ЦИФРОВЫЕ УСТРОЙСТВА И МИКРОПРОЦЕССОРЫ

ИЗДАТЕЛЬСТВО ТГТУ
УДК 004.383.3(075)
ББК 3844.1я73-5
М822

Р е ц е н з е н т ы :
Кандидат технических наук, доцент
В.В. Орлов

Кандидат технических наук, доцент
В.П. Шелохвостов

С о с т а в и т е л ь
С.П. Москвитин

М822 Цифровые устройства и микропроцессоры : методические указания / сост. С.П. Москвитин. – Тамбов : Изд-во Тамб. гос. техн. ун-та, 2008. – 32 с. – 50 экз.

Представлены варианты заданий на курсовой проект и методические указания к его выполнению с примерами основных приёмов программирования и необходимыми приложениями.

Предназначены для студентов, обучающихся по направлению 210300 «Радиотехника» и изучающих дисциплину «Цифровые устройства и микропроцессоры» дневной и заочной форм обучения

УДК 004.383.3(075)
ББК 3844.1я73-5

ГОУ ВПО «Тамбовский государственный технический университет»

ЦИФРОВЫЕ УСТРОЙСТВА И МИКРОПРОЦЕССОРЫ

Методические указания
для студентов, обучающихся по направлению 210300
«Радиотехника» и изучающих дисциплину «Цифровые устройства и микропроцессоры» дневной и заочной форм обучения

Тамбов
♦ Издательство ТГТУ ♦
2008

Учебное издание

ЦИФРОВЫЕ УСТРОЙСТВА И МИКРОПРОЦЕССОРЫ

Методические указания

Составитель
МОСКВИТИН Сергей Петрович

Редактор О.М. Гурьянова
Инженер по компьютерному макетированию М.А. Филатова

Подписано в печать 24.12.2008.
Формат 60 × 84/16. 1,86 усл. печ. л. Тираж 50 экз. Заказ № 597.

Издательско-полиграфический центр
Тамбовского государственного технического университета
392000, Тамбов, Советская, 106, к. 14

ВВЕДЕНИЕ

Дисциплина «Цифровые устройства и микропроцессоры» охватывает широкий круг схемотехнических вопросов проектирования функциональных элементов и узлов современных радиотехнических систем. Большое внимание при этом уделяется вопросам разработки устройств на современной элементной базе.

В результате изучения дисциплины «Цифровые устройства и микропроцессоры» студенты должны получить знания в области современной элементной базы цифровых и микропроцессорных устройств, активно овладеть методикой проектирования аппаратных и программных средств микропроцессорных устройств.

Изучение этой дисциплины позволит студенту по техническому заданию проектировать микроконтроллеры на современных микропроцессорных больших интегральных схемах и составлять программы на языке АССЕМБЛЕР, а также иметь представление о путях развития современной микроэлектроники.

ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ

Объектом курсового проекта является микропроцессорное устройство обработки информации (МПУОИ) на базе однокристального МП комплекта серии КР580 (аналог фирмы INTEL i8080), реализующее заданную функцию обработки радиотехнических сигналов.

Варианты задания и исходные данные для проектирования МПУОИ выдаются преподавателем.

УКАЗАНИЯ ПО ОФОРМЛЕНИЮ КУРСОВОГО ПРОЕКТА

Курсовой проект должен содержать следующие разделы:

- анализ задания курсового проекта;
- составление электрической структурной схемы устройства;
- составление электрической функциональной схемы устройства;
- составление алгоритма решения поставленной задачи;
- разработка программы для микроконтроллера;
- заключение и список литературы.

Графическая часть курсовой работы должна включать структурную и функциональную схемы устройства, а также электрическую принципиальную схему включения микроконтроллера в измерительную цепь. Оформление пояснительной записки и графической части должно соответствовать требованиям СТП последнего года издания. Исходный текст программы должен быть написан с пояснениями каждого из действий. Рекомендуемый объём пояснительной записки составляет 20 листов.

При вычерчивании блок-схемы алгоритма следует использовать графические символы согласно ГОСТ 19.003–80, представленные в табл. 1.

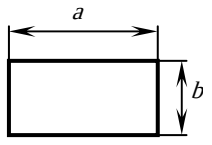
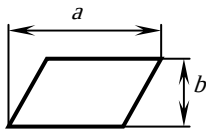
Структура блок-схем, а также содержание записей, помещаемых внутри блоков или рядом с ними, не регламентируется. Но при этом необходимо придерживаться следующих правил и рекомендаций (ГОСТ 19.002–80):

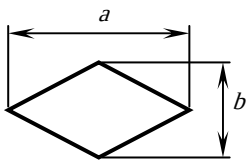
1. Записи в блок-схемах должны быть ясными и однозначно определять, какой этап решения задачи данный блок отражает.
2. Номер блока ставится слева вверху в разрыве контура символа.
3. Нормативным направлением линий считается направление сверху вниз и слева направо.
4. Линии потока проводятся только по вертикали и горизонтали и к осевым линиям символов. Для линий, связывающих два элемента блок-схемы, допускается не более трёх операторов.
5. Линии потока можно прерывать. Для указания связи между прерываниями используется символ «соединитель».

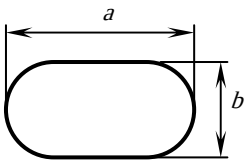
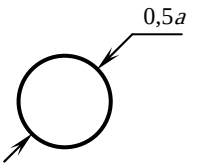
При составлении блок-схемы алгоритма заданный алгоритм решения задачи представляется графически в виде отдельных геометрических фигур (табл. 1), взаимное расположение которых отображает последовательность решения задач на проектируемом МПУОИ. Составленная блок-схема алгоритма должка сопровождаться подробным описанием действий, выполняемых в каждом блоке.

Для составления блок-схемы алгоритма необходимо изучить особенности системы команд того микропроцессорного комплекта (МПК), на базе которого строится МПУОИ. В частности для курсового проекта на базе МПК КР580 необходимо изучить структуру микропроцессора

Таблица 1

Наименование	Обозначения и размеры (a/b = 1/1,5, где a – ширина, b – высота)	Функция
1. Процесс		Выполнение операции (группы операций), в результате которой изменяется значение, форма представления или расположение данных
2. Ввод/вывод		Преобразование данных в форму, пригодную для обработки (ввод), или отображение результатов обработки (вывод)

Наименование	Обозначения и размеры (a/b = 1/1,5, где a – ширина, b – высота)	Функция
3. Решение		Выбор направления выполнения алгоритма или программы в зависимости от некоторых переменных условий

4. Пуск/остановка		Начало, конец, прерывание процесса обработки данных или выполнения программы
5. Соединитель		Указание связи между прерванными линиями потока, связывающие символы

КР580ВМ80 и его систему команд, а также директивы ассемблера. Система команд на ассемблере приведена в прил. 1, а система команд в машинных кодах (16-ричные коды) – в прил. 2 (перевод команд ассемблера в команды в машинных кодах).

Из существующих способов написания программы решения задачи – на машинном языке, на языке ассемблера и на языках высокого уровня – в курсовом проекте необходимо представить программу на языке ассемблера и на машинном языке (в машинных кодах). Процедура перевода программы с языка ассемблера в машинные коды изложена в прил. 2.

Рекомендуется использовать стандартные приёмы программирования: организация счёта, временной задержки, сложения и вычитания, умножения и деления и другие, описания которых даны ниже. Стандартные участки необходимо оформить в виде подпрограмм.

Разработка программы должна сопровождаться пояснениями (комментариями), которые пишутся в поле комментария ассемблерной строки, но игнорируются ассемблирующей программой при трансляции в машинные коды. Ниже рассматривается пример выполнения курсового проекта с написанием программы на языке ассемблера и в машинных кодах.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ОСНОВНЫМ ПРИЁМАМ ПРОГРАММИРОВАНИЯ

1. ОСНОВНЫЕ ДИРЕКТИВЫ АССЕМБЛЕРА

Директивы ассемблера – это указания ассемблирующей программе о выполнении определённых действий в процессе ассемблирования. Они не являются командами и не переводятся в машинный код. Операторы директив необязательны.

1. **Директива ORG** (организовать). Эта директива определяет ячейку памяти, куда будет загружаться первый байт следующей команды или байт данных. Этот адрес указывается в поле операнда.

Если в начале программы директива ORG отсутствует, то по умолчанию подразумевается директива ORG с нулевым адресом (таким образом, в примере курсового проекта директива ORG необязательна).

При необходимости в программе может быть несколько директив ORG. До новой директивы ORG команды и данные размещаются в смежных ячейках памяти.

2. **Директива END** (конец). Эта директива информирует ассемблирующую программу о достижении конца программы. В каждой программе может быть только одна директива END, находящаяся в последней строке.

3. **Директива EQU** (приравнять, присвоить). При выполнении директивы EQU ассемблирующая программа присваивает значение выражения, находящегося в поле операнда символическому наименованию, находящемуся в поле метки. Когда наименование встречается в поле операнда, ассемблирующая программа подставляет вместо него присвоенное значение.

Символическое наименование может появиться в поле метки только одной директивы EQU. При программировании следует сгруппировать все директивы EQU в начале или конце программы.

2. ОРГАНИЗАЦИЯ СЧЁТА

Операция счёта выполняется с помощью некоторого регистра общего назначения, в который загружается константа, равная заданному коэффициенту пересчёта N. Затем из содержимого этого регистра вычитаются единицы до получения нулевого результата, после чего следует условный переход по нулю. При отсутствии свободного регистра в качестве счётчика можно использовать ячейку ОЗУ.

Обычно операция счёта связана с созданием временной задержки, с какими-либо внешними событиями или с циклическим участком программы, когда некоторая основная функция повторяется несколько раз. Здесь возможны два способа организации счёта: в первом случае сначала выполняется основная функция, а затем – счёт и проверка состояния счётчика; во втором – наоборот. Соответственно, обеспечиваются следующие пределы изменения коэффициента пересчёта: $1 \le N \le 2^n$ и $0 \le N \le 2^{n-1}$. Так как разрядность регистра равна 8, то максимальная ёмкость счётчика, выполненного на одном регистре, – 256. Если требуется обеспечить $N > 256$, то следует организовать совместную работу двух или нескольких регистров.

3. ОРГАНИЗАЦИЯ ВРЕМЕННОЙ ЗАДЕРЖКИ

Функция временной задержки или генерации временных интервалов заданной длительности часто необходима для организации взаимодействия микропроцессора (МП) с внешними устройствами. Так как тактовая частота МП обычно стабили-

зирована, то задержка может осуществляться с большой точностью. Длительность такта – 0,5 мкс при работе МП на частоте 2 МГц.

Для реализации малых задержек можно использовать приведённые в табл. 4 команды, выполнение которых не изменяет содержимого регистров и элементов памяти МПК (за исключением содержимого счётчика команд).

С помощью команд из табл. 2 можно обеспечить задержку на любое число тактов (кроме шести), начиная с четырёх.

Однако задержки большой длительности целесообразно получать путем организации счёта, т.е. циклических участков в программе. В этом случае задача сводится к выбору числа каскадов счётчика, определению требуемых коэффициентов пересчёта и к введению дополнительных малых задержек для обеспечения точного значения заданного времени задержки.

Таблица 2

Команда	Число тактов	Число байтов в формате команды	Название команды и примечание
NOP	4	1	Пустая операция
MOV A, A	5	1	Пересылка (A) ← (A)
ADI 00H	7	2	Сложение A с нулём (может повлиять на флаги!)
XTHL XTHL	18 18	1 1	Двукратный обмен вершины стека с H-L – задержка на 36 тактов
PUSH B POP B	11 10	1 1	Запись в стек с последующим считыванием – задержка на 21 такт

На базе однокаскадного счётчика целесообразно получать задержки длительностью до десятка миллисекунд. Для получения более длительных задержек число каскадов счётчика увеличивается программным путём (см. табл. 2).

4. СЛОЖЕНИЕ И ВЫЧИТАНИЕ ЧИСЕЛ

Сначала рассмотрим выполнение этих операций над числами без знаков, что соответствует, например, операциям над кодами и положительными числами.

Если операнды являются однобайтовыми числами, то их сложение и вычитание осуществляется командами ADD R и SUB R. При выполнении команды сложения возможно переполнение разрядной сетки, о чём свидетельствует значение флаги переноса C. Особенностью выполнения команды вычитания является то, что её результат представлен в дополнительном коде. Напомним, что при положительном числе дополнительный код совпадает с прямым, а при отрицательном он равен дополнению данного числа до 2^n. Для двоичных отрицательных чисел дополнительный код вычисляют путём инверсии всех разрядов прямого кода с последующим добавлением единицы.

Если прямой код числа содержится в аккумуляторе, то переход к дополнительному коду осуществляется с помощью команд инверсии CMA и инкремента (сложения с единицей) INR A. Обратный переход от дополнительного кода числа к прямому осуществляют аналогичным образом – с помощью команд декремента (вычитания единицы) DCR A и инверсии CMA.

Если хотя бы один из операндов занимает более n разрядов (где n – разрядность МП), то говорят, что вычисления производятся с повышенной точностью. В МП серии КР580 для сложения двухбайтовых слов предусмотрена команда DAD RP. В общем же случае сложение и вычитание многобайтовых слов осуществляется путём многократного использования команд ADC R и SBB R.

О наличии переполнения разрядной сетки, которое может произойти при сложении как двух положительных, так и двух отрицательных чисел, свидетельствует факт неравнозначности значений флагов переноса и знака. Если же складываются числа разных знаков, то переполнение не может произойти, а значение флага переноса игнорируется.

5. УМНОЖЕНИЕ И ДЕЛЕНИЕ

Умножение чисел $\alpha\beta = \gamma$ реализуется с помощью команд суммирования и сдвига. Для получения γ к сумме частичных произведений $\sum$ добавляется α, если очередной разряд $\beta = 1$, после чего осуществляется сдвиг $\sum$; указанная процедура повторяется для каждого разряда β. При этом анализ β можно начинать со старших разрядов и соответственно сдвигать влево, либо наоборот.

Знак γ обычно формируют независимо от описанной процедуры умножения на основе анализа знаков сомножителей, а при осуществлении умножения используют прямые коды сомножителей без знака. В этом случае необходимо учитывать, что освобождающиеся разряды при сдвигах должны заполняться нулями.

Деление двоичных чисел основывается на последовательном вычитании делителя из разрядов делимого и остатка от деления. Таким образом, деление реализуется с помощью операций сдвига, вычитания и анализа результата вычитания. По аналогии с умножением здесь целесообразно осуществлять деление чисел без знаков, а знак частного определять отдельно на основе анализа знаков операндов. Перед выполнением деления необходимо убедиться, что делитель не равен нулю.

Рассмотренные варианты выполнения арифметических действий с помощью МП подразумевают представление чисел в форме с фиксированной запятой (точкой).

6. ОРГАНИЗАЦИЯ ОЧЕРЕДЕЙ, СТЕКОВ И ПОДПРОГРАММ

6.1. Очередь

Очередь представляет собой структуру данных, в которой элементы можно исключать только с одного конца (начало очереди), а включать только с другого (конец очереди). Главная особенность очереди заключается в том, что она содержит порядок элементов неизменным – принцип FIFO (First In, First Out – первый приходит, первый уходит). Число элементов в очереди называется его длиной.

При реализации очереди её элементы остаются неподвижными, а вводятся два указателя для начала и конца очереди. Указатель начала адресует элемент, подлежащий исключению, а указатель конца очереди адресует ячейку сразу за последним элементом. Очередь обычно используется при вводе и выводе символьных данных, а для их организации – область смежных ячеек памяти, число которых определяется максимально возможной длиной очереди.

Чтобы исключить элемент очереди, необходимо произвести считывание элемента, адресуемого указателем начала очереди, и увеличить этот указатель на 1 (очередь растёт в область больших адресов).

Включаемый элемент записывается в первую свободную ячейку и указатель очереди увеличивается на 1.

Возможны два особых случая: переполнение – включение в очередь элемента, когда все ячейки заняты, и антипереполнение – исключение элемента из пустой очереди.

6.2. Стек

Стек, как и очередь, представляет собой специальную разновидность одномерного массива. Он является специальной областью ОЗУ, организованную таким образом, что загрузку элементов в стек и извлечение их из стека можно осуществлять только с одного конца, называемого вершиной стека. Таким образом, в любой момент времени из стека можно произвести считывание только элемента, находящегося в его вершине и представляющего собой последние загруженные данные. Ячейку, выполняющую роль вершины стека, адресует указатель стека SP. Таким образом, стек работает по принципу LIFO (Last In, First Out – последний приходит, первый уходит).

6.3. Подпрограммы

Часто в программах некоторые последовательности команд, выполняющие определённую функцию, встречаются несколько раз, но оперируют различными данными. Такие последовательности можно оформить в *подпрограмму* и включить в программу один раз, а в нужных точках вызывать её для выполнения с текущими данными.

Вызов подпрограммы нарушает естественный порядок следования команд. Из точки вызова управление следует передать первой команде подпрограммы, затем обычным образом выполнить команды подпрограммы, а по её завершении управление необходимо передать (возвратить) в точку вызова. Инициирование выполнения подпрограммы осуществляется путём указания её имени в одной из команд вызова подпрограммы. Имя подпрограммы должно находиться в поле метки первой её команды. При выполнении команды вызова подпрограммы текущее содержание счётчика команд (адрес возврата) загружается в стек, а в счётчик команд загружается адрес перехода – второй и третий байт команды вызова подпрограммы. После этого начинает выполняться первая команда подпрограммы, затем вторая и т.д. Заключительной командой каждой подпрограммы должна быть команда возврата из подпрограммы, которая извлекает адрес возврата из стека и передаёт его в счётчик команд. Следовательно, теперь будет выполняться команда, находящаяся в вызывающей программе сразу после команды вызова подпрограммы.

Правильно составленная подпрограмма не должна изменять содержимого ни одного регистра, которое после выполнения подпрограммы потребуется вызывающей программе. Таким образом, содержимое регистров, используемых подпрограммой, необходимо временно запоминать, для того, чтобы по завершении подпрограммы его можно было восстановить без изменений. Наиболее удобной памятью для временного запоминания содержимого регистров является стек. Запомнить содержимое регистров можно в вызывающей программе перед вызовом подпрограммы либо в самой подпрограмме. Практика программирования показывает, что целесообразно всегда запоминать содержимое регистров в начале и восстанавливать его в конце подпрограмм.

При организации подпрограмм важно выбрать правильный способ передачи параметров подпрограмме. В подпрограмме умножения в курсовом проекте использована передача параметров во внутренних регистрах микропроцессора. При этом способе перед вызовом подпрограммы параметры, используемые в подпрограмме, загружаются во внутренние регистры. После вызова подпрограмма выполняет необходимые действия и загружает результат в определённый регистр (регистры) микропроцессора. Основной недостаток данного способа заключается в жёстких ограничениях на число передаваемых параметров.

Подпрограммы могут, например, использоваться для выполнения определённых арифметических действий: сложения и вычитания, умножения и деления и т.д.

Подпрограмма умножения, используемая в курсовом проекте, имеет вид (табл. 3).

3. Пример программы (подпрограмма MULT)

Метка	Код	Операнд	Комментарий
MULT	LXI	H, 0000H	Сброс регистра произведения
	NVI	B, 08H	Инициализация счётчика бит
M4:	DAD	H	Сдвиг частичной суммы влево
	RLC		Сдвиг множителя с переносом
	JNC	M5	Анализ бита множителя
	DAD	D	Суммирование множимого
M5:	DCR	B	Декремент счётчика бит
	JNZ	M4	Умножение на следующий бит
	RET		Умножение закончено, возврат

Эта подпрограмма предназначена для умножения 8-битных целых чисел без знака. Она реализует анализ отдельных цифр множителя с последующим накапливающим суммированием множимого и сдвигом частичной суммы или множимого. Умножение выполняется старшими разрядами вперёд. Сдвиг частичной суммы в регистрах H, L осуществляется командой удвоения DAD H.

Перед выполнением подпрограммы один множитель помещается в аккумулятор, второй – в регистр E. Результат будет получен в регистровой паре H, L. Команда LXI H, 0000H обнуляет содержимое регистровой пары H, L, чтобы предварительно записанные в этих регистрах данные не влияли на результат вычисления подпрограммы. Команда MVI B, 08H служит для начальной установки счётчика циклов (или счётчика бит). В ходе подпрограммы осуществляется умножение одного множимого целиком на каждый бит другого множителя. Как уже было указано, оба числа – 8-битные, поэтому требуется 8 циклов. Команда DAD H осуществляет удвоение частичной суммы, а команда RLC – удвоение содержимого аккумулятора (множителя). Команда JNZ анализирует содержимое следующего разряда. Если этот разряд равен 1, то множимое (в регистровой паре D, E) добавляется к частичной сумме. Если этот разряд равен нулю, суммирование пропускается. Команда DCR B осуществляет счёт циклов. Если вычисления не закончены, осуществляется условный переход на следующий цикл (командой JNZ M4). По окончании вычислений осуществляется возврат в основную программу по команде RET.

ПРИМЕР ВЫПОЛНЕНИЯ КУРСОВОГО ПРОЕКТА

1. ОБЩАЯ ПОСТАНОВКА ЗАДАЧИ

Пусть алгоритм работы устройства задан уравнением

$$V(t) = kx^2(t) + y(t - N),$$

где k – заданная константа, например, $k = 2{,}5$; $x(t)$ – сигнал на одном входе (порт X – 1 байт); $y(t - N)$ – сигнал, поступающий по второму входу и отстающий от $x(t)$ на N=10 тактов (порт Y – 1 байт); $V(t)$ – выходной сигнал, выводится через порт V (2 байта). Для хранения коэффициента k можно отвести ячейку ОЗУ или формировать этот коэффициент в самой программе, что обычно предпочтительней.

Для хранения N слов, считанных в моменты времени, следующие за t_i после поступления $y(t_i - N)$, целесообразно организовать очередь. Внутри ОЗУ выделяется некоторая область, выполняющая функции стека. Как обычно, регистровая пара HL будет адресным индексным регистром при косвенной адресации ПЗУ и ОЗУ. Различие в ПЗУ и ОЗУ при обращении к памяти будет заключаться в установке адресной линии A15 в 1 или 0. Например, при A15 = 0 производится выборка из ПЗУ, а при A15 = 1 – обращение к ОЗУ.

Если в проектируемом устройстве содержится не более восьми портов для подключения внешних устройств ввода/вывода, то каждому порту выделяется индивидуальная адресная линия, что соответствует использованию кода «1 из 8-ми» для кодирования портов ввода/вывода:

PORT 1 – 00000001 (01H)	PORT 5 – 00000001 (10H=16)
PORT 2 – 00000010 (02H)	PORT 6 – 00000001 (20H=32)
PORT 3 – 00000100 (04H)	PORT 7 – 00000001 (40H=64)
PORT 4 – 00001000 (08H)	PORT 8 – 00000001 (80H=128)

2. РАСПРЕДЕЛЕНИЕ ПАМЯТИ

Весь объём адресуемой памяти (64К) с адресами 0000H...FFFFH можно распределить следующим образом. Для основной программы будем использовать адреса, начиная с 0000H до FFFFH. Для инициализации портов ввода/вывода зададим их адреса: порт ввода переменной $x(t)$ – 01H; порт ввода $y(t - N)$ – 02H, порт вывода $V(t)$ – 04H (применяется код «1 из 8-ми»).

3. СОСТАВЛЕНИЕ СТРУКТУРНОЙ СХЕМЫ АЛГОРИТМА РЕШЕНИЯ ЗАДАЧИ И ПРОГРАММЫ

В соответствии со словесным описанием составляем структурную схему алгоритма, на основании которой будет разрабатываться программа.

Следует придерживаться естественного хода решения, т.е. линейной структуры с необходимыми метками и циклами (рис. 1).

Программы на ассемблере и в машинных кодах, соответствующие этому алгоритму, приведены в табл. 4.

В приведённом алгоритме решения задачи константа k формируется программным путём, с помощью сдвигов влево, вправо и суммирования результатов константа N, а также остальные константы записываются в 16-ричном коде.

Следует обратить внимание, что приступать к выполнению курсового проекта имеет смысл только после изучения теоретических вопросов, посвящённых программному обеспечению (программированию на языке ассемблера и в машинных кодах). При этом нужно ориентироваться на рекомендуемую учебную литературу.

Блок начальных директив не обязателен. Он обычно вводится для удобства программирования. Адреса портов – 01H, 02H, 04H, 08H и т.д. Таким образом, для каждого порта выделяется отдельная линия.

Стек и очередь требуются в том случае, когда одна из переменных зависит от $t - N$. Если все переменные зависят от t, стек и очередь не нужны.

При наличии переменной, зависящей от $t - N$, сначала вводятся все значения этой переменной при $t \ldots t - N$ в виде очереди. Затем данные из очереди пересылаются в стек. При этом последним элементом оказывается значение переменной при $t - N$, которое извлекается первым и используется в вычислениях.

Организация стека и очереди является подготовкой для ввода соответствующей переменной, зависящей от $t - N$ (в примере – $y(t - N)$). В команде LXI H, 800AH младший байт операнда равен значению N (10 = = 0AH). Значение константы N, как и других констант, должно быть представлено в 16-ричной системе.

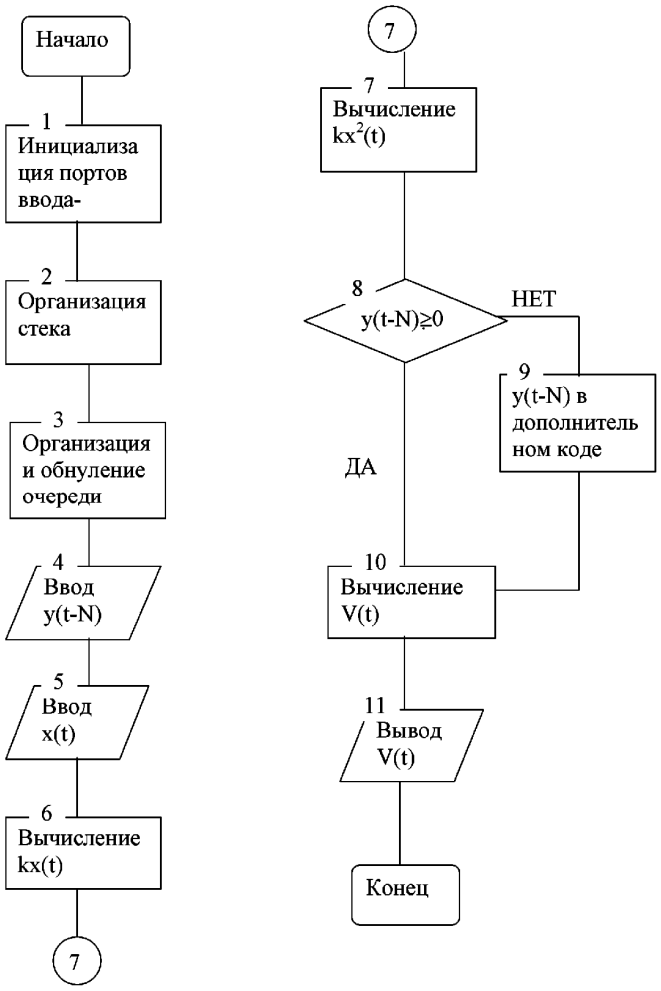


Рис. 1. Блок-схема алгоритма

Команды NOP используются для согласования быстродействующего микропроцессора с медленным устройством ввода. Так как микросхемы интерфейса ввода/вывода и периферийные устройства не заданы, количество этих команд не принципиально.

Умножение переменной на константу выполняется с помощью операций циклического сдвига RLC (умножение на 2) и RRC (умножение на 0,5). После каждой операции требуется команда ANA – сброс флага переноса, так как команды циклического сдвига влияют на этот флаг. Затем полученные результаты суммируются соответствующим образом.

4. Программа вычислений на языке ассемблера в машинных кодах

Адрес	Машинный код	Метка	Мнемо-ника	Операнд	Комментарии
		Блок начальных директив			
			ORG	0000H	Директива начального адреса
		PORTX	EQU	01H	Инициализация портов ввода переменных $x(t)$, $y(t)$, $y(t - N)$ и вывода $V(t)$
		PORTY	EQU	02H	
		PORTV	EQU	04H	
		Организация стека и очереди			
0000	31 00 84	M0:	LXI	SP,8400H	Организация стека в ОЗУ
0003	21 0A 80		LXI	H,800AH	Загрузка адреса указателя начала очереди в ОЗУ (N= 0AH)
0006	AF	M1:	XRA	A	Обнуление аккумулятора
0007	77		MOV	M,A	Обнуление ячейки очереди (адр. HL)
0008	3E 01		MVI	A,01H	Номер ячейки конца очереди: 8001H; счёт ячеек в регистре L
000A	2D		DCR	L	Адрес следующей ячейки
000B	BD		CMP	L	$(A) - (L) = 0$?
000C	C2 06 00		JNZ	M1	Переход на метку $M1$, если это не последняя ячейка очереди

000F	21 0A 80		LXI	H,800AH	Повторная загрузка указателя начала очереди
		Ввод переменной $y(t-N)$			
0012	AF	LINE:	XRA	A	Подготовка к вводу переменной
0013	DB 02		IN	PORTY	Ввод $y(t-N)$; параметр в аккумуляторе

Адрес	Машинный код	Метка	Мнемо-ника	Операнд	Комментарии
0015	77		MOV	M,A	Данные – в начале очереди
0016	F5		PUSH	PSW	$y(t-N)$ – в стеке для хранения
0017	00		NOP		Пустые операции для увеличения длительности такта ввода
0018	00		NOP		
0019	00		NOP		
001A	00		NOP		
001B	AF		XRA	A	Подготовка к вводу в аккумулятор содержимого последней ячейки очереди
001C	3E 01		MVI	A,01H	Указатель последней ячейки очере- ди – в регистре A
001E	2D		DCR	L	Переход на следующую ячейку
001F	BD		CMP	L	Последняя ли ячейка: (A) – (L) = 0?
0020	C2 12 00		JNZ	LINE	Если результат не нулевой, то переход на LINE
0023	AF		XRA	A	Подготовка к вводу переменной $x(t)$
0024	DB 01		IN	PORTX	$x(t)$ – в аккумуляторе
0026	00		NOP		Пустые операции для увеличения длительности такта ввода
0027	00		NOP		
0028	00		NOP		
0029	00		NOP		
002A	21 00 00		LXI	H, 0000H	Обнуление регистровой пары H, L
002D	16 00		MVI	D, 00H	Обнуление регистров общего назначения D и E
002F	5A		MOV	E, D	
0030	5F		MOV	E, A	Копирование $x(t)$ в регистре E

Адрес	Машинный код	Метка	Мнемо-ника	Операнд	Комментарии
0031	07		RLC		2 $x(t)$ – в аккумуляторе
0032	A7		ANA	A	Сброс флага переноса: CY = 0
0033	57		MOV	D,A	Временное запоминание 2 $x(t)$ в регистре D
0034	AF		XRA	A	Подготовка к записи в аккумулятор $x(t)$
0035	7B		MOV	A,E	$x(t)$ – в аккумуляторе
0036	0F		RRC		Получение 0,5 $x(t)$ в аккумуляторе
0037	A7		ANA	A	Сброс флага переноса: CY = 0
0038	82		ADD	D	(A) + (D) = 2,5 $x(t)$ в аккумуляторе
0039	CD 6C 00		CALL	MULT	Вызов подпрограммы умножения однобайтных слов; множимое: 2,5 $x(t)$ – в регистре A; множитель: $x(t)$ – в регистре E; произведение: 2,5 $x^2(t)$ – в H, L
003C	AF		XRA	A	Подготовка к обнулению РОНов
003D	57		MOV	D,A	Обнуление регистра D
003E	5A		MOV	E,D	Обнуление регистра E
003F	42		MOV	B,D	Обнуление регистров B, C
0040	48		MOV	C,B	
0041	54		MOV	D,H	Запоминание 2,5 $x^2(t)$ в регистровой паре D, E
0042	5D		MOV	E,L	

Продолжение табл. 4

Адрес	Машинный код	Метка	Мнемо-ника	Операнд	Комментарии
			Обработка логического блока		
0043	F1		POP	PSW	$y(t-N)$ – в аккумуляторе
0044	BF		CMP	A	Знак результата – в регистре F
0045	FA 4B 00		JM	M2	При $S = 1$ переход на метку M2
0048	F2 57 00		JP	M3	При $S = 0$ переход на метку M3
004B	2F	M2:	CMA		INV ($y(t-N)$)
004C	3C		INR	A	INV ($y(t-N)$) + 1 – дополнительный код
004D	47		MOV	B,A	Дополнительный код в регистровой паре B, C
004E	09		DAD	B	Сложение двухбайтовых слов. Результат – в H, L
			Вывод результата		
004F	AF		XRA	A	Подготовка к выводу младшего байта
0050	7D		MOV	A,L	
0051	D3 04		OUT	PORTV	Вывод младшего байта
0053	AF		XRA	A	Подготовка к выводу старшего байта
0054	7C		MOV	A,H	
0055	D3 04		OUT	PORTV	Вывод старшего байта
0057	AF	M3:	XRA	A	Процедура вывода младшего байта при $S = 0$
0058	67		MOV	H,A	Обнуление регистровой памяти H, L
0059	6F		MOV	L,A	
005A	EB		XCHG		2,5 $x^2(t)$ из регистровой памяти (D, E) поменять местами с содержимым (H, L). Обнуление регистровой пары D, E

Адрес	Машинный код	Метка	Мнемо-ника	Операнд	Комментарии
005B	3A 0A 80		LDA	800AH	$y(t-N)$ – в аккумуляторе
005E	57		MOV	D,A	$y(t-N)$ – в регистровой паре D, E для суммирования
005F	19		DAD	D	Сложение двухбайтовых слов. Результат – в H, L
0060	AF		XRA	A	Подготовка к выводу младшего байта
0061	7D		MOV	A,L	
0062	D3 04		OUT	PORTV	Вывод младшего байта
0064	AF		XRA	A	Подготовка к выводу старшего байта
0065	7C		MOV	A,H	
0066	D3 04		OUT	PORTV	Вывод старшего байта
0068	C3 00 00		JMP	M0	Безусловный переход к новому циклу вычислений
006B	76		HLT		Останов
006C	21 00 00	MULT:	LXI	H,0000H	Начало подпрограммы умножения. Сброс регистра суммы частичных произведений
006F	06 08		MVI	B,08H	Начальная установка счётчика циклов
0071	29	M4:	DAD	H	Логический сдвиг H, L на разряд влево
0072	07		RLC		Очевидный разряд множителя с переносом
0073	D2 77 00		JNC	M5	Переход на метку M5, если следующей разряд нуль
0076	19		DAD	D	Добавление множимого к частичной сумме

Адрес	Машинный код	Метка	Мнемо-ника	Операнд	Комментарии
0077	05	M5:	DCR	B	Счёт циклов
0078	C2 71 00		JNZ	M4	Переход, если вычисления не закончены
007B	C9		RET		Возврат в основную программу
			END		Директива достижения конца программы

Для перемножения переменных и возведения в квадрат используется специальная подпрограмма умножения, которая приведена в конце программы. Эта подпрограмма умножения более подробно объясняется в табл. 3.

Если заданы три переменные, требуется добавить соответствующие блоки и операции.

Сложение числа в дополнительном коде эквивалентно вычитанию этого числа в прямом коде. Это позволяет заменить вычитание двухбайтных чисел их сложением.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1

СИСТЕМА КОМАНД НА ЯЗЫКЕ АССЕМБЛЕРА

Мнемоника	Описание команд	Условное обозначение	Число байт
	Команды пересылки данных		
MOV R1, R2	Передача из регистра R2 в регистр R1	(R1) ← (R2)	1
MOV M, R	Передача из регистра в память	((H)(L)) ← (R)	1
MOV R, M	Передача из памяти в регистр	(R) ← ((H)(L))	1
MVI R, data	Передача байта в регистр	(R) ← (byte2)	2
MVI M, data	Передача байта в память	((H)(L)) ← (byte2)	2
LXI RP, data 16	Загрузка парных регистров B, C; D, E; H, L; SP	(RL) ← (byte2) (RH) ← (byte3)	3
LDAX B	Загрузка аккумулятора по адресу, указанному парой регистров B, C	(A) ← ((B)(C))	1
LDAX D	То же, парой регистров D, E	(A) ← ((D)(E))	1
STAX B	Занесение содержимого аккумулятора по адресу, указанному парой регистров B, C	((B)(C)) ← (A)	1
STAX D	То же, парой регистров D, E	((D)(E)) ← (A)	1
LDA Addr	Загрузка аккумулятора по адресу, указанному в команде	(A) ← ((byte3)(byte2))	3
STA Addr	Занесение содержимого аккумулятора по адресу, указанному в команде	((byte3)(byte2)) ← (A)	3
LHLD Addr	Загрузка регистров L, H из двух соседних ячеек	(L) ← ((byte3)(byte2)) (H) ← ((byte3)(byte2) + 1)	3

Мнемоника	Описание команд	Условное обозначение	Число байт
SHLD Addr	Занесение содержимого регистров L, H в две соседние ячейки, начиная с адреса, указанного в команде	((byte3)(byte2)) ← (L) ((byte3)(byte2)) ← (H)	3
XCHG	Обмен данными ме-жду H, L и D, E	(H) ↔ (D) (L) ↔ (E)	1
XTHL	Обмен данными ме-жду стеком и H, L	(L) ↔ ((SP)) (H) ↔ ((SP) + 1)	1
SPHL	Занесение содержимого регистра H, L в стек	(SP) ↔ (H)(L)	1
PUSH RP	Ввод содержимого регистров B, C; D, E или H, L в стек	((SP) – 1) ← (RH) ((SP) – 2) ← (RL) (SP) ← (SP) – 2	1
PUSH PSW	Ввод PSW (слово состояния процессо-ра – содержимое аккумулятора и регистра признаков) в стек	((SP) – 1) ← (A) $((SP) - 1)_0 \leftarrow$ (CY) $((SP) - 1)_2 \leftarrow 1$ $((SP) - 1)_2 \leftarrow$ (P) $((SP) - 1)_3 \leftarrow 0$ $((SP) - 1)_4 \leftarrow$ (AC) $((SP) - 1)_5 \leftarrow 0$ $((SP) - 1)_6 \leftarrow$ (Z) $((SP) - 1)_7 \leftarrow$ (S) (SP) ← (SP) – 2	1
POP RP	Выдача данных из стека в регистры B, C; D, E; H, L	(RL) ← ((SP)) (RH) ← ((SP) + 1) (SP) ← (SP) + 2	
POP PSW	Выдача данных из стека в аккумулятор и регистр признаков (слово состояния процессора)	(CY) ← $((SP))_0$ (P) ← $((SP))_2$ (AC) ← $((SP))_4$ (Z) ← $((SP))_6$ (S) ← $((SP))_7$ (A) ← ((SP) + 1) (SP) ← (SP) + 2	1
	Арифметико-логические команды		
ADD R	Сложение содержимого регистра и аккумулятора	(A) ← (A) + (R)	1

Продолжение табл.

Мнемоника	Описание команд	Условное обозначение	Число байт
ADC R	То же, но с учётом переноса	(A) ← (A) + (R) + (CY)	1
ADD M	Сложение содержимого ячейки памяти и аккумулятора	(A) ← (A) + ((H)(L))	1
ADC M	То же, но с учётом переноса	(A) ← (A) + ((H)(L)) + (CY)	1
ADI data	Сложение байта с содержимым аккумулятора	(A) ← (A) + (byte2)	2
ACI data	Сложение байта с содержимым аккумулятора с учётом переноса	(A) ← (A) + (byte2) + (CY)	2
DAD RP	Сложение содержимого пар регистров B, C; D, E; H, L; SP с содержимым пары H, L	(H)(L)←(H)(L) + (RH)(RL)	1
SUB R	Вычитание содержимого регистра из содержимого аккумулятора	(A) ← (A) – (R)	1
SBB R	То же, но с заёмом	(A) ← (A) – (R) – (CY)	1
SUB M	Вычитание содержимого ячейки памяти из содержимого аккумулятора	(A) ← (A) – ((H)(L))	1
SBB M	То же, но с заёмом	(A) ← (A) – ((H)(L)) – (CY)	1
SUI data	Вычитание байт из содержимого аккумулятора	(A) ← (A) – (byte2)	2
SBI	То же, но с учётом заёма	(A) ← (A) – (byte2) – (CY)	2
INR R	Увеличение содержимого регистра на единицу	(R) ← (R) + 1	1
INR M	Увеличение содержимого ячейки памяти на единицу	((H)(L)) ← ((H)(L)) + 1	1

Мнемоника	Описание команд	Условное обозначение	Число байт
DCR R	Уменьшение содержимого регистра на единицу	(R) ← (R) – 1	1
DCR M	Уменьшение содержимого ячейки на единицу	((H)(L)) ← ((H)(L)) + 1	1
INX RP	Увеличение содержимого парных регистров B, C; D, E; H, L; SP на единицу	((RH)(RL))←((RH)(RL)) + 1	1
DCX RP	Уменьшение содержимого парных регистров B, C; D, E; H, L; SP на единицу	((RH)(RL))←((RH)(RL)) – 1	1
ANA R	Поразрядное логическое умножение содержимого регистра аккумулятора	(A) ← (A) ∧ (R)	1
ANA M	Поразрядное логическое умножение содержимого ячейки памяти и аккумулятора	(A) ← (A) ∧ ((H)(L))	1
ANI data	Поразрядное логическое умножение содержимого аккумулятора и байта	(A) ← (A) ∧ (byte2)	2
XRA R	Поразрядное логическое ИЛИ над содержимым регистра и аккумулятора	(A) ← (A) ⊕ (R)	1
XRA M	Поразрядное исключающее ИЛИ над содержимым ячейки памяти и аккумулятора	(A) ← (A) ⊕ ((H)(L))	1
XRI data	Поразрядное исключающее ИЛИ над содержимым аккумулятора и байтом	(A) ← (A) ⊕ (byte2)	2

Мнемоника	Описание команд	Условное обозначение	Число байт
ORA R	Поразрядное логическое сложение содержимого регистра и аккумулятора	(A) ← (A) ∨ (R)	1
ORA M	Поразрядное логическое сложение содержимого ячейки памяти и аккумулятора	(A) ← (A) ∨ ((H)(L))	1
RI data	Поразрядное логическое сложение содержимого аккумулятора и байта	(A) ← (A) ∨ (byte2)	2
CMP R	Сравнение содержимого регистра и	(A) – (R)	1

	аккумулятора		
CMP M	Сравнение содержимого ячейки памяти и аккумулятора	(A) – ((H)(L))	1
CPI data	Сравнение байта с содержимым	(A) – (byte2)	2
RLC	Циклический сдвиг содержимого аккумулятора влево	$(A_{n+1}) \leftarrow (A_n)$ $(A_0) \leftarrow (A_7)$ (CY) ← (A_7)	1
RRC	То же, но вправо	$(A_n) \leftarrow (A_{n+1})$ $(A_7) \leftarrow (A_0)$ (CY) ← (A_0)	1
RAL	Циклический сдвиг содержимого аккумулятора влево через перенос	$(A_{n+1}) \leftarrow (A_n)$ (CY) ← (A_7) (A_0) ← (CY)	1
RAR	То же, но вправо	$(A_n) \leftarrow (A_{n+1})$ (CY) ← (A_0) (A_7) ← (CY)	1
	Коды передачи управления		
JMP Addr	Безусловный переход	(PC) ← (byte2)(byte3)	3
JC Addr	Переход при наличии переноса	Если CY = 1, то (PC) ← (byte2)(byte3), иначе (PC) ← (PC) + 3	3

Мнемоника	Описание команд	Условное обозначение	Число байт
JNC Addr	Переход при отсутствии переноса	Если CY = 0, то (PC) ← (byte2)(byte3), иначе (PC) ← (PC) + 3	3
JZ Addr	Переход при нуле	Если Z = 1, то (PC) ← (byte2)(byte3), иначе (PC) ← (PC) + 3	3
JNZ Addr	Переход при отсутствии нуля	Если Z = 0, то (PC) ← (byte2)(byte3), иначе (PC) ← (PC) + 3	3
JP Addr	Переход при плюсе	Если S = 0, то (PC) ← (byte2)(byte3), иначе (PC) ← (PC) + 3	3
JM Addr	Переход при минусе	Если S = 1, то (PC) ← (byte2)(byte3), иначе (PC) ← (PC) + 3	3
JPE Addr	Переход при чётности	Если P = 1, то (PC) ← (byte2)(byte3), иначе (PC) ← (PC) + 3	3
JPO Addr	Переход при нечётности	Если P = 0, то (PC) ← (byte2)(byte3), иначе (PC) ← (PC) + 3	3
PCHL	Занесение в счётчик команд содержимого регистра H, L	(PC) ← (HL)	1
CALL Addr	Вызов подпрограммы	(SP – 1)(SP – 2) ← (PC); (SP) ← (SP) – 2; (PC) ← (byte3)(byte2)	3
CC Addr	То же, но при переносе	Если CY = 1, то (SP – 1)(SP – 2) ← (PC); (SP) ← (SP) – 2; (PC) ← (byte3)(byte2), иначе (PC) ← (PC) + 3	3
CNC Addr	То же, но при отсутствии переноса	Если CY = 0, то (SP – 1)(SP – 2) ← (PC); (SP) ← (SP) – 2; (PC) ← (byte3)(byte2), иначе (PC) ← (PC) + 3	3

Мнемоника	Описание команд	Условное обозначение	Число байт
CNZ Addr	То же, но при отсутствии нуля	Если Z = 0, то (SP – 1)(SP – 2) ← (PC); (SP) ← (SP) – 2; (PC) ← (byte3)(byte2), иначе (PC) ← (PC) + 3	3
CP Addr	Вызов подпрограммы при плюсе	Если S = 0, то (SP – 1)(SP – 2) ← (PC); (SP) ← (SP) – 2; (PC) ← (byte3)(byte2), иначе (PC) ← (PC) + 3	3

CM Addr	То же, но при минусе	Если S = 1, то (SP – 1)(SP – 2) ← (PC); (SP) ← (SP) – 2; (PC) ← (byte3)(byte2), иначе (PC) ← (PC) + 3	3
CPE Addr	Вызов подпрограммы при чётности	Если P = 1, то (SP – 1)(SP – 2) ← (PC); (SP) ← (SP) – 2; (PC) ← (byte3)(byte2), иначе (PC) ← (PC) + 3	3
CPO Addr	То же, но при нечётности	Если P = 0, то (SP – 1)(SP – 2) ← (PC); (SP) ← (SP) – 2; (PC) ← (byte3)(byte2), иначе (PC) ← (PC) + 3	3
RET	Возврат	(PC) ← (SP)(SP + 1) (SP) ← (SP) + 2	1
RC	Возврат при переносе	Если CY = 1, то (PC) ← (SP)(SP + 1) (SP) ← (SP) + 2, иначе (PC) ← (PC) + 1	1
RNC	Возврат при отсутствии переноса	Если CY = 0, то (PC) ← (SP)(SP + 1) (SP) ← (SP) + 2, иначе (PC) ← (PC) + 1	1
RZ	Возврат при нуле	Если Z = 1, то (PC) ← (SP)(SP + 1) (SP) ← (SP) + 2, иначе (PC) ← (PC) + 1	1

Мнемоника	Описание команд	Условное обозначение	Число байт
RNZ	Возврат при отсутствии нуля	Если Z = 0, то (PC) ← (SP)(SP + 1) (SP) ← (SP) + 2, иначе (PC) ← (PC) + 1	1
RP	Возврат при плюсе	Если S = 0, то (PC) ← (SP)(SP + 1) (SP) ← (SP) + 2, иначе (PC) ← (PC) + 1	1
RM	Возврат при минусе	Если S = 1, то (PC) ← (SP)(SP + 1) (SP) ← (SP) + 2, иначе (PC) ← (PC) + 1	1
PRE	Возврат при чётности	Если P = 1, то (PC) ← (SP)(SP + 1) (SP) ← (SP) + 2, иначе (PC) ← (PC) + 1	1
PRO	Возврат при нечётности	Если P = 0, то (PC) ← (SP)(SP + 1) (SP) ← (SP) + 2, иначе (PC) ← (PC) + 1	1
RST	Повторный запуск	(SP – 1)(SP – 2) ← (PC); (SP) ← (SP) – 2	1
	Команды ввода/вывода и прочие команды		
IN PORT	Ввод	(A) ← (N)	2
OUT PORT	Вывод	(N) ← (A)	2
CMA	Инвертирование аккумулятора	A ← $\bar{A}$	1
STC	Установка флага переноса в единицу	CY ← 1	1
CMC	Инвертирование флага переноса	CY ← $\overline{CY}$	1
EI	Разрешить прерывание		1
DI	Запретить прерывание		1
DAA	Двоично-десятичная коррекция содержимого аккумулятора	Если (A) ≤ 9, то (A) ← (A) + 00000000 Если (A) > 9, то (A) ← (A) + 00000110	1

Продолжение табл.

Мнемоника	Описание команд	Условное обозначение	Число байт
NOP	Отсутствие операции		
HLT	Останов		

Примечания:

1. **PSW** – слово состояния программы, первый байт которого равен содержимому аккумулятора, второй – содержимому регистра признаков.
2. **A** – регистр-аккумулятор (аккумулятор).
3. **Addr** – 16-разрядный адрес внешней памяти.
4. **data** – 8-разрядный символ данных.
5. **data 16** – 16-разрядный символ данных.
6. **byte 2** – содержимое второго байта в многобайтных командах.
7. **byte 3** – содержимое третьего байта в многобайтных командах.
8. **PORT** – 8-разрядный адрес интерфейса ввода/вывода.
9. **R**, **R1**, **R2** – один из регистров с именами A, B, C, D, E, H, L, принадлежащими блоку РОН.
10. **RP** – пара 8-разрядных регистров B, C; D, E; H, L; SP, которые адресуются как один регистр при операциях с 16-разрядными словами.
11. **RH** – старший регистр в паре.
12. **RL** – младший регистр в паре.
13. **PC** – 8-разрядный счетчик команд:
14. **SP** – 16-разрядный регистр-указатель стека:
15. **N** – номер порта.
16. $\mathbf{R_n}$ – номер разряда в регистре.
17. ← – оператор пересылки.
18. ↔ – оператор обмена.
19. $\wedge$ – оператор И.
20. $\vee$ – оператор ИЛИ.
21. $\oplus$ – оператор исключающее ИЛИ;
22. $^{-}$ – оператор инверсии.
23. + – оператор арифметического сложения.
24. – – оператор арифметического вычитания:
25. **( )** – содержимое регистра или ячейки памяти.
26. **( ) ( )** – оператор объединения двух 8-разрядных регистров, при этом их содержимое рассматривается как одно 16-разрядное слово.
27. **Z** – признак нуля; если результат выполнения команды равен 0, то Z = l.
28. **S** – признак знака; если результат выполнения команды отрицателен, то S = l.
29. **P** – признак чётности; если сумма по модулю 2 значений всех разрядов результата выполнения операции равна 0, то P = 1.
30. **CY** – признак переноса; если на выходе старшего разряда сумматора в результате выполнения команды сложения возникает сигнал переноса или в результате выполнения команды вычитания возникает заем, то CY + 1.

ПРИЛОЖЕНИЕ 2

СПИСОК КОМАНД В МАШИННЫХ КОДАХ

Команда на ассемблере	Код операции	Команда на ассемблере	Код операции	Команда на ассемблере	Код операции
ACI data	CE	CMP D	BA	INR C	0C
ADC A	8F	CMP E	BB	INR D	14
ADC B	88	CMP H	BC	INR E	1C
ADC C	89	CMP L	BD	INR H	24
ADC D	8A	CMP M	BE	INR L	2C
ADC E	8B	CNC Addr	D4	INR M	34
ADC H	8C	CNZ Addr	C4	INX B	03
ADC L	8D	CP Addr	F4	INX D	13
ADC M	8E	CPE Addr	EC	INX H	23
ADD A	87	CPI Addr	FE	INX SP	33
ADD B	80	CPO Addr	E4	JC ADDR	DA
ADD C	81	CZ Addr	CC	JM ADDR	FA
ADD D	82	DAA	27	JMP ADDR	C3
ADD E	83	DAD B	09	JNC ADDR	D2
ADD H	84	DAD D	19	JNZ ADDR	C2
ADD L	85	DAD H	29	JP ADDR	F2
ADD M	86	DAD SP	39	JPE ADDR	EA
ADI DATA	C6	DCR A	3D	JPO ADDR	E2
ANA A	A7	DCR B	05	JZ ADDR	CA
ANA B	A0	DCR C	0D	LDA ADDR	3A
ANA C	A1	DCR D	15	LDAX B	0A
ANA D	A2	DCR E	1D	LDAX D	1A
ANA E	A3	DCR H	25	LHLD ADDR	2A
ANA H	A4	DCR L	2D	LXI B, DATA 16	01
ANA L	A5	DCR M	35	LXI D, DATA 16	11
ANA M	A6	DCX B	0B	LXI H, DATA 16	21
ANI DATA	E6	DCX D	1B	LXI SP, DATA 16	31
CALL ADDR	CD	DCX H	2B	MOV A, A	7F
CC ADDR	DC	DCX SP	3B	MOV A, B	78
CM ADDR	FC	DI	F3	MOV A, C	79
CMA	2F	EI	FB	MOV A, D	7A
CMC	3F	HTL	76	MOV A, E	7B
CMP A	BF	INR PORT	DB	MOV A, H	7C
CMP B	B8	INR A	3C	MOV A, L	7D

Команда на ассемблере	Код операции	Команда на ассемблере	Код операции	Команда на ассемблере	Код операции
CMP C	B9	INR B	04	MOV A, M	7E
MOV B, A	47	MOV H, L	65	POP D	D1
MOV B, B	40	MOV H, M	66	POP H	E1
MOV B, C	41	MOV L, A	6F	POP PSW	F1
MOV B, D	42	MOV L, B	68	PUSH B	C5
MOV B, E	43	MOV L, C	69	PUSH D	D5
MOV B, H	44	MOV L, D	6A	PUSH H	E5
MOV B, L	45	MOV L, E	6B	PUSH PSW	F5
MOV B, M	46	MOV L, H	6C	RAL	17
MOV C, A	4F	MOV L, L	6D	RAR	1F
MOV C, B	48	MOV L, M	6E	RC	D8
MOV C, C	49	MOV M, A	77	RET	C9
MOV C, D	4A	MOV M, B	70	RLC	07
MOV C, E	4B	MOV M, C	71	RM	F8
MOV C, H	4C	MOV M, D	72	RNC	D0
MOV C, L	4D	MOV M, E	73	RNZ	C0
MOV C, M	4E	MOV M, H	74	RP	F0
MOV D, A	57	MOV M, L	75	RPE	E8
MOV D, B	50	MVI A, DATA	3E	RPO	E0
MOV D, C	51	MVI B, DATA	06	RRC	0F
MOV D, D	52	MVI C, DATA	0E	RST 0	C7
MOV D, E	53	MVI D, DATA	16	RST 1	CF
MOV D, H	54	MVI E, DATA	1E	RST 2	D7
MOV D, L	55	MVI H, DATA	26	RST 3	DF
MOV D, M	56	MVI L, DATA	2E	RST 4	E7
MOV E, A	5F	MVI M,DATA	36	RST 5	EF

MOV E, B	58	NOP	00	RST 6	F7
MOV E, C	59	ORA A	B7	RST 7	FF
MOV E, D	5A	ORA B	B0	RZ	C8
MOV E, E	5B	ORA C	B1	SBB A	9F
MOV E, H	5C	ORA D	B2	SBB B	98
MOV E, L	5D	ORA E	B3	SBB C	99
MOV E, M	5E	ORA H	B4	SBB D	9A

Команда на ассемблере	Код операции	Команда на ассемблере	Код операции	Команда на ассемблере	Код операции
MOV H, A	67	ORA L	B5	SBB E	9B
MOV H, B	60	ORA M	B6	SBB H	9C
MOV H, C	61	ORI DATA	F6	SBB M	9D
MOV H, D	62	OUT PORT	D3	SBB M	9E
MOV H, E	63	PCHL	E9	SBI DATA	DE
MOV H, H	64	POP B	C1	SHLD ADDR	22
SPHL	F9	SUB E	93	XRA D	AA
STA Addr	32	SUB H	94	XRA E	AB
STAX B	02	SUB L	95	XRA H	AC
STAX D	12	SUB M	96	XRA L	AD
STC	37	SUI data	D6	XRA M	AE
SUB A	97	XCHG	EB	XRI data	EE
SUB B	90	XRA A	AF	XTHL	E3
SUB C	91	XRA B	A8		
SUB D	92	XRA C	A9		

СПИСОК ЛИТЕРАТУРЫ

1. Системы на микроконтроллерах и БИС программируемой логики. – М. : ЭКОМ, 2002.
2. Сектор электронных компонентов. Россия-99. – М. : ДОДЭКА, 1999.
3. Проектирование импульсных и цифровых устройств радиотехнических систем : учеб. пособие для радиотехнич. спец. вузов / под ред. Ю.М. Казаринова. – М. : Высшая школа, 1985. – 319 с.
4. Калабеков, Б.А. Микропроцессоры и их применение в системах передачи и обработки сигналов : учеб. пособие для вузов / Б.А. Калабеков. – М. : Радио и связь, 1988. – 368 с.
5. Абель, П. Язык ассемблера для IBM PC и программирования /П. Абель ; пер. с англ. Ю.В. Сальникова. – М. : Высшая школа, 1992. – 447 с.
6. Пухальский, Г.И. Цифровые устройства : учеб. пособие для втузов / Г.И. Пухальский, Т.Я. Новосельцева. – СПб. : Политехника, 1996. – 885 с.
7. Угрюмов, Е.П. Цифровая схемотехника / Е.П. Угрюмов. – СПб. : БХВ – Санкт Петербург, 2000. – 518 с.