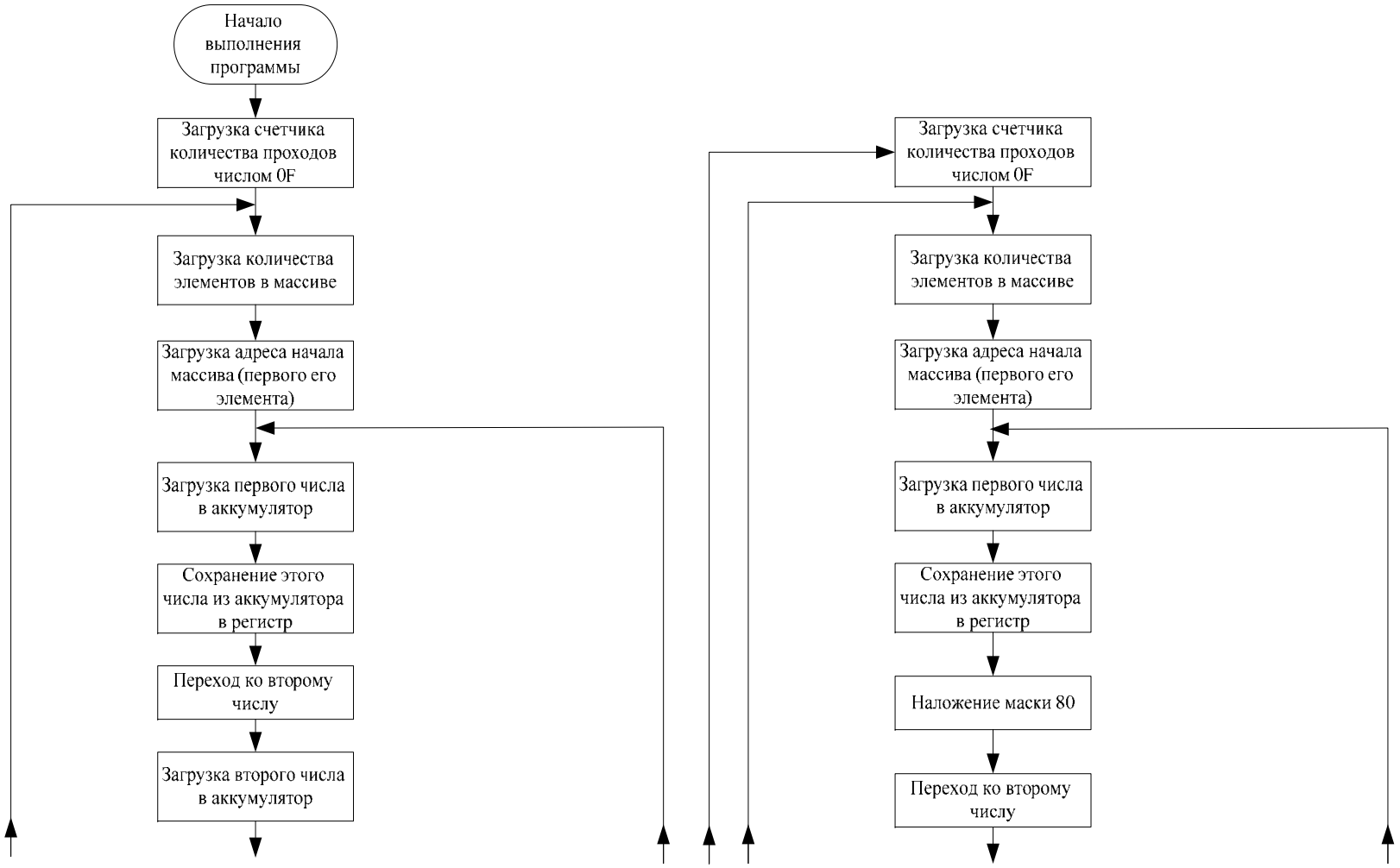
Вынесем повторяющийся кусок кода вынести в подпрограмму:

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | mvi h, 0F | Загружаем счетчик количества проходов внешнего цикла |
| Label2: | | |
| | lda 0AA0 | Загружаем количество элементов массива |
| | mov l,a | |
| | dcr h | Уменьшаем счетчик количества проходов |
| | jz Label1 | Проверяем условие окончания сортировки |
| | lxi b, 0A50 | Загружаем начальный адрес массива (адрес первого элемента) |
| Label3: | | |
| | mov a, l | Проверяем достижение последнего элемента массива, если он достигнут – переходим на проверку счетчика количества проходов |
| | sui 01 | |
| | mov l,a | |
| | jz Label2 | |
| | ldax b | Загружаем младший байт первого числа |
| | mov d, a | Сохраняем его в регистре D |
| | inx b | Переходим к младшему байту второго числа |
| | inx b | |
| | ldax b | Загружаем младший байт второго числа |
| | sub d | Вычитаем из него младший байт первого числа из регистра D |
| | dcx b | Переходим к старшему байту первого числа |
| | ldax b | Загружаем старший байт первого числа |
| | mov d, a | Сохраняем его в регистре D |
| | inx b | Переходим к старшему байту второго числа |
| | inx b | |
| | ldax b | Загружаем старший байт второго числа |
| | sbb d | Вычитаем из него старший байт первого числа из регистра D **с учетом флага заема (Tc)** |

*Продолжение таблицы*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | dcx b | Переходим к младшему байту второго числа |
| | jnc Label3 | Если есть заем (т.е. первое число оказалось больше второго), то меняем числа местами, если нет заема – переходим к проверке достижения последнего элемента массива |
| | call Label4 | Переходим на подпрограмму обмена для обмена местами **старших** байтов чисел |
| | dcx b | Переходим к младшему байту первого числа |
| | dcx b | |
| | dcx b | |
| | ldax b | Загружаем младший байт первого числа |
| | mov d, a | Сохраняем его в регистре D |
| | inx b | Переходим к младшему байту второго числа |
| | call Label4 | Переходим на подпрограмму обмена для обмена местами **младших** байтов чисел |
| | jmp Label3 | Переходим к проверке достижения последнего элемента массива |
| Label1: | | |
| | rst1 | |
| Label4: | | Подпрограмма обмена чисел местами |
| | inx b | |
| | ldax b | |
| | dcx b | |
| | dcx b | |
| | stax b | |
| | inx b | |
| | inx b | |
| | mov a, d | |
| | stax b | |
| | ret | |

При необходимости сортировки чисел со знаком следует дописать программу сортировки таким образом, чтобы после сортировки всего массива отрицательные и положительные числа менялись местами. Приведем пример для сортировки по убыванию 15 однобайтовых чисел (рис. 1.12).


Начало выполнения программы
Загрузка счетчика количества проходов числом 0F
Загрузка количества элементов в массиве
Загрузка адреса начала массива (первого его элемента)
Загрузка первого числа в аккумулятор
Сохранение этого числа из аккумулятора в регистр
Переход ко второму числу
Загрузка второго числа в аккумулятор
Загрузка счетчика количества проходов числом 0F
Загрузка количества элементов в массиве
Загрузка адреса начала массива (первого его элемента)
Загрузка первого числа в аккумулятор
Сохранение этого числа из аккумулятора в регистр
Наложение маски 80
Переход ко второму числу

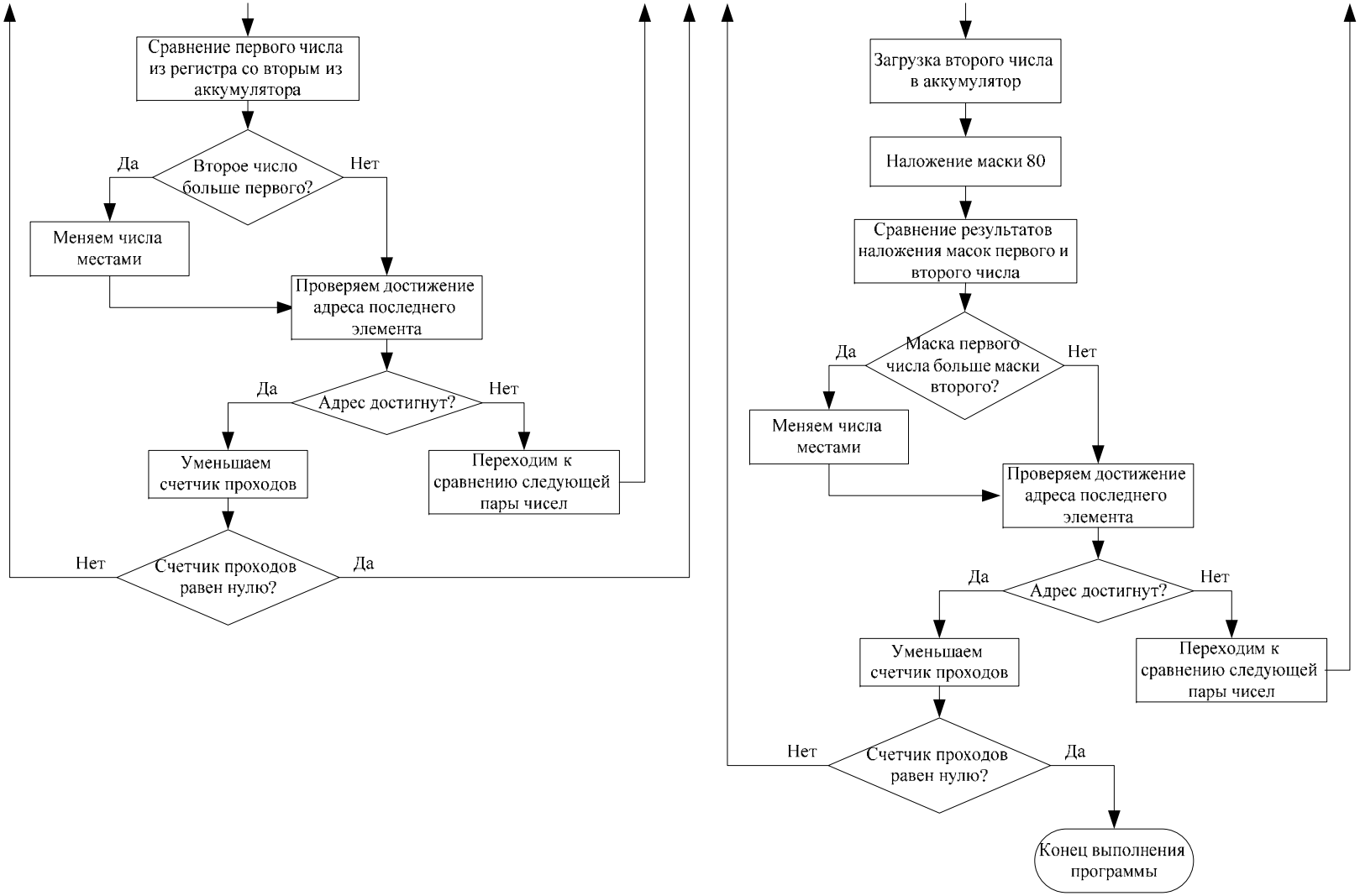



*Рис. 1.12. Сортировка для 15 двухбайтовых чисел по возрастанию*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | mvi h, 0F | Загружаем счетчик количества проходов внешнего цикла |
| Label2: | | |
| | lda 0AA0 | Загружаем количество элементов массива |
| | mov l,a | |
| | dcr h | Уменьшаем счетчик количества проходов |
| | jz Label1 | Проверяем условие окончания сортировки |
| | lxi b, 0A50 | Загружаем начальный адрес массива (адрес первого элемента) |
| Label3: | | |
| | mov a, l | Проверяем достижение последнего элемента массива, если он достигнут – переходим на проверку счетчика количества проходов |
| | sui 01 | |
| | mov l,a | |
| | jz Label2 | |
| | ldax b | Загружаем первое число |
| | mov d, a | Сохраняем его в регистре D |
| | inr c | Загружаем второе (следующее) число |
| | ldax b | |
| | cmp d | Сравниваем его с первым числом из регистра D |
| | jc Label3 | Если нет заема (т.е. первое число оказалось меньше второго), то меняем числа местами, если есть заем – переходим к проверке достижения последнего элемента массива |
| | dcr c | |
| | stax b | |
| | inr c | |
| | mov a, d | |
| | stax b | |
| | jmp Label3 | Переходим к проверке достижения последнего элемента массива после обмена чисел |
| Label1: | | Обмен местами положительных и отрицательных чисел |
| | mvi h, 0F | Загружаем счетчик количества проходов внешнего цикла |
| Label5: | | |
| | lda 0AA0 | Загружаем количество элементов массива |
| | mov l, a | |
| | dcr h | Уменьшаем счетчик количества проходов |
| | jz Label4 | Проверяем условие окончания сортировки |
| | lxi b, 0A50 | Загружаем начальный адрес массива (адрес первого элемента) |
| Label6: | | |
| | mov a, l | Проверяем достижение последнего элемента массива, если он достигнут – переходим на проверку счетчика количества проходов |
| | sui 01 | |
| | mov l, a | |
| | jz Label5 | |
| | ldax b | Загружаем первое число |
| | mov e, a | Сохраняем его в регистре E |
| | ani 80 | Накладываем маску для проверки числа на отрицательность |
| | mov d, a | Сохраняем результат в регистре D |
| | inr c | Загружаем второе (следующее) число |
| | ldax b | |
| | ani 80 | Накладываем маску для проверки числа на отрицательность |
| | cmp d | Сравниваем результат с числом из регистра D |

*Продолжение таблицы*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | jnc Label6 | Если нет заема (т.е. первое число оказалось отрицательным, а второе положительным), то меняем числа местами, если есть заем – переходим к проверке достижения последнего элемента массива |
| | ldax b | |
| | dcr c | |
| | stax b | |
| | inr c | |
| | mov a, e | |
| | stax b | |
| | jmp Label6 | Переходим к проверке достижения последнего элемента массива после обмена чисел |
| Label4: | | |
| | rst1 | |

## 1.4. Возврат к знаковым значениям чисел

Для перехода к знаковым значениям чисел достаточно поправить программу сортировки, написанную в предыдущем разделе, таким образом, чтобы вместе с модулями чисел одновременно сортировались и их знаковые значения, расположенные начиная с адреса 0A30.

Для наглядности выполним эту операцию с однобайтовыми числами. В регистровой паре BC будут храниться адреса модулей чисел (начиная с адреса 0A50), в паре DE будут храниться адреса их знаковых значений (начиная с адреса 0A30), а счетчик количества проходов будем сохранять по адресу 0A70. Блок-схема алгоритма представлена рис. 1.13.

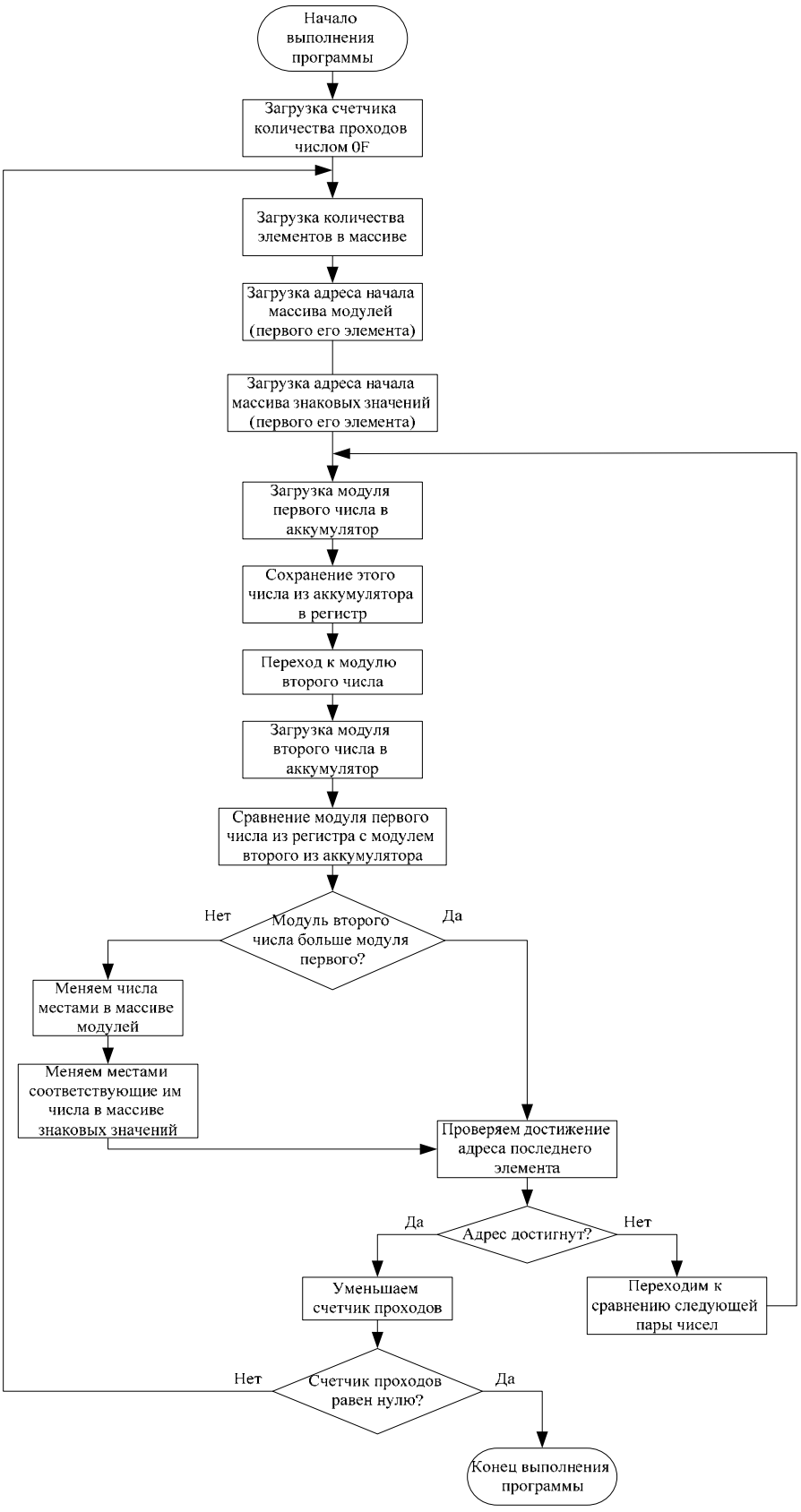
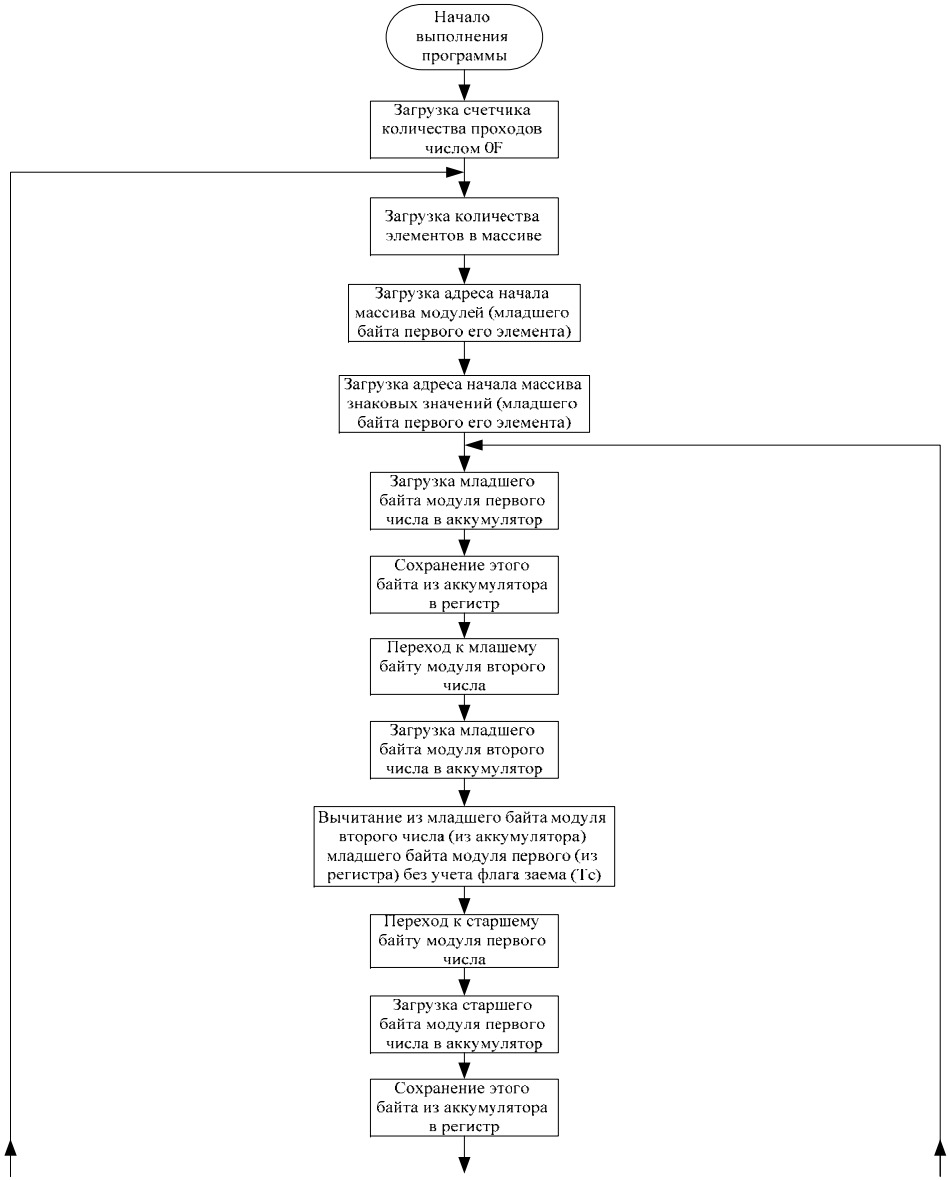



*Рис. 1.13. Сортировка 15 однобайтовых чисел по возрастанию с учетом модуля*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | mvi a, 0F | Загружаем счетчик количества проходов внешнего цикла |
| | sta 0A70 | |
| Label2: | | |
| | lda 0AA0 | Загружаем количество элементов массива |
| | mov l,a | |
| | lda 0A70 | Уменьшаем счетчик количества проходов |
| | sui 01 | |
| | sta 0A70 | |
| | jz Label1 | Проверяем условие окончания сортировки |
| | lxi b, 0A50 | Загружаем начальный адрес массива модулей чисел (адрес первого элемента) |
| | lxi d, 0A30 | Загружаем начальный адрес массива знаковых значений чисел (адрес первого элемента) |
| Label3: | | |
| | mov a, l | Проверяем достижение последнего элемента массива, если он достигнут – переходим на проверку счетчика количества проходов |
| | sui 01 | |
| | mov l,a | |
| | jz Label2 | |
| | ldax b | Загружаем модуль первого числа |
| | mov h, a | Сохраняем его в регистре H |
| | inr c | Переходим к модулю второго (следующего) числа |
| | ldax b | Загружаем модуль второго числа |
| | cmp h | Сравниваем его с модулем из регистра H |
| | inr e | Переходим к следующему числу в массиве знаковых значений чисел |
| | jnc Label3 | Если есть заем (т.е. первое число оказалось больше второго), то меняем числа местами, если нет заема – переходим к проверке достижения последнего элемента массива |
| | dcr c | Меняем местами числа в массиве модулей |
| | stax b | |
| | inr c | |
| | mov a, h | |
| | stax b | |
| | dcr e | Меняем местами числа в массиве знаковых значений, соответствующие числам в массиве модулей |
| | ldax d | |
| | mov h, a | |
| | inr e | |
| | ldax d | |
| | dcr e | |
| | stax d | |
| | inr e | |
| | mov a, h | |
| | stax d | |
| | jmp Label3 | Переходим к проверке достижения последнего элемента массива после обмена чисел |
| Label1: | | |
| | rst1 | |

Теперь перейдем к двухбайтовым числам (рис. 1.14).


Начало выполнения программы
Загрузка счетчика количества проходов числом 0F
Загрузка количества элементов в массиве
Загрузка адреса начала массива модулей (младшего байта первого его элемента)
Загрузка адреса начала массива знаковых значений (младшего байта первого его элемента)
Загрузка младшего байта модуля первого числа в аккумулятор
Сохранение этого байта из аккумулятора в регистр
Переход к млашему байту модуля второго числа
Загрузка младшего байта модуля второго числа в аккумулятор
Вычитание из младшего байта модуля второго числа (из аккумулятора) младшего байта модуля первого (из регистра) без учета флага заема (Tc)
Переход к старшему байту модуля первого числа
Загрузка старшего байта модуля первого числа в аккумулятор
Сохранение этого байта из аккумулятора в регистр

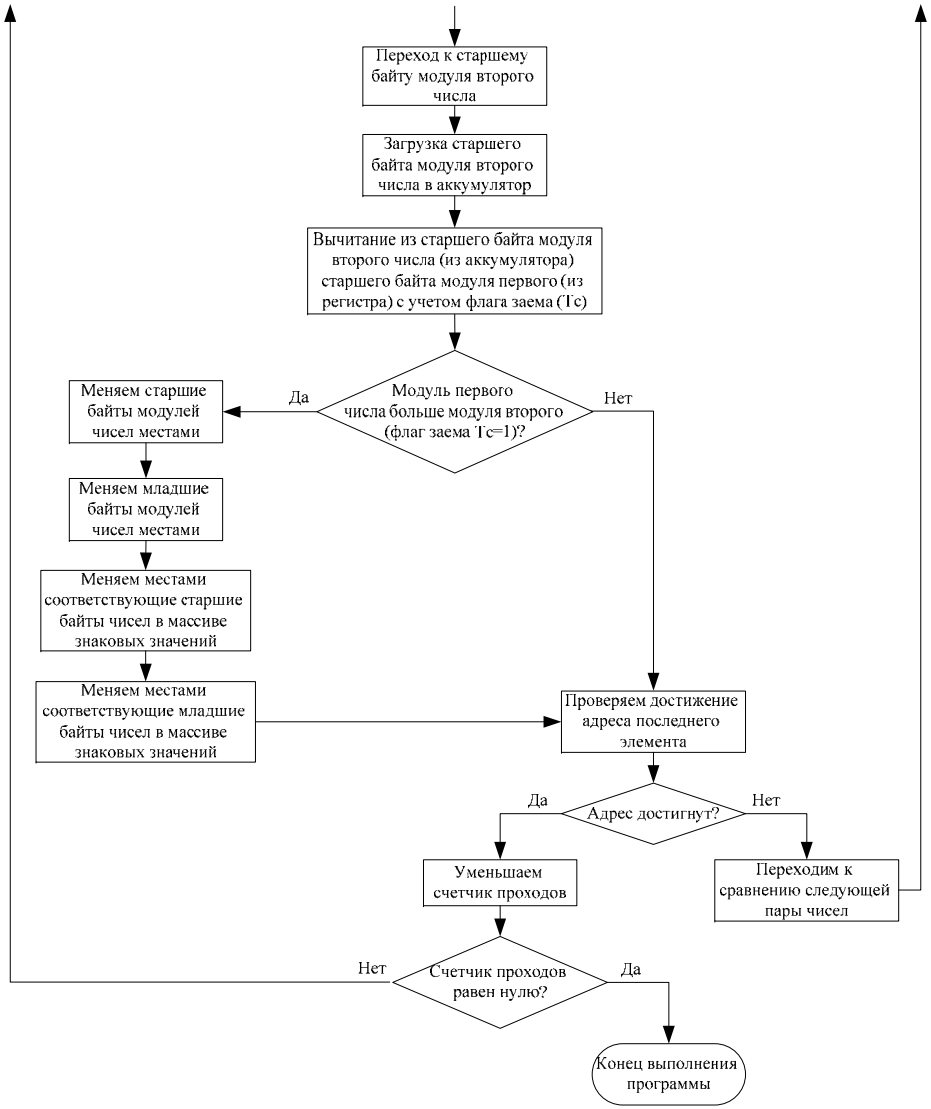



*Рис. 1.14. Сортировка 15 двухбайтовых чисел по возрастанию с учетом модуля*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | mvi a, 0F | Загружаем счетчик количества проходов внешнего цикла |
| | sta 0A70 | |
| Label2: | | |
| | lda 0AA0 | Загружаем количество элементов массива |
| | mov l,a | |
| | lda 0A70 | Уменьшаем счетчик количества проходов |
| | sui 01 | |
| | sta 0A70 | |
| | jz Label1 | Проверяем условие окончания сортировки |
| | lxi b, 0A50 | Загружаем начальный адрес массива модулей чисел (адрес первого элемента) |
| | lxi d, 0A30 | Загружаем начальный адрес массива знаковых значений чисел (адрес первого элемента) |
| Label3: | | |
| | mov a, l | Проверяем достижение последнего элемента массива, если он достигнут – переходим на проверку счетчика количества проходов |
| | sui 01 | |
| | mov l,a | |
| | jz Label2 | |
| | ldax b | Загружаем младший байт модуля первого числа |
| | mov h, a | Сохраняем его в регистре D |
| | inx b | Переходим к младшему байту модуля второго числа |
| | inx b | |
| | ldax b | Загружаем младший байт модуля второго числа |
| | sub h | Вычитаем из него младший байт модуля первого числа из регистра H |
| | dcx b | Переходим к старшему байту модуля первого числа |
| | ldax b | Загружаем старший байт модуля первого числа |
| | mov h, a | Сохраняем его в регистре H |
| | inx b | Переходим к старшему байту модуля второго числа |
| | inx b | |
| | ldax b | Загружаем старший байт модуля второго числа |
| | sbb h | Вычитаем из него старший байт модуля первого числа из регистра D **с учетом флага заема (Tc)** |
| | dcx b | Переходим к младшему байту модуля второго числа |
| | inx d | Переходим к младшему байту знакового значения второго числа |
| | inx d | |
| | jnc Label3 | Если есть заем (т.е. первое число оказалось больше второго), то меняем числа местами, если нет заема – переходим к проверке достижения последнего элемента массива |
| | inx b | Переходим к **старшему** байту модуля второго числа |
| | ldax b | Меняем местами **старшие** байты модулей первого и второго чисел |
| | dcx b | |
| | dcx b | |
| | stax b | |
| | inx b | |
| | inx b | |
| | mov a, h | |
| | stax b | |
| | dcx b | Переходим к младшему байту модуля первого числа |
| | dcx b | |
| | dcx b | |
| | ldax b | Меняем местами **младшие** байты модулей первого и второго чисел |
| | mov h, a | |
| | inx b | |
| | inx b | |
| | ldax b | |
| | dcx b | |

*Продолжение таблицы*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | dcx b | |
| | stax b | |
| | inx b | |
| | inx b | |
| | mov a, h | |
| | stax b | |
| | dcx d | Переходим к старшему байту знакового значения первого числа |
| | ldax d | Меняем местами **старшие** байты знаковых значений первого и второго чисел, соответствующие **старшим** байтам модулей первого и второго чисел |
| | mov h, a | |
| | inx d | |
| | inx d | |
| | ldax d | |
| | dcx d | |
| | dcx d | |
| | stax d | |
| | inx d | |
| | inx d | |
| | mov a, h | |
| | stax d | |
| | dcx d | Переходим к младшему байту знакового значения первого числа |
| | dcx d | |
| | dcx d | |
| | ldax d | Меняем местами **младшие** байты знаковых значений первого и второго чисел, соответствующие **младшим** байтам модулей первого и второго чисел |
| | mov h, a | |
| | inx d | |
| | inx d | |
| | ldax d | |
| | dcx d | |
| | dcx d | |
| | stax d | |
| | inx d | |
| | inx d | |
| | mov a, h | |
| | stax d | |
| | jmp Label3 | Переходим к проверке достижения последнего элемента массива |
| Label1: | | |
| | rst1 | |

Этот листинг можно также сократить, вынеся в подпрограммы часть кода, отвечающую за обмены байт в массиве модулей чисел, и часть кода, отвечающую за обмены байт в массиве знаковых значений чисел.

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | mvi a, 0F | Загружаем счетчик количества проходов внешнего цикла |
| | sta 0A70 | |
| Label2: | | |
| | lda 0AA0 | Загружаем количество элементов массива |
| | mov l,a | |
| | lda 0A70 | Уменьшаем счетчик количества проходов |
| | sui 01 | |
| | sta 0A70 | |
| | jz Label1 | Проверяем условие окончания сортировки |

*Продолжение таблицы*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| Label2: | | |
| | lxi b, 0A50 | Загружаем начальный адрес массива модулей чисел (адрес первого элемента) |
| | lxi d, 0A30 | Загружаем начальный адрес массива знаковых значений чисел (адрес первого элемента) |
| Label3: | | |
| | mov a, l | Проверяем достижение последнего элемента массива, если он достигнут – переходим на проверку счетчика количества проходов |
| | sui 01 | |
| | mov l,a | |
| | jz Label2 | |
| | ldax b | Загружаем младший байт модуля первого числа |
| | mov h, a | Сохраняем его в регистре D |
| | inx b | Переходим к младшему байту модуля второго числа |
| | inx b | |
| | ldax b | Загружаем младший байт модуля второго числа |
| | sub h | Вычитаем из него младший байт модуля первого числа из регистра H |
| | dcx b | Переходим к старшему байту модуля первого числа |
| | ldax b | Загружаем старший байт модуля первого числа |
| | mov h, a | Сохраняем его в регистре H |
| | inx b | Переходим к старшему байту модуля второго числа |
| | inx b | |
| | ldax b | Загружаем старший байт модуля второго числа |
| | sbb h | Вычитаем из него старший байт модуля первого числа из регистра D **с учетом флага заема (Tc)** |
| | dcx b | Переходим к младшему байту модуля второго числа |
| | inx d | Переходим к младшему байту знакового значения второго числа |
| | inx d | |
| | jnc Label3 | Если есть заем (т.е. первое число оказалось больше второго), то меняем числа местами, если нет заема – переходим к проверке достижения последнего элемента массива |
| | call Label4 | Переходим на подпрограмму обмена для обмена местами **старших** байтов модулей чисел |
| | dcx b | Переходим к младшему байту модуля первого числа |
| | dcx b | |
| | dcx b | |
| | ldax b | Загружаем младший байт модуля первого числа |
| | mov h, a | Сохраняем его в регистре D |
| | inx b | Переходим к младшему байту модуля второго числа |
| | call Label4 | Переходим на подпрограмму обмена для обмена местами **младших** байтов модулей чисел |
| | dcx d | Переходим к старшему байту знакового значения первого числа |
| | call Label5 | Переходим на подпрограмму обмена для обмена местами **старших** байтов знаковых значений чисел, соответствующих **старшим** байтам модулей первого и второго чисел |
| | dcx d | Переходим к младшему байту знакового значения первого числа |
| | dcx d | |
| | dcx d | |
| | call Label5 | Переходим на подпрограмму обмена для обмена местами **младших** байтов знаковых значений чисел, соответствующих **младшим** байтам модулей первого и второго чисел |
| | jmp Label3 | Переходим к проверке достижения последнего элемента массива |
| Label1: | | |
| | rst1 | |

*Продолжение таблицы*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| Label4: | | Подпрограмма обмена модулей чисел местами |
| | inx b | |
| | ldax b | |
| | dcx b | |
| | dcx b | |
| | stax b | |
| | inx b | |
| | inx b | |
| | mov a, h | |
| | stax b | |
| | ret | |
| Label5: | | Подпрограмма обмена знаковых значений чисел местами |
| | ldax d | |
| | mov h, a | |
| | inx d | |
| | inx d | |
| | ldax d | |
| | dcx d | |
| | dcx d | |
| | stax d | |
| | inx d | |
| | inx d | |
| | mov a, h | |
| | stax d | |
| | ret | |

Таким образом, мы выполнили первую часть задания на курсовое проектирование, связанную с сортировкой чисел в массиве.

# 2. НАХОЖДЕНИЕ СРЕДНЕГО АРИФМЕТИЧЕСКОГО ВСЕХ ЧИСЕЛ

Среднее арифметическое ряда чисел – это сумма всех чисел ряда, деленная на количество этих чисел в ряду:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{1}{n}(x_1 + x_2 + \cdots + x_n).$$

## 2.1. Нахождение суммы всех чисел

Поскольку по заданию требуется вычислить среднее арифметическое самих чисел, а не их модулей, то необходимо учитывать их знак в процессе сложения. Сам результат, соответственно, тоже должен быть знаковым. Так как при сложении 15 однобайтовых (двухбайтовых) чисел максимально возможный результат будет двухбайтовым (трехбайтовым), то чтобы не потерять знак в процессе суммирования, необходимо произвести расширение знака чисел – увеличить размер числа до нужного количества байт, заполняя добавленные байты битами, равными старшему биту исходного числа. Например, расширим знак числа $B4_{16} = 10110100_2$ и числа $6C_{16} = 01101100_2$ до второго байта:

$$FFB4_{16} = \overbrace{11111111}\,\textcircled{1}0110100_2$$

$$006C_{16} = \overbrace{00000000}\,\textcircled{0}1101100_2$$

Числа после выполнения процедуры расширения знака будем записывать в отдельный массив, начиная с адреса 0B00. После расширения знака можно выполнять операцию сложения чисел (теперь уже двухбайтовых, либо трехбайтовых и т.д.). Так как складывать необходимо как минимум двухбайтовые числа, то младшие байты будем суммировать с помощью команды сложения ADD, а старшие – с помощью команды ADC, чтобы учесть значение флага переноса Tc. Для примера сложим числа, над которыми уже выполнена операция расширения знака:

число $005D_{16} = 00000000\ 01011101_2$ сложим с числом $FFB2_{16} = 11111111\ 10110010_2$

$$\begin{array}{l} \overset{\curvearrowleft}{1}\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ \overset{\curvearrowleft}{1}\ 1\ 1\ 1\ 1 \\ 00000000\ 01011101_2 \\ +\ 11111111\ 10110010_2 \\ \hline 00000000\ 00001111_2 \end{array} \quad \rightarrow \quad 00000000\ 00001111_2 = 000F_{16},\ Tc=1$$

Сложим сначала младшие байты чисел.

080A) ADD H

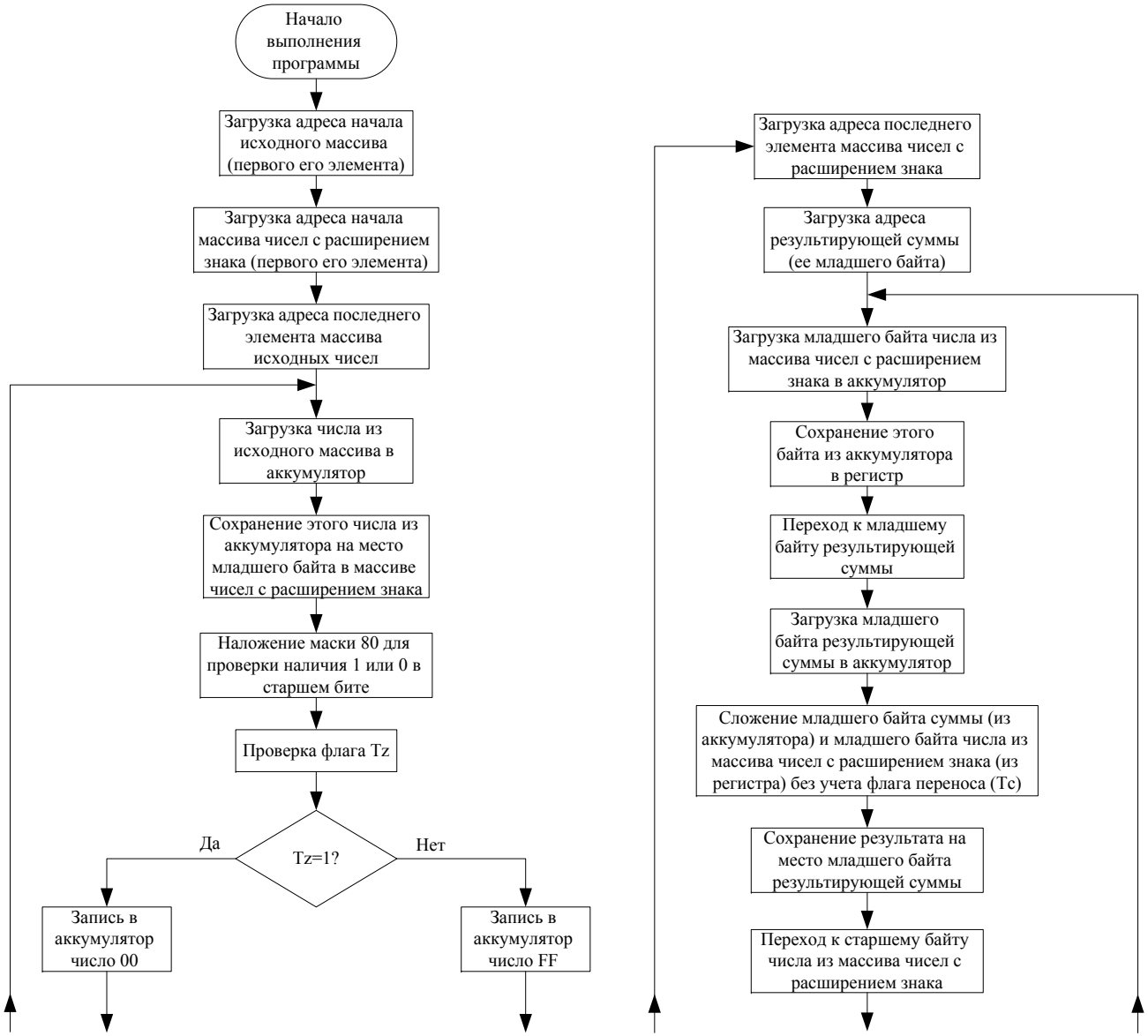
| Адрес/регистр | Код до выполнения операции | Код после выполнения операции |
|---|---|---|
| 080A | 84 | 84 |
| A | $5D_{16}=01011101_2$ | $0F_{16}=00001111_2$ |
| H | $B2_{16}=10110010_2$ | $A2_{16}=10110010_2$ |
| FL | 02(Флаг Tc=0) | 07(Флаг Tc=1) |
| PC | 080A | 080B |

Значение флага переноса Tc должно быть учтено при сложении старших байтов, что и позволяет выполнить команда ADC.

080B) ADC H

| Адрес/регистр | Код до выполнения операции | Код после выполнения операции |
|---|---|---|
| 0809 | 8C | 8C |
| A | $00_{16}=00000000_2$ | $00_{16}=00000000_2$ |
| H | $FF_{16}=11111111_2$ | $FF_{16}=11111111_2$ |
| FL | 07(Флаг Tc=1) | 57(Флаг Tc=1) |
| PC | 080B | 080C |

Здесь уже значение возникшего переноса учитывать не нужно, т.к. изначально мы работаем с однобайтовыми числами, над которыми выполнена операция расширения знака и необходимый нам перенос уже учтен в старшем байте суммы.


Начало выполнения программы
Загрузка адреса начала исходного массива (первого его элемента)
Загрузка адреса начала массива чисел с расширением знака (первого его элемента)
Загрузка адреса последнего элемента массива исходных чисел
Загрузка числа из исходного массива в аккумулятор
Сохранение этого числа из аккумулятора на место младшего байта в массиве чисел с расширением знака
Наложение маски 80 для проверки наличия 1 или 0 в старшем бите
Проверка флага Tz
Tz=1?
Да
Нет
Запись в аккумулятор число 00
Запись в аккумулятор число FF
Загрузка адреса последнего элемента массива чисел с расширением знака
Загрузка адреса результирующей суммы (ее младшего байта)
Загрузка младшего байта числа из массива чисел с расширением знака в аккумулятор
Сохранение этого байта из аккумулятора в регистр
Переход к младшему байту результирующей суммы
Загрузка младшего байта результирующей суммы в аккумулятор
Сложение младшего байта суммы (из аккумулятора) и младшего байта числа из массива чисел с расширением знака (из регистра) без учета флага переноса (Tc)
Сохранение результата на место младшего байта результирующей суммы
Переход к старшему байту числа из массива чисел с расширением знака

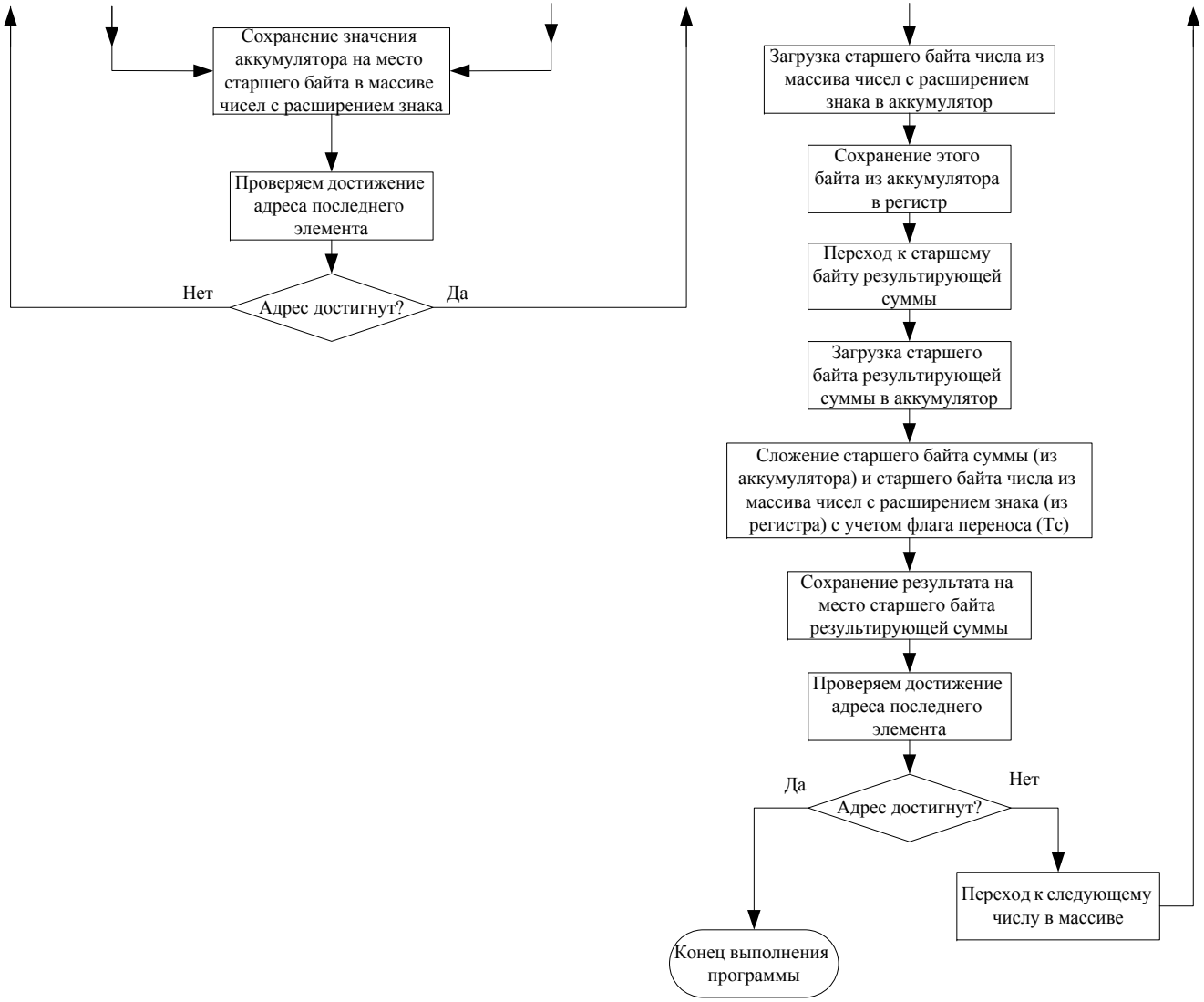
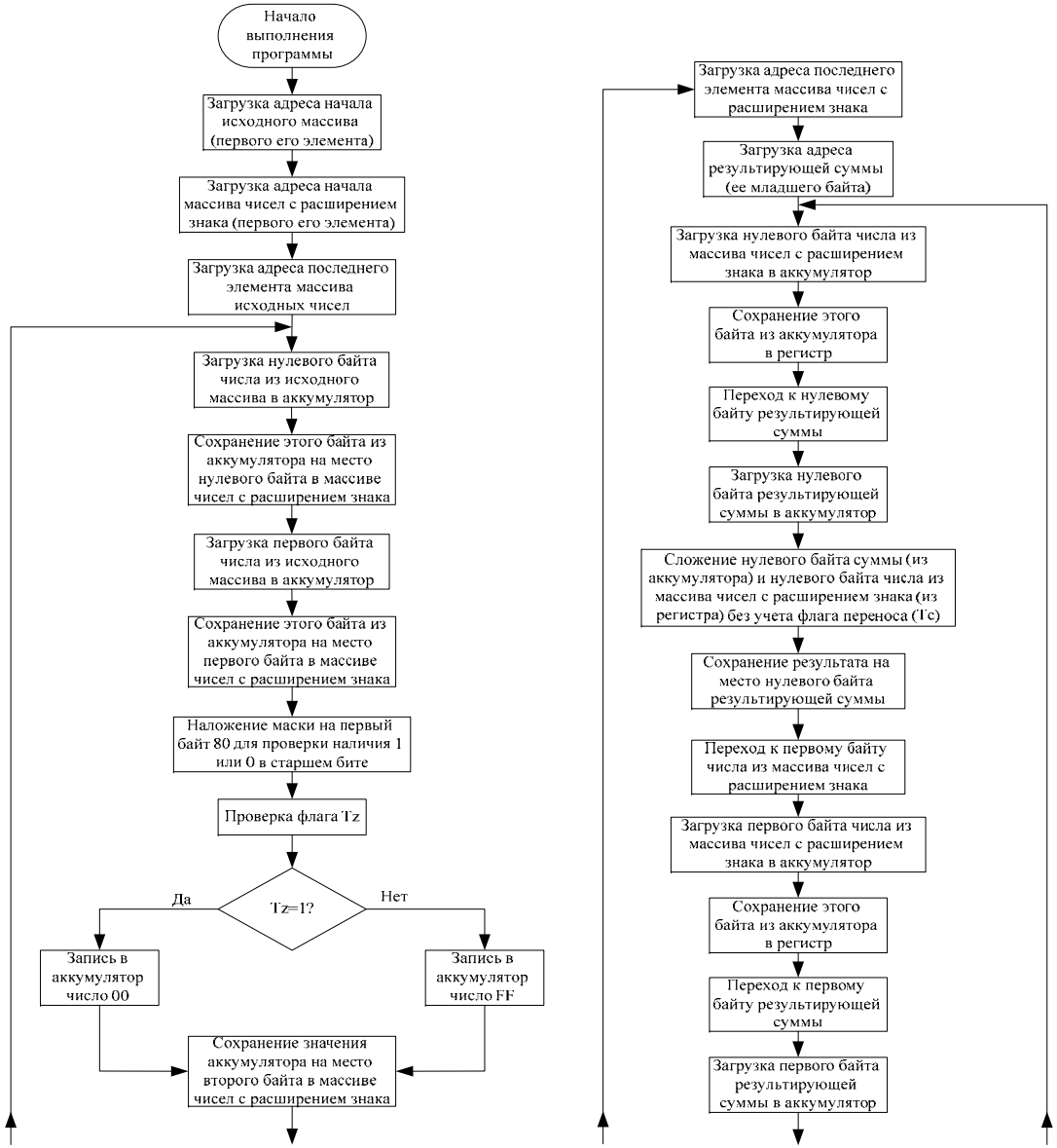



*Рис. 2.1. Выполнение операции расширения знака для массива 15 однобайтовых чисел и нахождение суммы чисел массива*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | lxi b, 0A00 | Загружаем начальный адрес исходного массива знаковых значений чисел (адрес первого элемента) |
| | lxi d, 0B00 | Загружаем начальный адрес массива знаковых значений чисел с расширением знака (адрес первого элемента) |
| Label2: | | |
| | ldax b | Загружаем число из исходного массива |
| | stax d | Сохраняем это число на место младшего байта в массиве чисел с расширением знака |
| | inx d | Переходим к старшему байту в массиве чисел с расширением знаков |
| | mov h, a | Проверяем исходное число на знак (на наличие 1 в старшем бите) |
| | ani 80 | |
| | jz Label1 | Заполняем старший байт числа в массиве чисел с расширением знака единицами, если число оказалось отрицательным и нулями – если положительным. |
| | mvi a, FF | |
| Label1: | | |
| | stax d | |
| | inx b | Переходим на следующий адрес в массиве исходных чисел |
| | inx d | Переходим на следующий адрес в массиве чисел с расширением знаков |
| | mvi a, 0F | Проверяем достижение последнего числа в массиве исходных чисел |
| | sub c | |
| | jnz Label2 | |
| | | Сложение чисел |
| | lxi b, 0B00 | Загружаем начальный адрес массива знаковых значений чисел с расширением знака (адрес первого элемента) |
| Label3: | | |
| | lxi d, 0AF0 | Загружаем адрес результирующей суммы (куда будем записывать результат сложения) |
| | ldax b | Загружаем **младший** байт числа |
| | mov h, a | Сохраняем его в регистре H |
| | ldax d | Загружаем **младший** байт суммы из адреса 0AF0 |
| | add h | Складываем младшие байты числа из регистра H и суммы из аккумулятора без учета флага переноса Tc |
| | stax d | Сохраняем результат по адресу 0AF0 |
| | inx b | Переходим к **старшему** байту числа |
| | inx d | Переходим к **старшему** байту суммы |
| | ldax b | Загружаем **старший** байт числа |
| | mov h, a | Сохраняем его в регистре H |
| | ldax d | Загружаем **старший** байт суммы из адреса 0AF1 |
| | adc h | Складываем старшие байты числа из регистра H и суммы из аккумулятора с учетом флага переноса Tc |
| | stax d | Сохраняем результат по адресу 0AF1 |
| | inx b | Переходим к младшему байту следующего числа |
| | mvi a, 1E | Проверяем, просуммированы ли все числа |
| | sub c | |
| | jnz Label3 | |
| | rst1 | |

Поправим теперь этот листинг для работы с двухбайтовыми числами (рис. 2.2). Основным отличием будет расширение знака двухбайтовых чисел до трехбайтовых. В операции суммирования команда ADC будет использована дважды, чтобы учесть перенос не только между нулевым и первым байтами, но и между первым и вторым.


Начало выполнения программы
Загрузка адреса начала исходного массива (первого его элемента)
Загрузка адреса начала массива чисел с расширением знака (первого его элемента)
Загрузка адреса последнего элемента массива исходных чисел
Загрузка нулевого байта числа из исходного массива в аккумулятор
Сохранение этого байта из аккумулятора на место нулевого байта в массиве чисел с расширением знака
Загрузка первого байта числа из исходного массива в аккумулятор
Сохранение этого байта из аккумулятора на место первого байта в массиве чисел с расширением знака
Наложение маски на первый байт 80 для проверки наличия 1 или 0 в старшем бите
Проверка флага Tz
Tz=1?
Да
Нет
Запись в аккумулятор число 00
Запись в аккумулятор число FF
Сохранение значения аккумулятора на место второго байта в массиве чисел с расширением знака
Загрузка адреса последнего элемента массива чисел с расширением знака
Загрузка адреса результирующей суммы (ее младшего байта)
Загрузка нулевого байта числа из массива чисел с расширением знака в аккумулятор
Сохранение этого байта из аккумулятора в регистр
Переход к нулевому байту результирующей суммы
Загрузка нулевого байта результирующей суммы в аккумулятор
Сложение нулевого байта суммы (из аккумулятора) и нулевого байта числа из массива чисел с расширением знака (из регистра) без учета флага переноса (Tc)
Сохранение результата на место нулевого байта результирующей суммы
Переход к первому байту числа из массива чисел с расширением знака
Загрузка первого байта числа из массива чисел с расширением знака в аккумулятор
Сохранение этого байта из аккумулятора в регистр
Переход к первому байту результирующей суммы
Загрузка первого байта результирующей суммы в аккумулятор

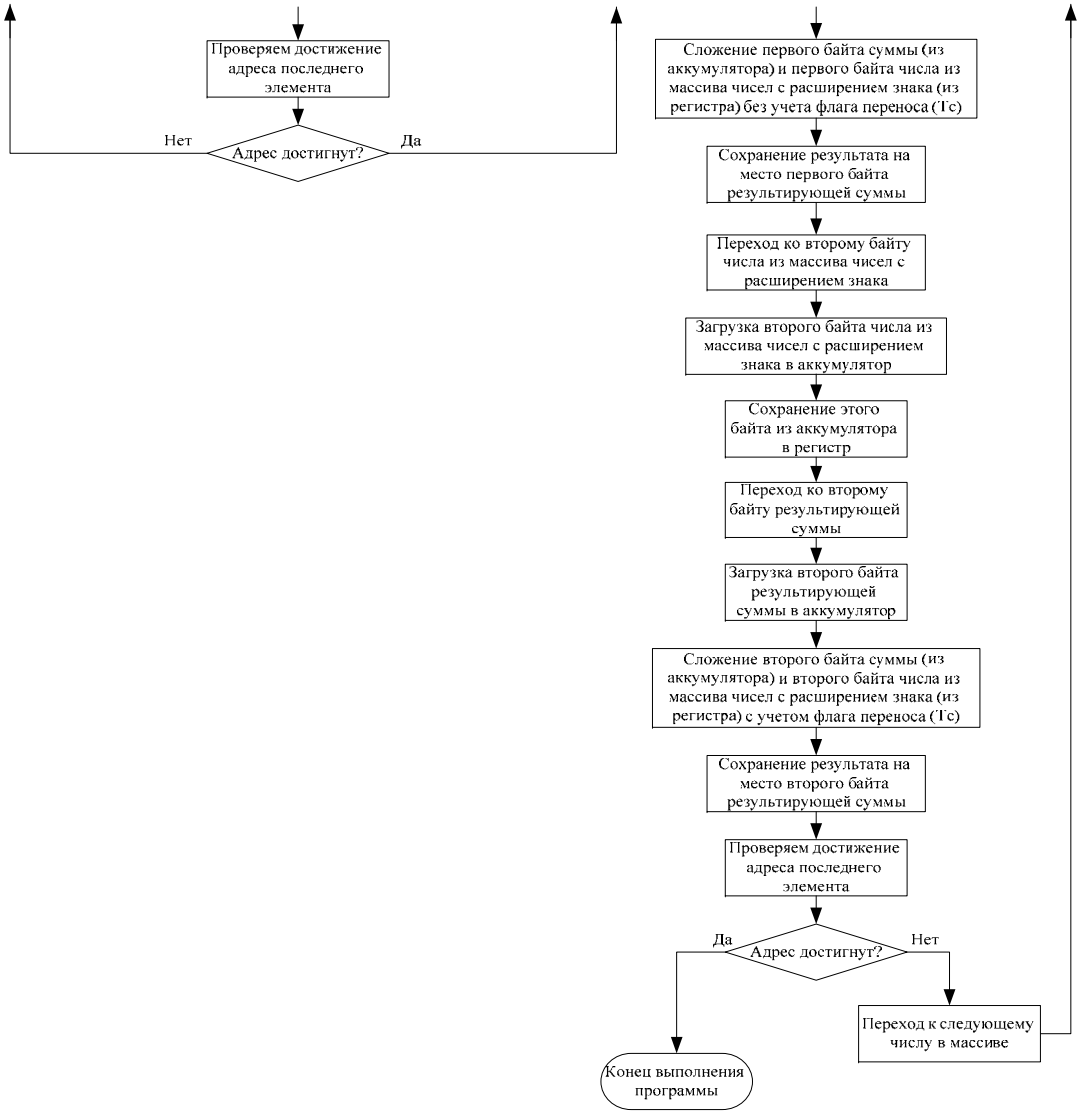
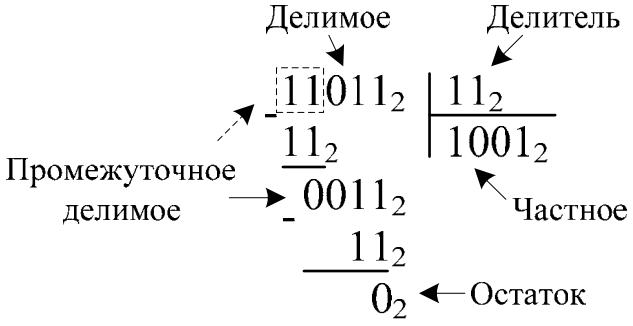



*Рис. 2.2. Выполнение операции расширения знака для массива 15-и двухбайтовых чисел и нахождение суммы чисел массива*

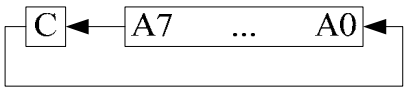
| Метка | Мнемоника | Комментарий |
|---|---|---|
| | lxi b, 0A00 | Загружаем начальный адрес исходного массива знаковых значений чисел (адрес первого элемента) |
| | lxi d, 0B00 | Загружаем начальный адрес массива знаковых значений чисел с расширением знака (адрес первого элемента) |
| Label2: | | |
| | ldax b | Загружаем **младший** байт числа из исходного массива |
| | stax d | Сохраняем этот **младший** байт числа на место **нулевого (младшего)** байта в массиве чисел с расширением знака |
| | inx b | Переходим к **старшему** байту в массиве исходных чисел |
| | inx d | Переходим к старшему байту в массиве чисел с расширением знаков |
| | ldax b | Загружаем **старший** байт числа из исходного массива |
| | stax d | Сохраняем этот **старший** байт числа на место **первого** байта в массиве чисел с расширением знака |
| | inx d | Переходим ко **второму (старшему)** байту в массиве чисел с расширением знаков |
| | mov h, a | Проверяем исходное число на знак (на наличие 1 в старшем бите старшего байта) |
| | ani 80 | |
| | jz Label1 | Заполняем второй (старший) байт числа в массиве чисел с расширением знака единицами, если число оказалось отрицательным и нулями – если положительным. |
| | mvi a, FF | |
| Label1: | | |
| | stax d | |
| | inx b | Переходим на следующий адрес в массиве исходных чисел |
| | inx d | Переходим на следующий адрес в массиве чисел с расширением знаков |
| | mvi a, 1E | Проверяем достижение последнего числа в массиве исходных чисел |
| | sub c | |
| | jnz Label2 | |
| | | Сложение чисел |
| | lxi b, 0B00 | Загружаем начальный адрес массива знаковых значений чисел с расширением знака (адрес первого элемента) |
| Label3: | | |
| | lxi d, 0AF0 | Загружаем адрес результирующей суммы (куда будем записывать результат сложения) |
| | ldax b | Загружаем **нулевой (младший)** байт числа |
| | mov h, a | Сохраняем его в регистре H |
| | ldax d | Загружаем **нулевой (младший)** байт суммы из адреса 0AF0 |
| | add h | Складываем нулевые (младшие) байты числа из регистра H и суммы из аккумулятора без учета флага переноса Tc |
| | stax d | Сохраняем результат по адресу 0AF0 |
| | inx b | Переходим к **первому** байту числа |
| | inx d | Переходим к **первому** байту суммы |
| | ldax b | Загружаем **первый** байт числа |
| | mov h, a | Сохраняем его в регистре H |
| | ldax d | Загружаем **первый** байт суммы из адреса 0AF1 |
| | adc h | Складываем первые байты числа из регистра H и суммы из аккумулятора с учетом флага переноса Tc |
| | stax d | Сохраняем результат по адресу 0AF1 |
| | inx b | Переходим ко **второму (старшему)** байту числа |
| | inx d | Переходим ко **второму (старшему)** байту суммы |
| | ldax b | Загружаем **второй (старший)** байт числа |
| | mov h, a | Сохраняем его в регистре H |
| | ldax d | Загружаем **второй (старший)** байт суммы из адреса 0AF2 |
| | adc h | Складываем вторые (старшие) байты числа из регистра H и суммы из аккумулятора с учетом флага переноса Tc |
| | stax d | Сохраняем результат по адресу 0AF2 |
| | inx b | Переходим к нулевому (младшему) байту следующего числа |
| | mvi a, 2D | Проверяем, просуммированы ли все числа |
| | sub c | |
| | jnz Label3 | |
| | rst1 | |

## 2.2. Выполнение операции деления суммы всех чисел на их количество

Деление двоичных чисел, как и чисел, представленных в любой другой системе счисления, основывается на последовательном вычитании делителя из делимого и остатков от деления, что является обычным делением в столбик. Однако двоичное деление можно реализовать проще, так как использование только двух цифр (0 и 1) исключает в каждом цикле деления необходимость определения числа делителей, содержащихся в текущем значении делимого или остатка (достаточно только их сравнить). Например, разделим число $1B_{16} = 00011011_2$ на число $03_{16} = 00000011_2$:




Для программной реализации получения промежуточного делимого будем сдвигать старшие биты делимого в младшие биты регистра, отведенного под это промежуточное делимое, до тех пор, пока он не будет больше, либо равен делителю. Затем будем вычитать из него делитель, а разность использовать в качестве старшего бита последующего промежуточного делимого. Эти операции будем проделывать до тех пор, пока не переберем все биты делимого (пока оно не обнулится в результате сдвига). Для выполнения операции сдвига воспользуемся командой циклического сдвига влево с переносом RAL:




080C) RAL

| Адрес/регистр | Код до выполнения операции | Код после выполнения операции |
|---|---|---|
| 080C | 17 | 17 |
| A | $D8_{16}=11011000_2$ | $B0_{16}=10110000_2$ |
| FL | 02(Флаг Tc=0) | 03(Флаг Tc=1) |
| PC | 080C | 080D |

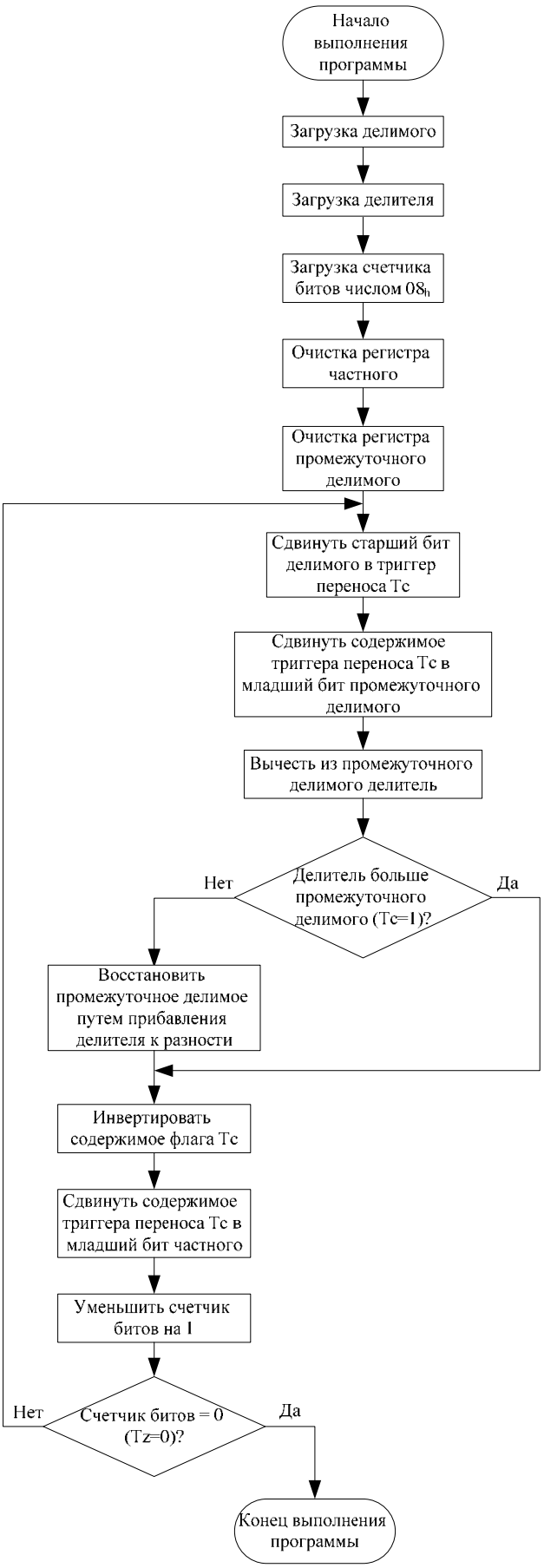



*Рис. 2.3. Деление двух однобайтовых чисел путем сдвига делимого*

При этом необходимо учитывать, что данный алгоритм реализует операцию целочисленного деления, т.е. деления при котором остатком является та часть числа, которая не разделилась на делитель (меньше делителя).

Делимое будем записывать в регистр E, делитель в регистр D, а результат получим в регистре H, остаток – в C.

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | mvi d, 0F | Загружаем делитель |
| | mvi c, 00 | Очищаем промежуточное делимое перед началом деления |
| | lxi h, 0008 | Загружаем счетчик битов в L и очищаем регистр частного H |
| Label2: | | |
| | mov a, e | Загружаем делимое в аккумулятор |
| | ral | Сдвигаем старший бит делимого в триггер переноса Tc |
| | mov e, a | Сохраняем результат в регистр делимого E |
| | mov a, c | Загружаем в аккумулятор промежуточное делимое из регистра C |
| | ral | Сдвигаем триггер переноса Tc в младший бит промежуточного делимого |
| | sub d | Вычитаем из промежуточного делимого делитель |
| | jnc Label1 | Если делитель больше промежуточного делимого (Tc=1), восстанавливаем промежуточное делимое путем прибавления делителя к разности |
| | add d | |
| Label1: | | |
| | mov c, a | Сохраняем промежуточное делимое в регистр C |
| | cmc | Инвертируем триггер переноса Tc |
| | mov a, h | Загружаем частное в аккумулятор |
| | ral | Сдвигаем триггер переноса Tc в младший бит регистра частного H |
| | mov h, a | Сохраняем результат в регистр частного H |
| | dcr l | Проверяем, отработаны ли все 8 битов |
| | jnz Label2 | |
| | rst1 | |

Другим способом деления является циклическое выполнение операции вычитания делителя из делимого до тех пор, пока делимое не станет меньше делителя (рис. 2.4). В этом случае количество циклов вычитания есть частное от деления, а делимое, ставшее меньше делителя, есть целочисленный остаток.

Делимое, как и в предыдущем варианте деления, будем записывать в регистр E, делитель в регистр D, а результат получим в регистре H, остаток – в C.

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | mvi d,0F | Загружаем делитель |
| | mov a,e | Загружаем делимое в аккумулятор |
| | mvi h,00 | Очищаем регистр частного H |
| Label1: | | |
| | mov c,a | Сохраняем промежуточное делимое в в регистре H |
| | sub d | Вычитаем из промежуточного делимого делитель |
| | inr h | Увеличиваем на 1 частное |
| | jnc Label1 | Если флаг Tc=0, то продолжаем выполнять вычитание, если нет – восстанавливаем значение частного |
| | dcr h | |
| | rst1 | |

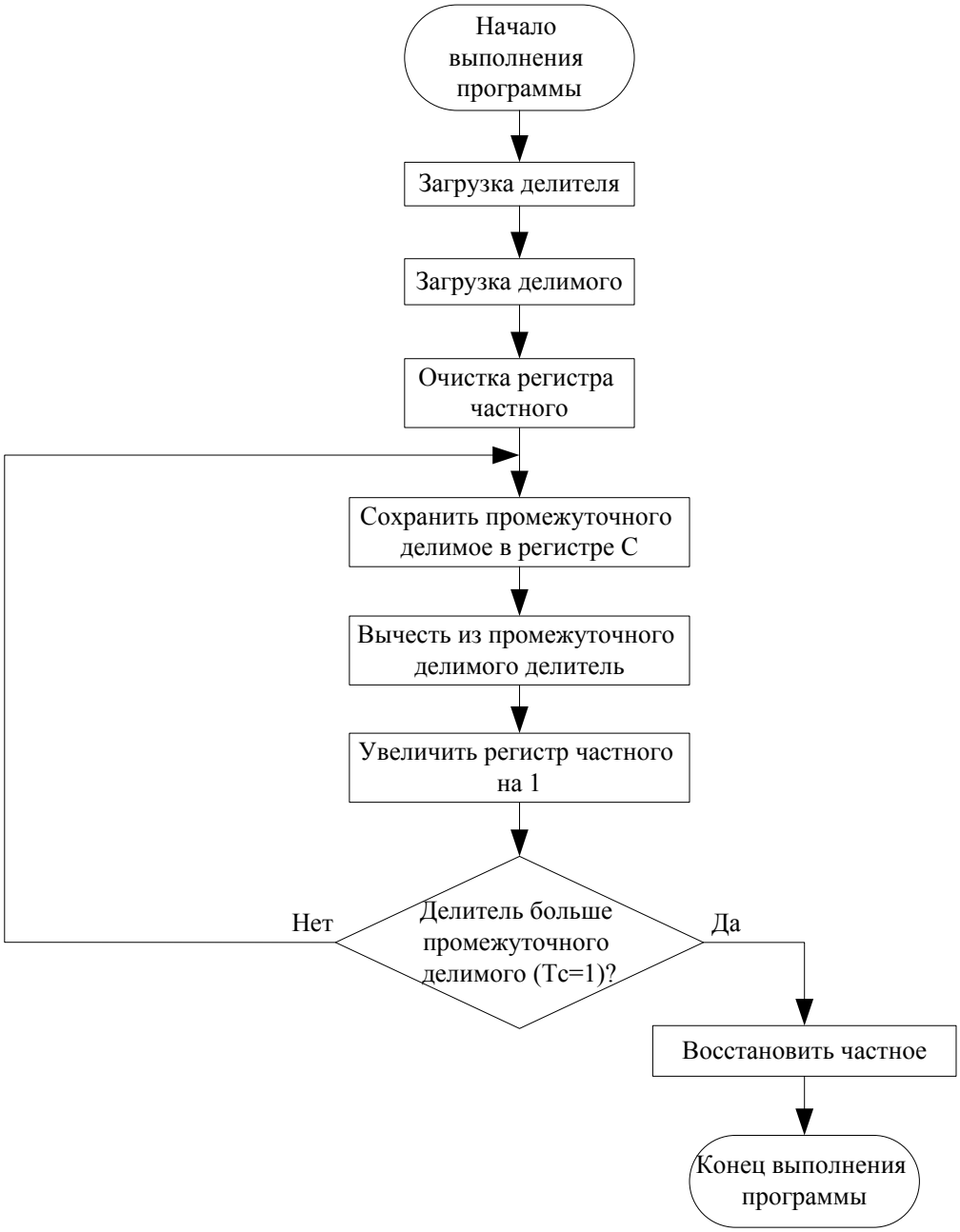



*Рис. 2.4. Деление двух однобайтовых чисел путем вычитания делимого*

Таким образом, мы разделили одно однобайтовое число на другое, но для решения поставленной задачи нам необходимо делить трехбайтовое число на однобайтовое. Для этого будем делить каждый байт в отдельности, начиная со старшего, а между операциями деления не будем обнулять значение промежуточного делимого, чтобы учесть переносы между байтами. Трехбайтовое делимое будет записано начиная с адреса 0AF0, а результат начиная с 0AF6, однобайтовый остаток по адресу 0AF5.

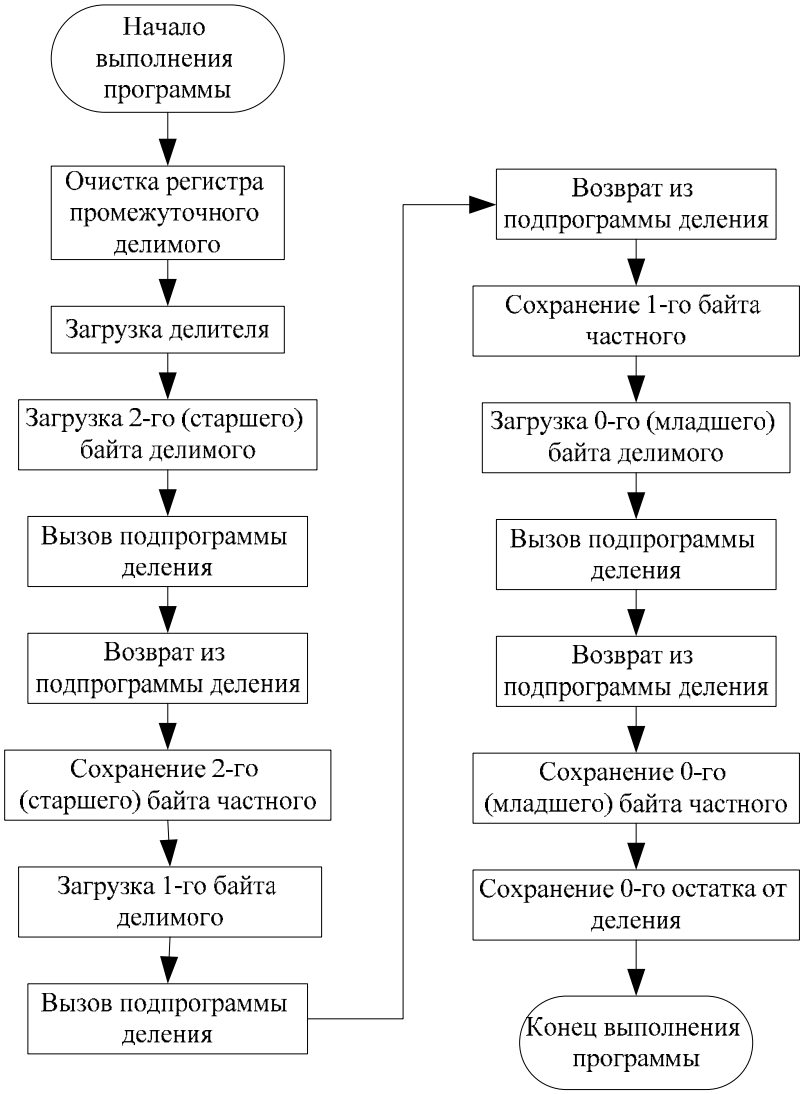



*Рис. 2.5. Деление трехбайтового числа на однобайтовое путем сдвига делимого*

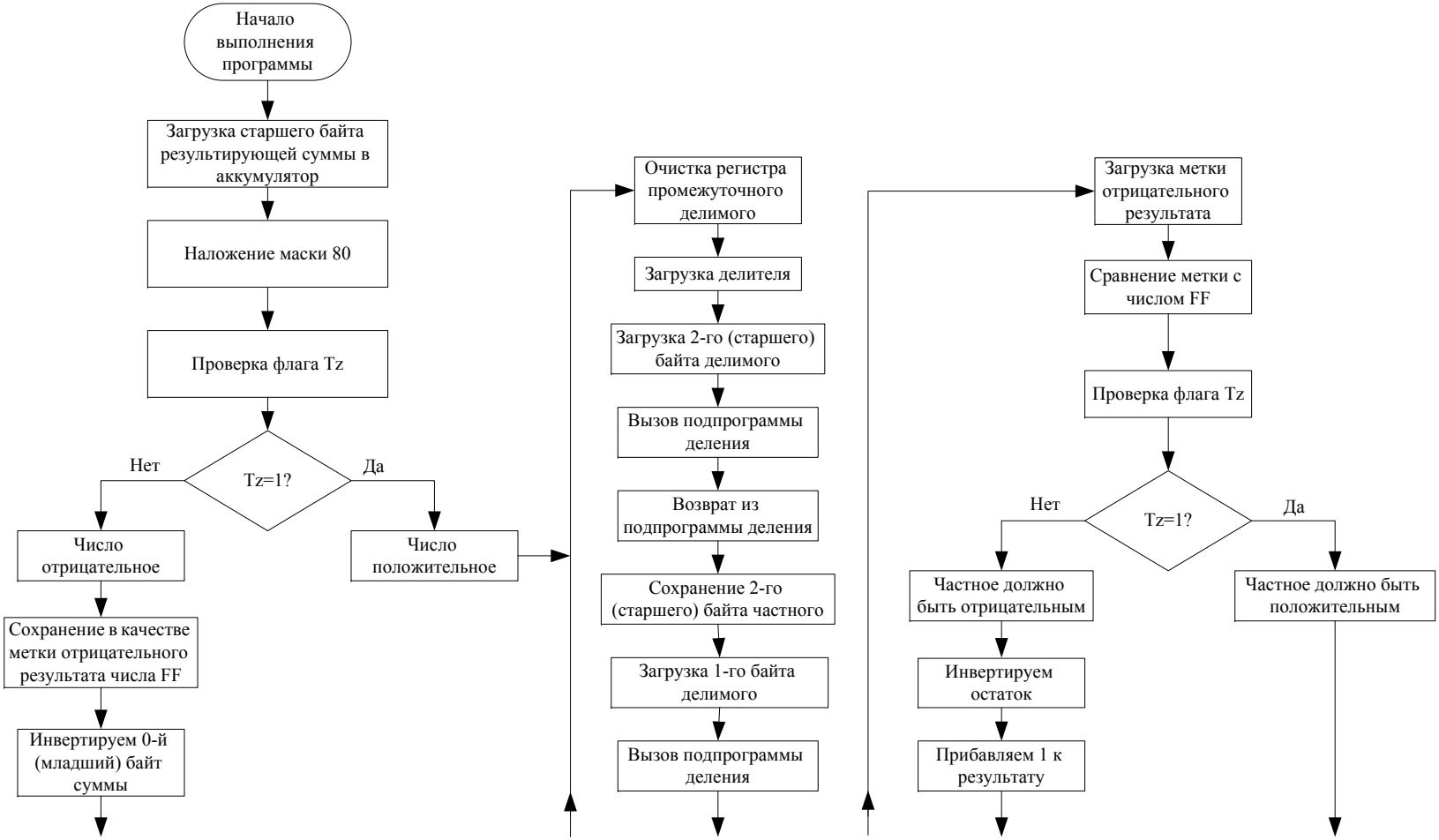
| Метка | Мнемоника | Комментарий |
|---|---|---|
| | mvi c, 00 | Очищаем промежуточное делимое перед началом деления |
| | mvi d, 0F | Загружаем делитель |
| | lda 0AF2 | Загружаем 2-й байт делимого |
| | mov e, a | |
| | call Label1 | Вызываем подпрограмму деления |
| | mov a, h | Сохраняем 2-й байт результата |
| | sta 0AF8 | |
| | lda 0AF1 | Загружаем 1-й байт делимого |
| | mov e, a | |
| | call Label1 | Вызываем подпрограмму деления |
| | mov a, h | Сохраняем 1-й байт результата |
| | sta 0AF7 | |
| | lda 0AF0 | Загружаем 0-й байт делимого |
| | mov e, a | |
| | call Label1 | Вызываем подпрограмму деления |
| | mov a, h | Сохраняем 0-й байт результата |
| | sta 0AF6 | |

*Продолжение таблицы*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | mov a, c | Сохраняем остаток от деления |
| | sta 0AF5 | |
| | rst1 | |
| | | |
| Label1: | | Подпрограмма деления |
| | lxi h, 0008 | Загружаем счетчик битов в L и очищаем регистр частного H |
| Label3: | | |
| | mov a, e | Загружаем делимое в аккумулятор |
| | ral | Сдвигаем старший бит делимого в триггер переноса Tc |
| | mov e, a | Сохраняем результат в регистр делимого E |
| | mov a, c | Загружаем в аккумулятор промежуточное делимое из регистра C |
| | ral | Сдвигаем триггер переноса Tc в младший бит промежуточного делимого |
| | sub d | Вычитаем из промежуточного делимого делитель |
| | jnc Label2 | Если делитель больше промежуточного делимого (Tc=1), восстанавливаем промежуточное делимое путем прибавления делителя к разности |
| | add d | |
| Label2: | | |
| | mov c, a | Сохраняем промежуточное делимое в регистр C |
| | cmc | Инвертируем триггер переноса Tc |
| | mov a, h | Загружаем частное в аккумулятор |
| | ral | Сдвигаем триггер переноса Tc в младший бит регистра частного H |
| | mov h, a | Сохраняем результат в регистр частного H |
| | dcr l | Проверяем отработаны ли все 8 битов |
| | jnz Label3 | |
| | ret | |

Поскольку делить необходимо не только положительные числа, но и отрицательные, то перед выполнением операции деления над результирующей суммой из пятнадцати чисел необходимо ввести проверку этой суммы на знак. Если она окажется отрицательной, следует получить ее прямой код, сохранив при этом так называемую метку, указывающую на то, что после выполнения деления частному и остатку от деления необходимо вернуть их отрицательный знак (привести его к обратному дополнительному коду).

В качестве метки будем сохранять число FF по адресу 0AF3, если число окажется отрицательным. После выполнения операции деления необходимо будет проверить наличие этого числа FF по указанному адресу. Если оно там присутствует, то необходимо получить обратный дополнительный код от результата деления.


Начало выполнения программы
Загрузка старшего байта результирующей суммы в аккумулятор
Наложение маски 80
Проверка флага Tz
Tz=1?
Нет
Да
Число отрицательное
Число положительное
Сохранение в качестве метки отрицательного результата числа FF
Инвертируем 0-й (младший) байт суммы
Очистка регистра промежуточного делимого
Загрузка делителя
Загрузка 2-го (старшего) байта делимого
Вызов подпрограммы деления
Возврат из подпрограммы деления
Сохранение 2-го (старшего) байта частного
Загрузка 1-го байта делимого
Вызов подпрограммы деления
Загрузка метки отрицательного результата
Сравнение метки с числом FF
Проверка флага Tz
Tz=1?
Нет
Да
Частное должно быть отрицательным
Частное должно быть положительным
Инвертируем остаток
Прибавляем 1 к результату

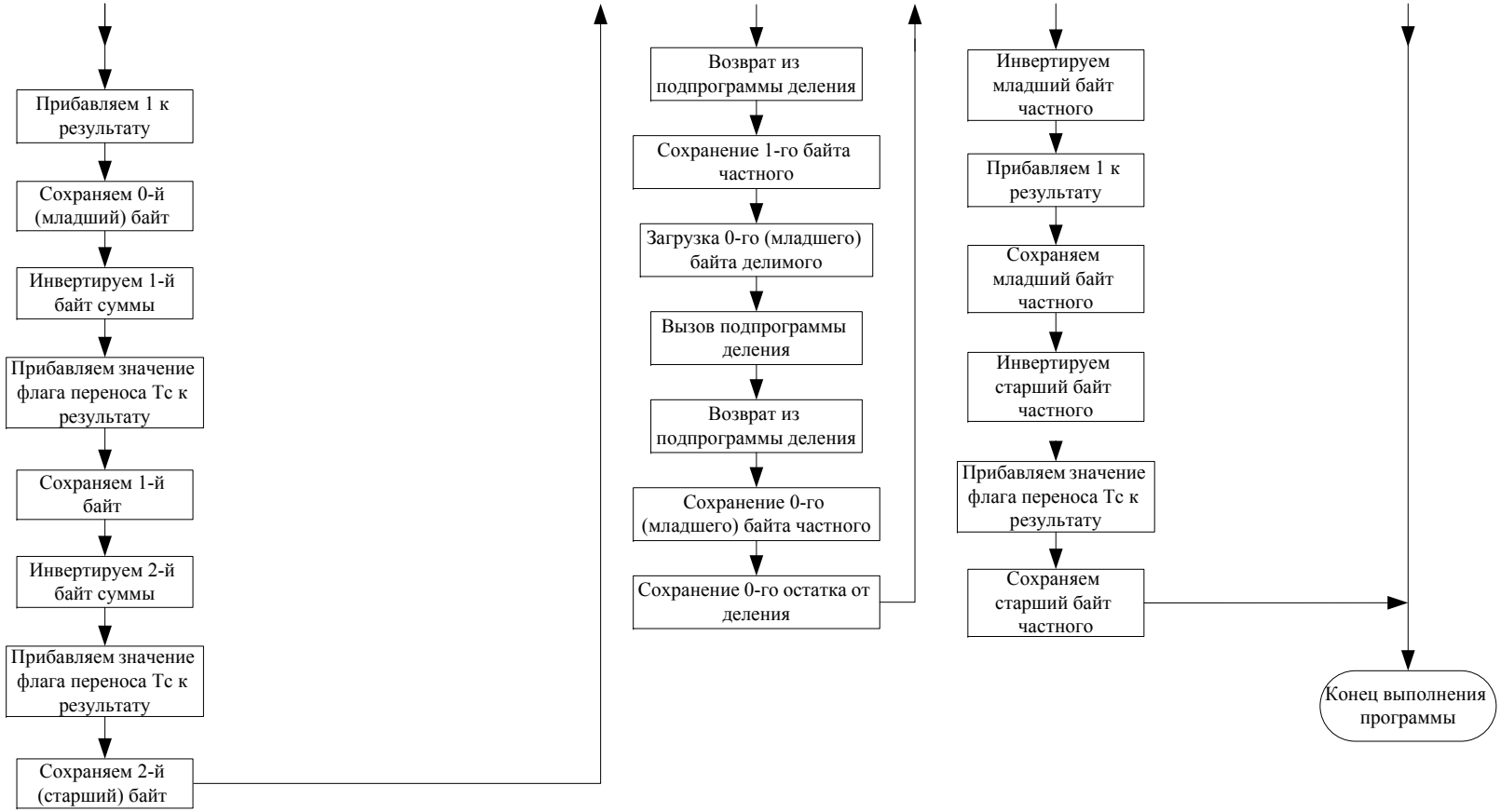



*Рис. 2.6. Деление с учетом знака трехбайтового числа на однобайтовое путем сдвига делимого*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | lda 0AF2 | Загружаем старший байт результирующей суммы |
| | ani 80 | Проверяем результат на отрицательность |
| | jz Label1 | |
| | mvi a, FF | Сохраняем метку отрицательного результата |
| | sta 0AF3 | |
| | lda 0AF0 | Загружаем 0-й (младший) байт суммы |
| | cma | Получаем прямой код 0-го (младшего) байта путем его инвертирования и прибавления 1 без учета флага переноса Tc |
| | adi 01 | |
| | sta 0AF0 | Сохраняем прямой код 0-го (младшего) байта |
| | lda 0AF1 | Загружаем 1-й байт суммы |
| | cma | Получаем прямой код 1-го байта путем его инвертирования и прибавления флага переноса Tc |
| | aci 00 | |
| | sta 0AF1 | Сохраняем прямой код 1-го байта |
| | lda 0AF2 | Загружаем 2-й (старший) байт суммы |
| | cma | Получаем прямой код 2-го (старшего) байта путем его инвертирования и прибавления флага переноса Tc |
| | aci 00 | |
| | sta 0AF2 | Сохраняем прямой код 2-го (младшего) байта |
| Label1: | | |
| | mvi c, 00 | Очищаем промежуточное делимое перед началом деления |
| | mvi d, 0F | Загружаем делитель |
| | lda 0AF2 | Загружаем 2-й байт делимого |
| | mov e, a | |
| | call Label2 | Вызываем подпрограмму деления |
| | mov a, h | Сохраняем 2-й байт результата |
| | sta 0AF8 | |
| | lda 0AF1 | Загружаем 1-й байт делимого |
| | mov e, a | |
| | call Label2 | Вызываем подпрограмму деления |
| | mov a, h | Сохраняем 1-й байт результата |
| | sta 0AF7 | |
| | lda 0AF0 | Загружаем 0-й байт делимого |
| | mov e, a | |
| | call Label2 | Вызываем подпрограмму деления |
| | mov a, h | Сохраняем 0-й байт результата |
| | sta 0AF6 | |
| | mov a, c | Сохраняем остаток от деления |
| | sta 0AF5 | |
| | lda 0AF3 | Проверяем наличие метки отрицательного результата и, если ее нет, то прекращаем выполнение программы, если есть – получаем обратный дополнительный код частного |
| | cpi FF | |
| | cnz Label3 | |
| | lda 0AF5 | Загружаем остаток |
| | cma | Получаем прямой код остатка путем его инвертирования и прибавления 1 к результату без учета флага переноса Tc |
| | adi 01 | |
| | sta 0AF5 | Сохраняем результат |

| *Продолжение таблицы* | | |
|---|---|---|
| Метка | Мнемоника | Комментарий |
| | lda 0AF6 | Загружаем 0-й (младший) байт частного |
| | cma | Получаем обратный дополнительный код 0-го (младшего) байта частного путем его инвертирования и прибавления 1 без учета флага переноса Tc |
| | adi 01 | |
| | sta 0AF6 | Сохраняем 0-й байт результата |
| | lda 0AF7 | Загружаем 1-й (старший) байт частного |
| | cma | Получаем обратный дополнительный код 1-го (старшего) байта частного путем его инвертирования и прибавления флага перено-са Tc |
| | aci 00 | |
| | sta 0AF7 | Сохраняем 1-й (старший) байт результата |
| Label3: | | |
| | rst1 | |
| | | |
| Label2: | | Подпрограмма деления |
| | lxi h, 0008 | Загружаем счетчик битов в L и очищаем регистр частного H |
| Label5: | | |
| | mov a, e | Загружаем делимое в аккумулятор |
| | ral | Сдвигаем старший бит делимого в триггер переноса Tc |
| | mov e, a | Сохраняем результат в регистр делимого E |
| | mov a, c | Загружаем в аккумулятор промежуточное делимое из регистра C |
| | ral | Сдвигаем триггер переноса Tc в младший бит промежуточного делимого |
| | sub d | Вычитаем из промежуточного делимого делитель |
| | jnc Label4 | Если делитель больше промежуточного делимого (Tc=1), восста-навливаем промежуточное делимое путем прибавления делителя к разности |
| | add d | |
| Label4: | | |
| | mov c, a | Сохраняем промежуточное делимое в регистр C |
| | cmc | Инвертируем триггер переноса Tc |
| | mov a, h | Загружаем частное в аккумулятор |
| | ral | Сдвигаем триггер переноса Tc в младший бит регистра частного H |
| | mov h, a | Сохраняем результат в регистр частного H |
| | dcr l | Проверяем отработаны ли все 8 битов |
| | jnz Label5 | |
| | ret | |

# 3. КОМПИЛЯЦИЯ ПРОГРАММЫ

Таким образом, мы разобрались в типовом задании на курсовое проектирование и остается только собрать кусочки уже написанного листинга в общую программу.

Хорошим тоном программирования могло бы стать небольшое добавление к уже разработанным блокам общего кода, заключающееся в очистке (обнулении) всех используемых в программе адресов перед ее выполнением (кроме адресов, отведенных под исходный массив) для исключения возможных «наложений» вычислений, например в процессе повторного прогона программы.

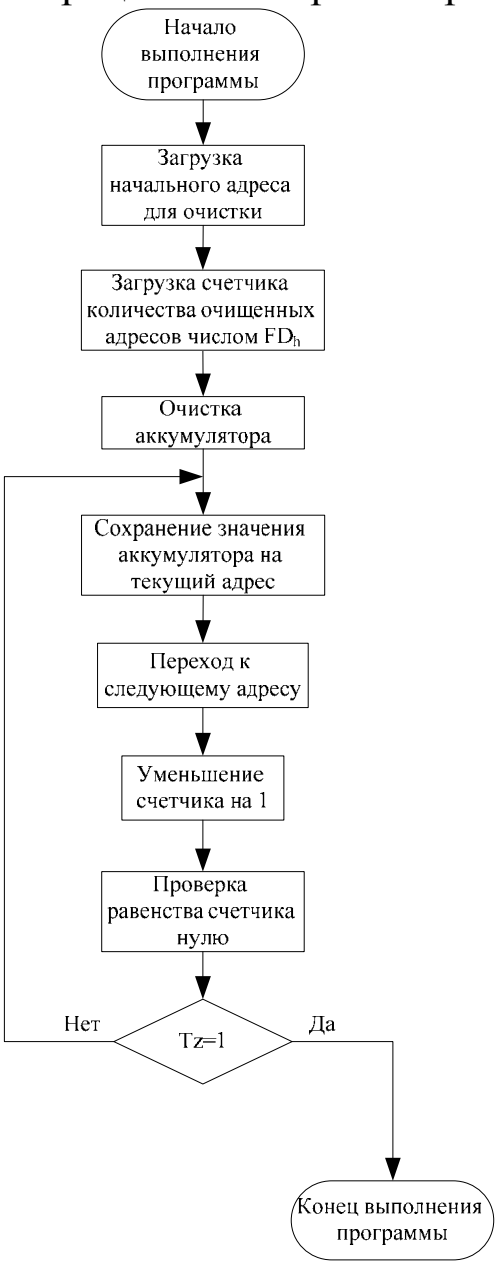
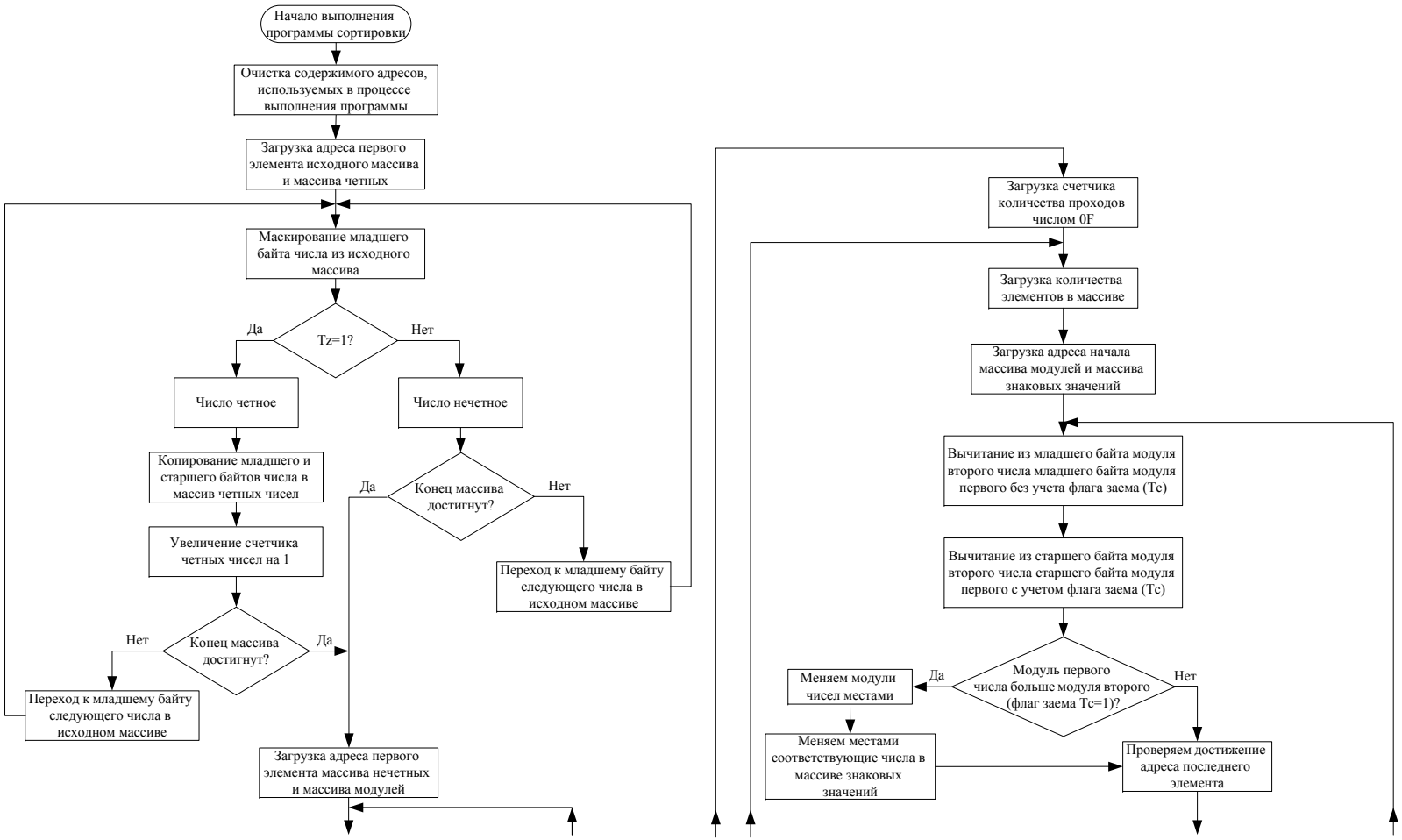



*Рис. 3.1. Очистка рабочих адресов*

Очистим адреса, начиная с 0A30 (с него начинается запись массива четных чисел) и заканчивая адресом 0B2D (на нем заканчивается массив чисел с расширением знаков). Поскольку обнулять значения адресов будем в цикле, то условием выхода из цикла будет количество очищенных байт – $FD_h$ (0B2D – 0A30).

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | lxi b,0A30 | Загружаем начальный адрес (с которого будем очищать) |
| | mvi d,FD | Загружаем счетчик количества адресов числом FD |
| | xra a | Очищаем аккумулятор |
| Label1: | | |
| | stax b | Обнуляем значение ячейки адреса |
| | inx b | Переходим к следующему адресу |
| | dcr d | Уменьшаем счетчик |
| | jnz Label1 | Проверяем равенство счетчика нулю, если равен, то прекращаем выполнять обнуление |
| | rst1 | |

Теперь остается собрать готовые блоки в единую программу. Поскольку каждый такой блок был рассмотрен достаточно подробно, приведем только упрощенный алгоритм всего кода в целом (рис. 3.2, 3.3).


Начало выполнения программы сортировки
Очистка содержимого адресов, используемых в процессе выполнения программы
Загрузка адреса первого элемента исходного массива и массива четных
Маскирование младшего байта числа из исходного массива
Да
Tz=1?
Нет
Число четное
Число нечетное
Копирование младшего и старшего байтов числа в массив четных чисел
Увеличение счетчика четных чисел на 1
Нет
Конец массива достигнут?
Да
Переход к младшему байту следующего числа в исходном массиве
Да
Конец массива достигнут?
Нет
Переход к младшему байту следующего числа в исходном массиве
Загрузка адреса первого элемента массива нечетных и массива модулей
Загрузка счетчика количества проходов числом 0F
Загрузка количества элементов в массиве
Загрузка адреса начала массива модулей и массива знаковых значений
Вычитание из младшего байта модуля второго числа младшего байта модуля первого без учета флага заема (Tc)
Вычитание из старшего байта модуля второго числа старшего байта модуля первого с учетом флага заема (Tc)
Да
Модуль первого числа больше модуля второго (флаг заема Tc=1)?
Нет
Меняем модули чисел местами
Меняем местами соответствующие числа в массиве знаковых значений
Проверяем достижение адреса последнего элемента

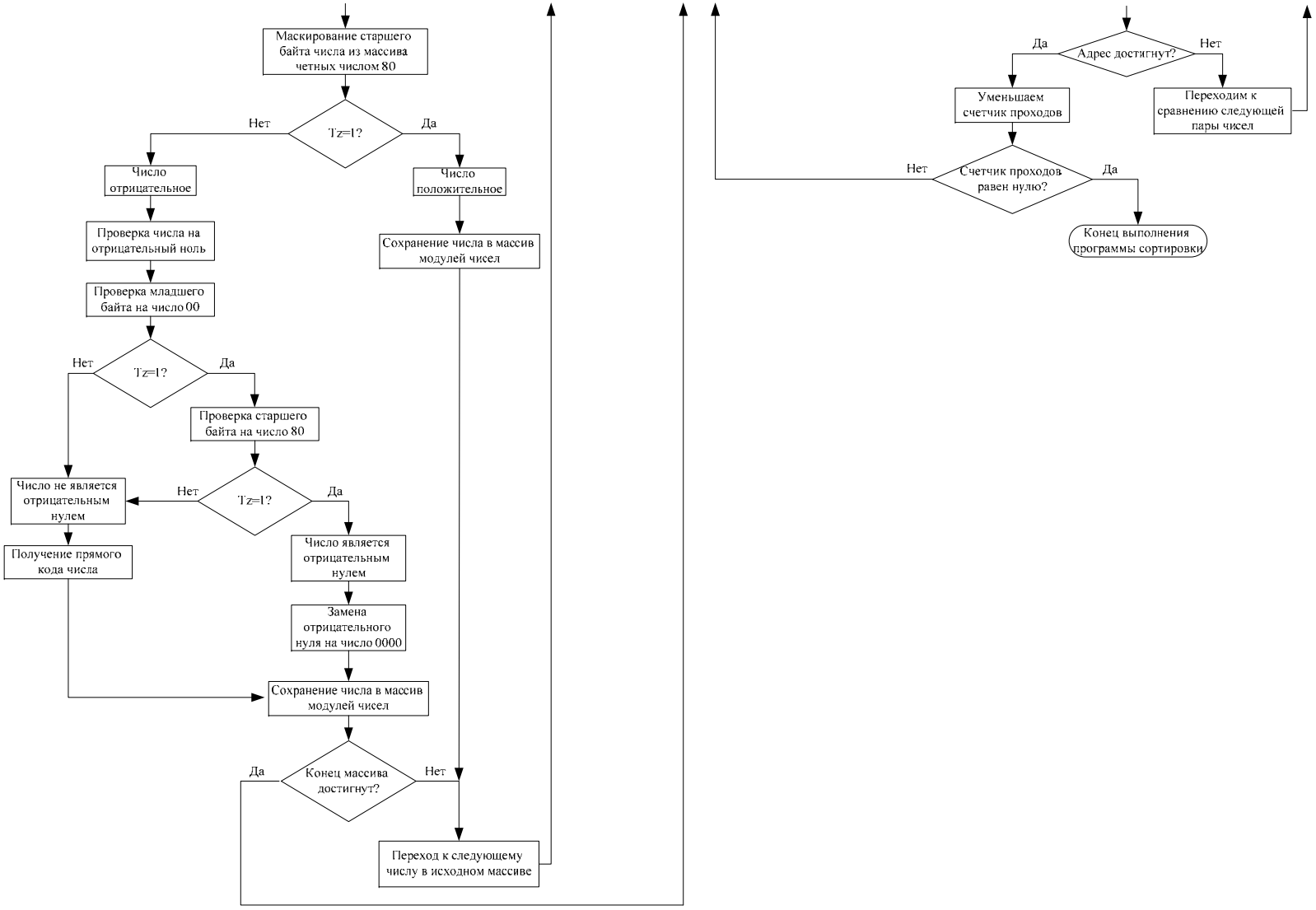



*Рис. 3.2. Сортировка по возрастанию всех четных чисел с учетом их модулей*

Начало выполнения программы получения среднего арифметического
Загрузка адреса начала исходного массива и массива чисел с расширением знака
Сохранение нулевого и первого байта из исходного массива на место нулевого и первого байта в массиве чисел с расширением знака
Наложение маски на первый байт 80 для проверки наличия 1 или 0 в старшем бите
Да
Tz=1?
Нет
Запись в аккумулятор число 00
Запись в аккумулятор число FF
Сохранение значения аккумулятора на место второго байта в массиве чисел с расширением знака
Проверяем достижение адреса последнего элемента
Нет
Адрес достигнут?
Да
Загрузка адреса последнего элемента массива чисел с расширением знака
Маскирование старшего байта результирующей суммы числом 80
Нет
Tz=1?
Да
Число отрицательное
Число положительное
Сохранение в качестве метки отрицательного результата числа FF
Получение прямого кода результирующей суммы
Очистка регистра промежуточного делимого
Загрузка делителя
Загрузка 2-го (старшего) байта делимого
Вызов подпрограммы деления
Возврат из подпрограммы деления
Сохранение 2-го (старшего) байта частного
Загрузка 1-го байта делимого
Вызов подпрограммы деления
Возврат из подпрограммы деления
Сохранение 1-го байта частного
Загрузка 0-го (младшего) байта делимого
Вызов подпрограммы деления

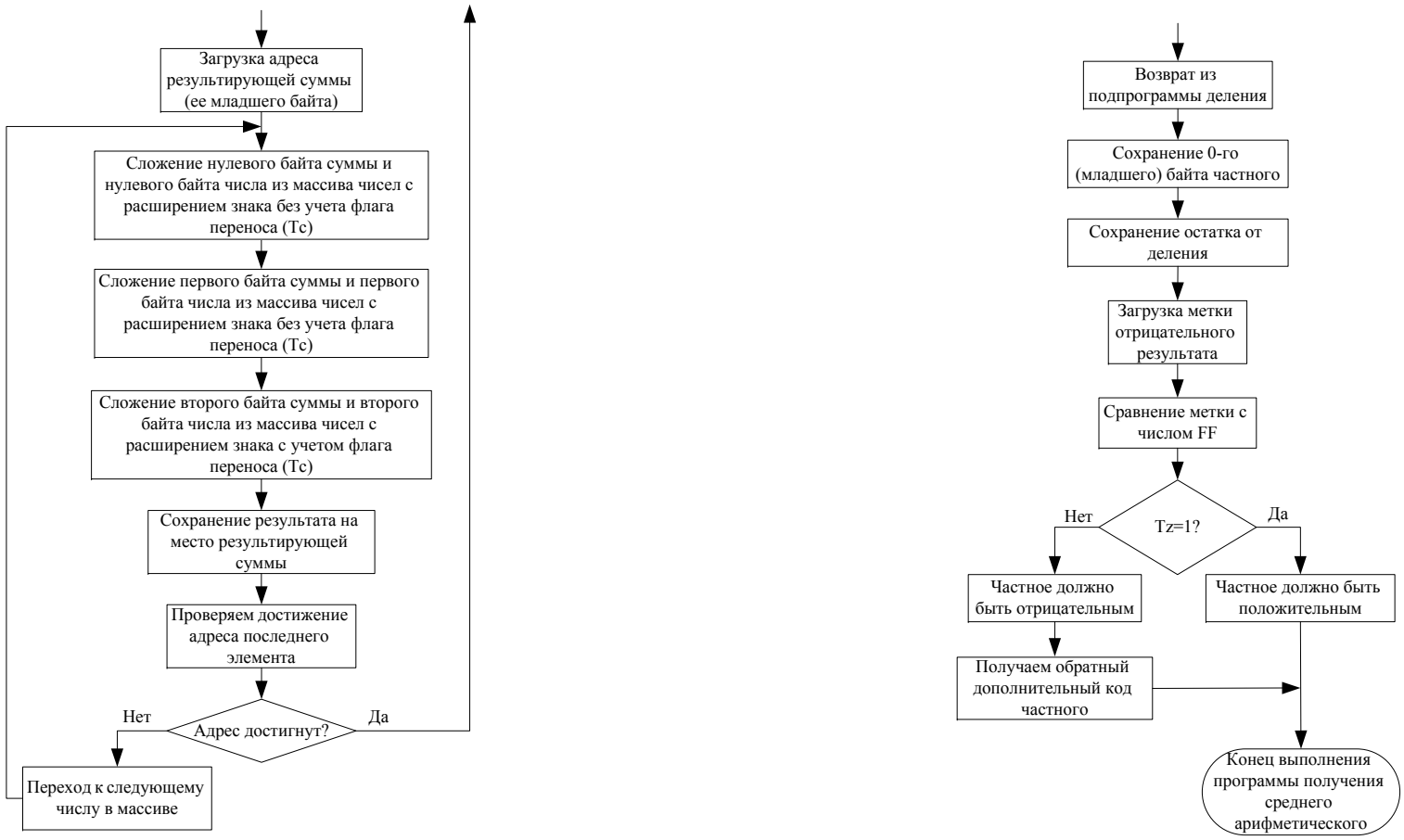



*Рис. 3.3. Среднее арифметическое всех чисел*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | | **Очистка используемых адресов** |
| | lxi b,0A30 | Загружаем начальный адрес (с которого будем очищать) |
| | mvi d,FD | Загружаем счетчик количества адресов числом FD |
| | xra a | Очищаем аккумулятор |
| Label61: | | |
| | stax b | Обнуляем значение ячейки адреса |
| | inx b | Переходим к следующему адресу |
| | dcr d | Уменьшаем счетчик |
| | jnz Label61 | Проверяем равенство счетчика нулю, если равен, то прекращаем выполнять обнуление |
| | | **Выборка четных чисел** |
| | lxi b, 0A00 | Загрузка начальных адресов |
| | lxi d, 0A30 | |
| Label12: | | |
| | ldax b | Загрузка числа из массива |
| | ani 01 | Проверка на четность (если нечетно, то переходим на Label1, если четно – выполняем следующую комманду) |
| | jnz Label11 | |
| | ldax b | Пересохраняем **младший байт** числа на новый адрес |
| | stax d | |
| | inx b | Увеличиваем адрес для перехода к следующему байту числа |
| | inx d | Увеличиваем адрес для перехода к следующему байту числа |
| | ldax b | Пересохраняем **старший байт** числа на новый адрес |
| | stax d | |
| | dcx b | Уменьшаем адрес для возврата к младшему байту |
| | inx d | Увеличиваем адрес |
| | lda 0AA0 | Загружаем счетчик четных чисел с адреса 0AA0 |
| | adi 01 | Увеличиваем счетчик четных чисел на 1 и сохраняем его по адресу 0AA0 |
| | sta 0AA0 | |
| Label11: | | |
| | inx b | **Дважды увеличиваем адрес** для перехода к младшему байту следующего числа |
| | inx b | |
| | mvi a, 1E | Проверяем окончание цикла, если адрес в регистрах B и C ≠ (т.е. <) 1E, то продолжаем выполнение цикла, если C=1E– прекращаем выполнение программы |
| | sub c | |
| | jnz Label12 | |
| | | **Определение модулей четных чисел** |
| | lxi b, 0A30 | Загрузка начальных адресов |
| | lxi d, 0A50 | |
| Label22: | | |
| | inx b | Переходим к **старшему** байту числа в исходном массиве |
| | ldax b | Загрузка старшего байта числа |
| | ani 80 | Проверка на знак |
| | dcx b | Переходим к младшему байту числа в исходном массиве |
| | ldax b | Загрузка младшего байта числа |
| | jnz Label21 | Если число отрицательно, то переходим на подпрограмму на Label1, если положительно – выполняем следующую команду. |
| | stax d | Пересохраняем младший байт числа на новый адрес |
| | inx b | Увеличиваем адрес для перехода к старшему байту числа в исходном массиве |
| | inx d | Увеличиваем адрес для перехода к старшему байту числа в массиве модулей чисел |
| | ldax b | Загрузка старшего байта числа |
| Label24: | | |
| | stax d | Пересохраняем старший байт числа на новый адрес |
| | inx b | Увеличиваем адрес для перехода к следующему элементу исходного массива |

*Продолжение таблицы*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | inx d | Увеличиваем адрес для перехода к следующему элементу массива модулей чисел |
| | mvi a, 4E | Проверяем окончание цикла, если адрес в регистрах B и C ≠ (т.е. <) 4E, то продолжаем выполнение цикла, если C=4E – прекращаем выполнение программы |
| | sub c | |
| | jnz Label22 | |
| | | **Сортировка модулей четных чисел и возврат к их знаковым значениям** |
| | mvi a, 0F | Загружаем счетчик количества проходов внешнего цикла |
| | sta 0A70 | |
| Label32: | | |
| | lda 0AA0 | Загружаем количество элементов массива |
| | mov l,a | |
| | lda 0A70 | Уменьшаем счетчик количества проходов |
| | sui 01 | |
| | sta 0A70 | |
| | jz Label31 | Проверяем условие окончания сортировки |
| | lxi b, 0A50 | Загружаем начальный адрес массива модулей чисел (адрес первого элемента) |
| | lxi d, 0A30 | Загружаем начальный адрес массива знаковых значений чисел (адрес первого элемента) |
| Label33: | | |
| | mov a, l | Проверяем достижение последнего элемента массива, если он достигнут – переходим на проверку счетчика количества проходов |
| | sui 01 | |
| | mov l,a | |
| | jz Label32 | |
| | ldax b | Загружаем младший байт модуля первого числа |
| | mov h, a | Сохраняем его в регистре D |
| | inx b | Переходим к младшему байту модуля второго числа |
| | inx b | |
| | ldax b | Загружаем младший байт модуля второго числа |
| | sub h | Вычитаем из него младший байт модуля первого числа из регистра H |
| | dcx b | Переходим к старшему байту модуля первого числа |
| | ldax b | Загружаем старший байт модуля первого числа |
| | mov h, a | Сохраняем его в регистре H |
| | inx b | Переходим к старшему байту модуля второго числа |
| | inx b | |
| | ldax b | Загружаем старший байт модуля второго числа |
| | sbb h | Вычитаем из него старший байт модуля первого числа из регистра D **с учетом флага заема (Tc)** |
| | dcx b | Переходим к младшему байту модуля второго числа |
| | inx d | Переходим к младшему байту знакового значения второго числа |
| | inx d | |
| | jnc Label33 | Если есть заем (т.е. первое число оказалось больше второго), то меняем числа местами, если нет заема – переходим к проверке достижения последнего элемента массива |
| | call Label34 | Переходим на подпрограмму обмена для обмена местами **старших** байтов модулей чисел |
| | dcx b | Переходим к младшему байту модуля первого числа |
| | dcx b | |
| | dcx b | |
| | ldax b | Загружаем младший байт модуля первого числа |
| | mov h, a | Сохраняем его в регистре D |
| | inx b | Переходим к младшему байту модуля второго числа |

*Продолжение таблицы*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | call Label34 | Переходим на подпрограмму обмена для обмена местами **младших** байтов модулей чисел |
| | dcx d | Переходим к старшему байту знакового значения первого числа |
| | call Label35 | Переходим на подпрограмму обмена для обмена местами **старших** байтов знаковых значений чисел, соответствующих **старшим** байтам модулей первого и второго чисел |
| | dcx d | Переходим к младшему байту знакового значения первого числа |
| | dcx d | |
| | dcx d | |
| | call Label35 | Переходим на подпрограмму обмена для обмена местами **младших** байтов знаковых значений чисел, соответствующих **младшим** байтам модулей первого и второго чисел |
| | jmp Label33 | Переходим к проверке достижения последнего элемента массива |
| Label31: | | |
| | | **Нахождение суммы всех чисел** |
| | lxi b, 0A00 | Загружаем начальный адрес исходного массива знаковых значений чисел (адрес первого элемента) |
| | lxi d, 0B00 | Загружаем начальный адрес массива знаковых значений чисел с расширением знака (адрес первого элемента) |
| Label42: | | |
| | ldax b | Загружаем **младший** байт числа из исходного массива |
| | stax d | Сохраняем этот **младший** байт числа на место **нулевого (младшего)** байта в массиве чисел с расширением знака |
| | inx b | Переходим к **старшему** байту в массиве исходных чисел |
| | inx d | Переходим к старшему байту в массиве чисел с расширением знаков |
| | ldax b | Загружаем **старший** байт числа из исходного массива |
| | stax d | Сохраняем этот **старший** байт числа на место **первого** байта в массиве чисел с расширением знака |
| | inx d | Переходим ко **второму (старшему)** байту в массиве чисел с расширением знаков |
| | mov h, a | Проверяем исходное число на знак (на наличие 1 в старшем бите старшего байта) |
| | ani 80 | |
| | jz Label41 | Заполняем второй (старший) байт числа в массиве чисел с расширением знака единицами, если число оказалось отрицательным и нулями – если положительным. |
| | mvi a, FF | |
| Label41: | | |
| | stax d | |
| | inx b | Переходим на следующий адрес в массиве исходных чисел |
| | inx d | Переходим на следующий адрес в массиве чисел с расширением знаков |
| | mvi a, 1E | Проверяем достижение последнего числа в массиве исходных чисел |
| | sub c | |
| | jnz Label42 | |
| | | Сложение чисел |
| | lxi b, 0B00 | Загружаем начальный адрес массива знаковых значений чисел с расширением знака (адрес первого элемента) |
| Label43: | | |
| | lxi d, 0AF0 | Загружаем адрес результирующей суммы (куда будем записывать результат сложения) |
| | ldax b | Загружаем **нулевой (младший)** байт числа |
| | mov h, a | Сохраняем его в регистре H |
| | ldax d | Загружаем **нулевой (младший)** байт суммы из адреса 0AF0 |
| | add h | Складываем нулевые (младшие) байты числа из регистра H и суммы из аккумулятора без учета флага переноса Tc |
| | stax d | Сохраняем результат по адресу 0AF0 |

*Продолжение таблицы*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | inx b | Переходим к **первому** байту числа |
| | inx d | Переходим к **первому** байту суммы |
| | ldax b | Загружаем **первый** байт числа |
| | mov h, a | Сохраняем его в регистре H |
| | ldax d | Загружаем **первый** байт суммы из адреса 0AF1 |
| | adc h | Складываем первые байты числа из регистра H и суммы из аккумулятора с учетом флага переноса Tc |
| | stax d | Сохраняем результат по адресу 0AF1 |
| | inx b | Переходим ко **второму (старшему)** байту числа |
| | inx d | Переходим ко **второму (старшему)** байту суммы |
| | ldax b | Загружаем **второй (старший)** байт числа |
| | mov h, a | Сохраняем его в регистре H |
| | ldax d | Загружаем **второй (старший)** байт суммы из адреса 0AF2 |
| | adc h | Складываем вторые (старшие) байты числа из регистра H и суммы из аккумулятора с учетом флага переноса Tc |
| | stax d | Сохраняем результат по адресу 0AF2 |
| | inx b | Переходим к нулевому (младшему) байту следующего числа |
| | mvi a, 2D | Проверяем, просуммированы ли все числа |
| | sub c | |
| | jnz Label43 | |
| | | **Выполнение операции деления суммы всех чисел на их количество** |
| | lda 0AF2 | Загружаем старший байт результирующей суммы |
| | ani 80 | Проверяем результат на отрицательность |
| | jz Label51 | |
| | mvi a, FF | Сохраняем метку отрицательного результата |
| | sta 0AF3 | |
| | lda 0AF0 | Загружаем 0-й (младший) байт суммы |
| | cma | Получаем прямой код 0-го (младшего) байта путем его инвертирования и прибавления 1 без учета флага переноса Tc |
| | adi 01 | |
| | sta 0AF0 | Сохраняем прямой код 0-го (младшего) байта |
| | lda 0AF1 | Загружаем 1-й байт суммы |
| | cma | Получаем прямой код 1-го байта путем его инвертирования и прибавления флага переноса Tc |
| | aci 00 | |
| | sta 0AF1 | Сохраняем прямой код 1-го байта |
| | lda 0AF2 | Загружаем 2-й (старший) байт суммы |
| | cma | Получаем прямой код 2-го (старшего) байта путем его инвертирования и прибавления флага переноса Tc |
| | aci 00 | |
| | sta 0AF2 | Сохраняем прямой код 2-го (младшего) байта |
| Label51: | | |
| | mvi c, 00 | Очищаем промежуточное делимое перед началом деления |
| | mvi d, 0F | Загружаем делитель |
| | lda 0AF2 | Загружаем 2-й байт делимого |
| | mov e, a | |
| | call Label52 | Вызываем подпрограмму деления |
| | mov a, h | Сохраняем 2-й байт результата |
| | sta 0AF8 | |
| | lda 0AF1 | Загружаем 1-й байт делимого |
| | mov e, a | |
| | call Label52 | Вызываем подпрограмму деления |
| | mov a, h | Сохраняем 1-й байт результата |
| | sta 0AF7 | |
| | lda 0AF0 | Загружаем 0-й байт делимого |
| | mov e, a | |

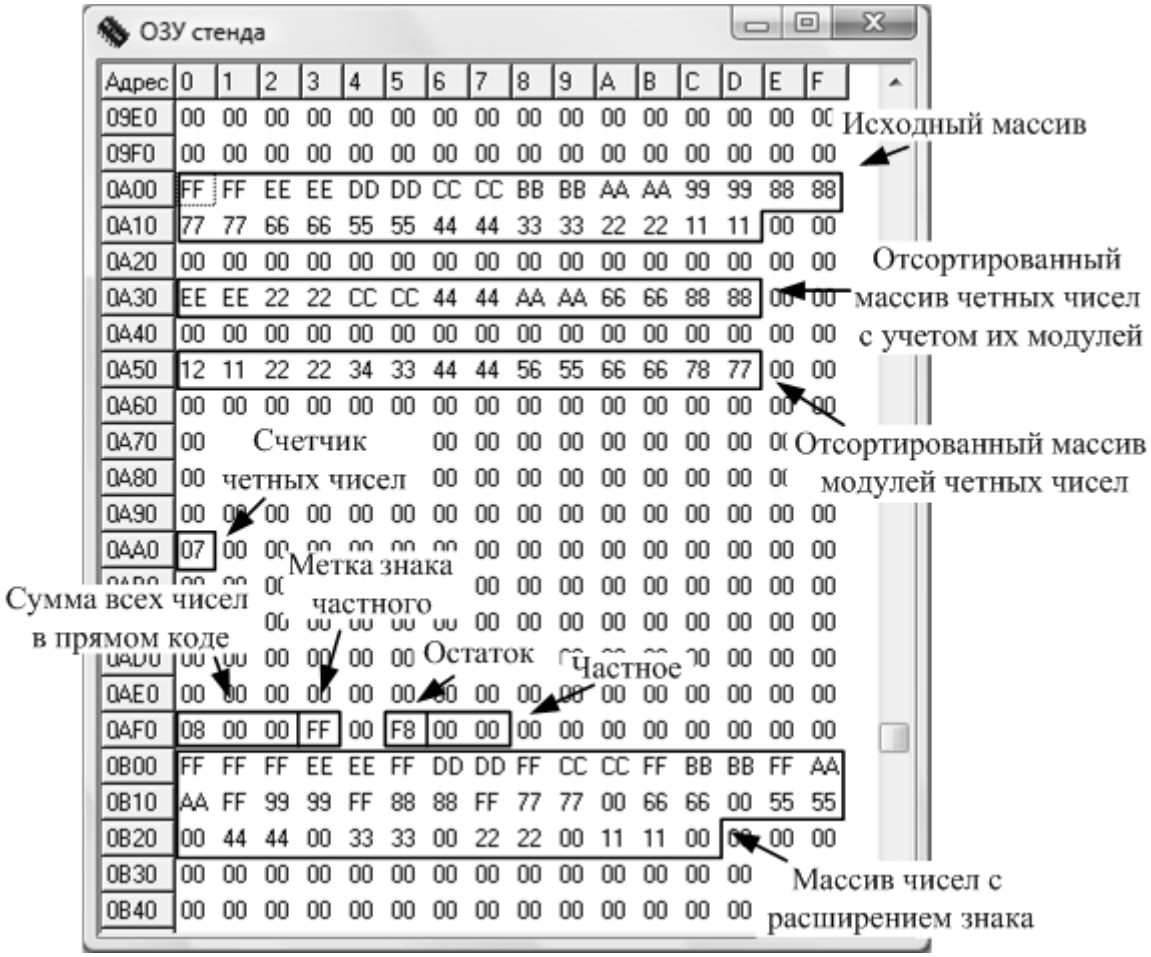
*Продолжение таблицы*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | call Label52 | Вызываем подпрограмму деления |
| | mov a, h | Сохраняем 0-й байт результата |
| | sta 0AF6 | |
| | mov a, c | Сохраняем остаток от деления |
| | sta 0AF5 | |
| Label51: | | |
| | lda 0AF3 | Проверяем наличие метки отрицательного результата и, если ее нет, то прекращаем выполнение программы, если есть – получаем обратный дополнительный код частного |
| | cpi FF | |
| | cnz Label53 | |
| | lda 0AF5 | Загружаем остаток |
| | cma | Получаем прямой код остатка путем его инвертирования и прибавления 1 к результату без учета флага переноса Tc |
| | adi 01 | |
| | sta 0AF5 | Сохраняем результат |
| | lda 0AF6 | Загружаем 0-й (младший) байт частного |
| | cma | Получаем обратный дополнительный код 0-го (младшего) байта частного путем его инвертирования и прибавления 1 без учета флага переноса Tc |
| | adi 01 | |
| | sta 0AF6 | Сохраняем 0-й байт результата |
| | lda 0AF7 | Загружаем 1-й (старший) байт частного |
| | cma | Получаем обратный дополнительный код 1-го (старшего) байта частного путем его инвертирования и прибавления флага переноса Tc |
| | aci 00 | |
| | sta 0AF7 | Сохраняем 1-й (старший) байт результата |
| Label53: | | |
| | rst1 | |
| | | **Блок подпрограмм** |
| Label52: | | Подпрограмма деления |
| | lxi h, 0008 | Загружаем счетчик битов в L и очищаем регистр частного H |
| Label55: | | |
| | mov a, e | Загружаем делимое в аккумулятор |
| | ral | Сдвигаем старший бит делимого в триггер переноса Tc |
| | mov e, a | Сохраняем результат в регистр делимого E |
| | mov a, c | Загружаем в аккумулятор промежуточное делимое из регистра C |
| | ral | Сдвигаем триггер переноса Tc в младший бит промежуточного делимого |
| | sub d | Вычитаем из промежуточного делимого делитель |
| | jnc Label54 | Если делитель больше промежуточного делимого (Tc=1), восстанавливаем промежуточное делимое путем прибавления делителя к разности |
| | add d | |
| Label54: | | |
| | mov c, a | Сохраняем промежуточное делимое в регистр C |
| | cmc | Инвертируем триггер переноса Tc |
| | mov a, h | Загружаем частное в аккумулятор |
| | ral | Сдвигаем триггер переноса Tc в младший бит регистра частного H |
| | mov h, a | Сохраняем результат в регистр частного H |
| | dcr l | Проверяем, отработаны ли все 8 битов |
| | jnz Label55 | |
| | ret | |
| Label34: | | Подпрограмма обмена модулей чисел местами |
| | inx b | |
| | ldax b | |
| | dcx b | |
| | dcx b | |
| | stax b | |

*Продолжение таблицы*

| Метка | Мнемоника | Комментарий |
|---|---|---|
| | inx b | Подпрограмма обмена модулей чисел местами |
| | inx b | |
| | mov a, h | |
| | stax b | |
| | ret | |
| Label35: | | Подпрограмма обмена знаковых значений чисел местами |
| | ldax d | |
| | mov h, a | |
| | inx d | |
| | inx d | |
| | ldax d | |
| | dcx d | |
| | dcx d | |
| | stax d | |
| | inx d | |
| | inx d | |
| | mov a, h | |
| | stax d | |
| | ret | |
| Label21: | | Подпрограмма получения абсолютного значения числа |
| | cpi 00 | Проверка на возникновение отрицательного нуля |
| | jnz Label23 | Если в младшем байте 00, то переходим к проверке старшего |
| | inx b | на число 80, если нет – получаем модуль |
| | ldax b | Загрузка старшего байта числа |
| | cpi 80 | Проверка на возникновение отрицательного нуля |
| | dcx b | Переходим к младшему байту числа |
| | ldax b | Загружаем младший байт числа |
| | jnz Label23 | Если возник отрицательный ноль, заменяем его на 0000. |
| | mvi a, 00 | |
| | stax d | Сохраняем младший байт числа на новый адрес |
| | inx b | Увеличиваем адрес для перехода к старшему байту числа в исходном массиве |
| | inx d | Увеличиваем адрес для перехода к старшему байту числа в массиве модулей чисел |
| | jmp Label24 | Возврат из подпрограммы, если был отрицательный ноль |
| Label23: | | |
| | cma | Инвертирование младшего байта числа |
| | adi 01 | Прибавление 1 к результату |
| | stax d | Пересохраняем младший байт числа на новый адрес |
| | inx b | Увеличиваем адрес для перехода к старшему байту числа в исходном массиве |
| | inx d | Увеличиваем адрес для перехода к старшему байту числа в массиве модулей чисел |
| | ldax b | Загружаем старший байт числа из исходного массива |
| | cma | Инвертирование старшего байта числа |
| | aci 00 | Прибавление флага переноса Tc к результату |
| | jmp Label24 | Возврат из подпрограммы, если отрицательного нуля не было. |

Ниже приведен пример выполнения программы.



# ЗАКЛЮЧЕНИЕ

В результате курсового проектирования должна быть разработана программа на эмуляторе УМПК-80, которая предъявляется для проверки и защиты на CD, флэш-памяти или пересылается на электронную почту преподавателя. При защите проверяется работоспособность программы вначале на том наборе, на котором происходила отладка, а затем на нескольких тестовых массивах, задаваемых преподавателем.

После проверки, доработки и окончательной защиты курсовая работа сдается в электронном виде и в виде твердой копии (помимо титульного листа, присутствуют задание, блок-схема алгоритма, листинг программы с комментариями, результаты проверки работоспособности программы дома и при защите на нескольких тестовых наборах данных, а также список литературы).

Обычно полный объем курсовой работы составляет не более 20–25 листов формата А4.

# СПИСОК ЛИТЕРАТУРЫ

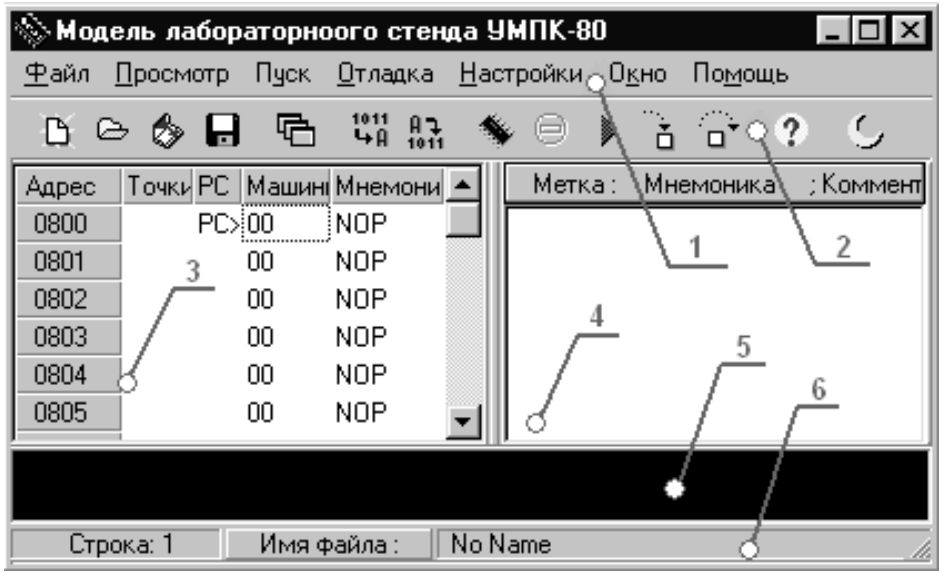
1. Палагута К.А. Микропроцессоры INTEL 8080, 8085 (КР580ВМ80А, КР1821М85А) и их программирование. – М.: МГИУ, 2007. – 104 с.

2. Микропроцессор К580ВМ80 и основы построения микро-ЭВМ с его применением: методические указания / Сост. Палагута К.А. – М.: МГИУ, 2006. – 59 с.

3. Микропроцессоры и интерфейсные средства: методические указания к выполнению курсовой работы / Сост. Кузнецов А.В., Палагута К.А. – М.: МГИУ, 2008. – 95 с.

# ПРИЛОЖЕНИЯ

При запуске исполняемого файла эмулятора УМПК-80 Вы попадаете в главное окно модели, в котором производится основная работа с исследуемой программой:




В состав окна входит:

1. Меню эмулятора
2. Панель инструментов
3. Секция машинных кодов
4. Секция мнемокодов
5. Секция сообщений
6. Строка состояния

Это меню находится в верхней части главного окна модели. Оно содержит все команды, необходимые для работы с моделью. В него входят:

- подменю «Файл», содержит команды для работы с файлами;
- подменю «Просмотр», содержит команды для управления выводом информации;
- подменю «Пуск», содержит команды для выполнения и обработки исследуемой программы;
- подменю «Отладка», содержит команды для отладки исследуемой программы;
- пункт «Настройки», открывает окно настроек модели;

➢ подменю «Окно», содержит команды для управления окнами модели;

➢ подменю «Помощь», содержит команды для вызова справочной информации.

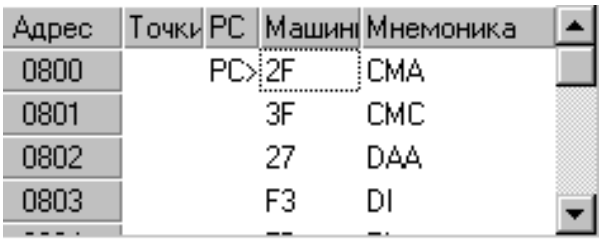
Активизация меню происходит по нажатию клавиши <F10> или щелчком левой кнопки мыши на соответствующем пункте. Также можно выбрать нужный пункт меню сочетанием клавиши <Alt> и клавиши, соответствующей подчеркнутой букве в названии пункта меню. Перемещение между пунктами осуществляется клавишами управления курсором.

Панель инструментов предназначена для быстрого доступа к наиболее часто используемым командам в меню.

| Кнопка | Действие |
|---|---|
|  | Создание нового файла |
|  | Открытие файла |
|  | Закрытие файла |
|  | Сохранение файла на диск |
|  | Открытие всех окон |
|  | Дизассемблирование исследуемой программы |
|  | Ассемблирование исследуемой программы |
|  | Сброс процессора |
|  | Останов выполнения исследуемой программы |
|  | Выполнение исследуемой программы в автоматическом режиме |
|  | Выполнение исследуемой программы по командным циклам |
|  | Выполнение исследуемой программы по машинным циклам |
|  | Вызов справочной информации |
|  | Выход из модели |

Панель инструментов можно отключить (подключить), выбрав в главном меню программы пункт «Просмотр / Панель инструментов». При галочке справа от команды панель инструментов видна.

Секция главного окна модели предназначена для ввода и редактирования исполнительного кода программы в машинных кодах и представляет собой таблицу, каждая строка которой соответствует одной ячейке ОЗУ:




Столбцы таблицы имеют следующее назначение:

| Заголовок столбца | Назначение столбца |
|---|---|
| Адрес | Содержит адреса ячеек памяти |
| Точки останова | Отображает установленные точки останова |
| PC | Отображает счетчик команд в виде «PC>» |
| Машинный код | Содержит значение ячейки памяти по адресу указанному в столбце «адрес» |
| Мнемоника | Содержит мнемоническое описание значения ячейки |

Для занесения значения в ячейку нужно клавишами управления или мышью установить на нее курсор (пунктирная рамка) и набрать новое значение, после чего подтвердить ввод клавишей <enter>. Необходимо заметить, что всегда нужно заносить оба разряда числа, т.е. для ввода значения, например, «7» или «F» нужно соответственно набрать «07», «0F».

В секции машинных кодов присутствует локальное меню – меню секции машинных кодов, которое вызывается щелчком правой кнопки мыши. Оно содержит дополнительные команды редактирования программы в hex-кодах.

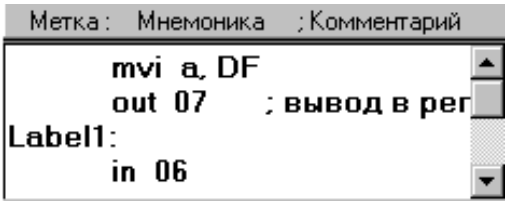
Для отладки программы удобно использовать точки останова, т.е. точки, в которых происходит остановка выполнения программы. Для ее установки нужно щелкнуть левой кнопкой мыши на пересечении столбца «точки останова» и строки с нужным адресом ячейки. Точку останова также можно установить, выбрав пункт «отладка/точка останова» в меню модели или нажать клавиши <Ctrl+F8>. При этом точка будет поставлена в текущей позиции курсора секции машинных кодов. Для удаления одной точки останова необходимо повторить вышеописанные действия. Для снятия всех точек используйте пункт меню «отладка/удалить все точки».

Дополнительные возможности:

➢ Для быстрого перехода к ячейке, на которую указывает счетчик команд, щелкните левой кнопкой мыши в столбце «PC».

➢ Для быстрой установки счетчика команд на нужную ячейку щелкните два раза на строке с этой ячейкой в столбце «адрес».

Секция главного окна модели предназначена для ввода и редактирования исполнительного кода программы в мнемонических кодах и представляет собой обычный текстовый редактор:




Перемещения внутри редактора осуществляются клавишами управления курсором.

Секция имеет локальное меню секции мнемокодов, которое вызывается щелчком правой кнопки мыши. Оно содержит дополнительные команды редактирования программы в мнемонических кодах.

При вводе программы необходимо соблюдать следующий порядок: сначала пишется имя метки с двоеточием в конце, затем команда и далее комментарий. Допускается отсутствие в строке любой из этих частей.

Под меткой понимается расположенная в одной строке последовательность символов, заканчивающаяся двоеточием. В состав имени могут входить прописные и строчные буквы латинского алфавита, цифры и символ подчеркивания.

Комментарием считается текст от символа точки с запятой до конца строки.

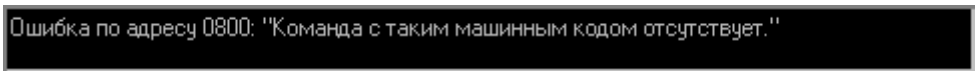
Примечание:

➢ В каждой строке может быть только одна команда.

➢ В командах и именах меток РЕГИСТР БУКВ ЗНАЧЕНИЯ НЕ ИМЕЕТ.

➢ Для ввода символа табуляции нужно использовать сочетание клавиш <Ctrl+Tab>.

В секцию сообщений выводятся сообщения об ошибках, возникающих в результате ассемблирования, дизассемблирования и выполнения программы:




Перейти в нее можно с помощью клавиши <Tab> или щелчком левой кнопки мыши. Установив курсор на одно из сообщений, можно перейти на строку с ошибкой. Для этого нужно дважды щелкнуть левой кнопкой мыши на соответствующем сообщении или с помощью меню секции сообщений, вызвав его щелчком правой кнопкой мыши. Секцию сообщений можно убрать, выбрав пункт «закрыть» вышеуказанного меню, или через меню модели (пункт «просмотр/ просмотр сообщений»).

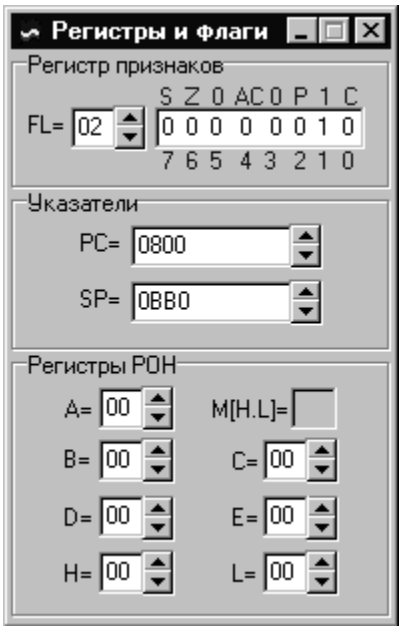
Строка состояния находится в нижней части главного окна модели. Она состоит из двух частей. В левой ее части отображается текущая позиция курсора в секции мнемокодов, в правой – имя текущего файла.


Наибольший интерес в процессе разработки программного кода представляет подменю «просмотр», позволяющее отобразить содержимое регистров и флага, просмотреть состояние памяти, стека, портов стенда.

## Регистры и флаги

Окно «регистры и флаги» вызывается командой «просмотр/регистры и флаги» в главном меню модели или сочетанием клавиш <Ctrl+R>. В нем находятся регистры микропроцессора К580ВМ80. Окно разбито на три части – секции: «регистр признаков», «указатели», «регистры РОН» (рис. П1).

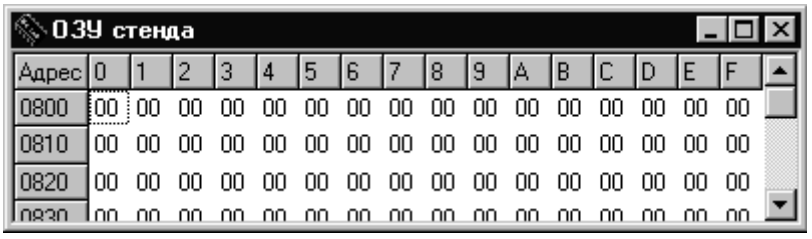
Перемещения по секциям и их элементам осуществляется клавишами <Tab> (вперед) и <Shift+Tab> (назад) или с помощью мыши. Если к полю ввода подвести и задержать на несколько секунд указатель мыши, то в появившейся подсказке будут находиться десятичное и двоичное представления значения этого поля.




| Секция | Назначение секции |
|---|---|
| Регистр признаков | В ней находится регистр признаков МП, в котором содержатся пять флагов: S, Z, AC, P, C. С правой стороны в секции находится шестнадцатеричное значение регистра признаков, а с левой – его побитовое представление. Установка нового значения производится в поле ввода «FL» |
| Указатели | Здесь находятся указатель стека SP и счетчик команд PC. При вводе их новых значений необходимо иметь в виду, что в поле ввода нельзя набрать более четырех символов, а также новое значение должно лежать в диапазоне адресов:– для счетчика команд [ 0800h – (SP-1) ];– для указателя стека [ (PC+1) – 0BB0h ]. При нарушении этого условия восстановится значение, которое было до начала редактирования поля ввода |
| Регистры РОН | В этой секции содержатся: аккумулятор (A), шесть регистров общего назначения (B, C, D, E, H, L), а также значение ячейки по адресу, содержащемуся в регистровой паре HL, которая не является элементом микропроцессора и находится в секции только для удобства исследования команд МП, использующих косвенную адресацию через регистровую пару HL |

## ОЗУ стенда

Окно «ОЗУ стенда» вызывается командой «просмотр/ОЗУ стенда» в главном меню модели или сочетанием клавиш <Ctrl+M>.

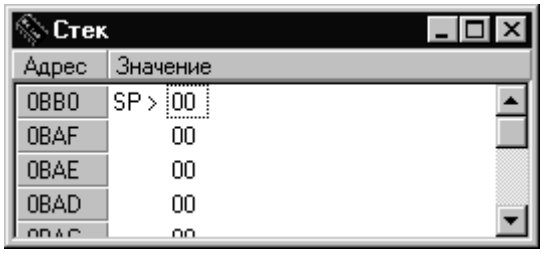



В нем можно просмотреть любой фрагмент памяти. Содержимое памяти отображается в виде таблицы, в каждой строке

которой показывается по шестнадцать байт. Адрес ячейки, лежащей на пересечении строки «0820h» и столбца «D», равен «082Dh». Основной функцией окна является просмотр памяти, однако, при желании можно отредактировать значение любой ячейки. Для этого необходимо клавишами управления курсором или мышью подвести курсор (пунктирная рамка) к ячейке памяти, значение которой Вы хотите исправить, и нажать клавишу <enter> или дважды щелкнуть на ней левой кнопкой мыши. При этом активизируется редактор ОЗУ, в котором нужно ввести новое значение ячейки, после чего редактор надо закрыть.

## Стек

Окно «стек» вызывается командой «просмотр/стек» в главном меню модели или сочетанием клавиш <Ctrl+S>.




В нем можно просмотреть фрагмент памяти, относящийся к стеку, т.е. ячейки с адресами от 0BB0h до адреса, на который указывает указатель стека SP. Содержимое памяти отображается в виде таблицы, в каждой строке которой показывается по одному байту. Таблица имеет три столбца. В первом столбце находятся адреса ячеек, в третьем – их значения. Во втором столбце находится визуальное отображение указателя стека. Основной функцией окна является просмотр памяти, однако, при желании можно отредактировать значение любой ячейки. Для этого необходимо клавишами управления курсором или мышью подвести курсор (пунктирная рамка) к ячейке памяти, значение которой Вы хотите исправить, и нажать клавишу <enter> или дважды щелкнуть на ней левой кнопкой мыши. При этом активизируется редактор ОЗУ, в котором нужно ввести новое значение ячейки, после чего редактор надо закрыть.

## Порты

Окно «порты» вызывается командой «просмотр/порты» в главном меню модели или сочетанием клавиш <Ctrl+P>.

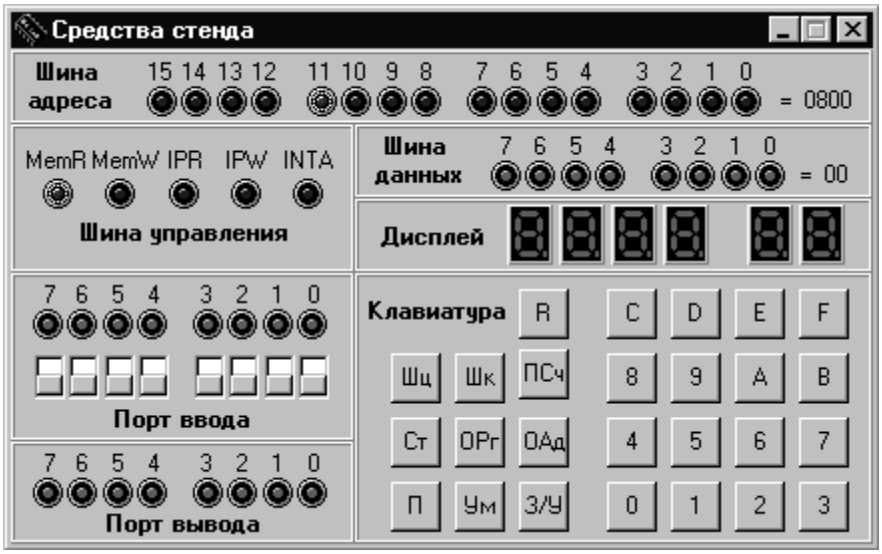

В нем можно просмотреть содержимое портов ввода/вывода. В окне находятся две странички, на одной из которых находятся порты ввода, а на другой – порты вывода. Для перехода на нужную страничку нужно щелкнуть левой кнопкой мыши на закладке с соответствующим названием. Для редактирования значения определенного порта необходимо подвести курсор к ячейке с нужным адресом порта, ввести новое значение и нажать клавишу <enter>. Необходимо иметь в виду, что строка соответствует старшей части адреса порта, а столбец – младшей, т.е. адрес порта, лежащего на пересечении строки «2» и столбца «A», равен «2A».

## Средства стенда

Окно «средства стенда» вызывается командой «просмотр/средства стенда» в главном меню модели или сочетанием клавиш <Ctrl+I>.

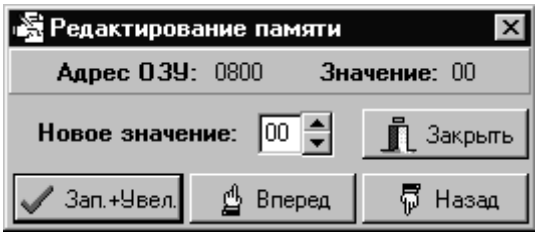
В нем находятся средства, относящиеся к лабораторному стенду, а именно (слева направо, сверху вниз):

| Элемент стенда | Описание элемента |
|---|---|
| Шина адреса | Имитирует светодиодную индикацию шины адреса стенда |
| Шина управления | Имитирует светодиодную индикацию шины данных стенда |
| Шина данных | Имитирует светодиодную индикацию шины управления стенда |
| Дисплей | Имитирует шестиразрядный восьмисегментный дисплей стенда |
| Порт ввода | Имитирует порт ввода с переключателей. Для переключения любого из них необходимо дважды щелкнуть на нем левой кнопкой мыши |
| Клавиатура | Имитирует клавиатуру стенда |
| Порт вывода | Имитирует порт вывода на светодиодные индикаторы |




## Редактор ОЗУ

Окно «редактор ОЗУ» вызывается двойным кликом по ячейке из окон «ОЗУ стенда» и «Стек», так как их основная функция – просмотр памяти.




Редактор предназначен для редактирования памяти. Редактирование производится по одному байту. Адрес текущей ячейки отображается слева от надписи «адрес ОЗУ». Редактор имеет четыре кнопки:

| Кнопка | Описание действий при нажатии на кнопку |
|---|---|
| Зап.+Увел. | Производится запись отредактированного значения ячейки по текущему адресу, после чего текущий адрес увеличивается на единицу |
| Вперед | Текущий адрес увеличивается на единицу без записи нового значения ячейки |
| Назад | Текущий адрес уменьшается на единицу без записи нового значения ячейки |
| Закрыть | Закрывается редактор ОЗУ |