Я Н Л О Г А Н

М И К Р О - Э В М " Z X S P E C T R U M "

Руководство по применению микро ЭВМ "ZX SPECTRUM"
фирмы "SINCLAIR RESEARCH Ltd."

С О Д Е Р Ж А Н И Е

Введение

Почти невозможно поверить, что всего лишь за 2.5 года фирма Синклер Рисоч в Кембридже изготовила и продала около 500 тыс. микрокомпьютеров. Весной 1980 г. был разработан микрокомпьютер ZX-80. Разработка этой машины явилась большим успехом, поскольку это был первый сравнительно дешевый микрокомпьютер для любителей. Однако, Клив Синклер и его лаборатория изготовила машину ZX81 всего за один год. Эта модель была более совершенной по сравнению с ZX80 и явилась развитием микрокомпьютера с низким разрешением, черно-белым экраном.

Теперь мы имеем ZX-Спектрум. Эта машина была разработана по принципу ZX80 и ZX81. Сделав это, фирма Синклер Рисоч разработала компьютер с высоким разрешением и цветным экраном. Однако, с сожалением надо отметить, что ZX80 и ZX81 были вытеснены, хотя обе являются прекрасными машинами. Они просты в работе, что приятно для программиста. Но это не означает, что Спектрум - трудная машина для пользования, но я чувствую, что для того, чтобы от нее получить все, потребуется написание более совершенных программ.

Эта книга была написана таким образом, чтобы читатель мог дополнить свои знания по Спектрум, полученные по двум руководствам, прилагаемым к машине, (Стив Викерс) и тем самым получить более глубокое понимание работы со Спектрум и микрокомпьютерами в целом.

1. Принцип работы микрокомпьютерной системы

1.1. Предисловие

Всегда очень любопытно погрузиться в чтение книги, открывая ее то там, то здесь ... Но компьютер является самой логичной машиной из всех машин и каждый, кто попытается совершенствовать свои познания, должен начинать с самого начала.

1.2. Три взгляда на машину

Возможно описать любую микрокомпьютерную систему, приняв три различных концепции. Первая является общим взглядом на на систему, которая будет включать в себя фактически микрокомпьютер и всю его периферию. Вторая - описывает физические связи микрокомпьютера. Третья описывает логику работы компьютера.

Концепция "системы", вероятно, уже хорошо знакома большинству читателей, но все-таки она приводится, т.к. могут быть читатели, которые не знакомы с системой Спектрум.

1.3. Описание системы

Сам по себе микрокомпьютер Спектрум представляет собой пластмассовый корпус 233x144x30 mm.

На верхней части расположены 40 клавиш, которые образуют клавиатуру.

Сзади расположены входные и выходные разъемы и разъем для подключения блока питания.

На основной печатной плате (ПП) находится микропроцессор Z80 и другие элементы, включая динамик. Плата находится в одном корпусе с клавиатурой, но отделена от нее. Клавиатура и ПП соединены между собой двумя ленточными кабелями.

Система разработана таким образом, что она может отображать ее на принтер или другое периферийное оборудование.

1.4. Физические связи

Для доступа к основной печатной плате необходимо снять верхнюю крышку с клавиатурой, отвернув 5 винтов снизу. Надо проявлять осторожность, чтобы не порвать два кабеля, идущие от клавиатуры. Отключать их не рекомендуется, но при необходимости это можно сделать: зажав их двумя одинаковыми палочками, равномерно вынуть.

Рассмотрим далее каждый элемент.

Микропроцессор Z80

Это кремниевый кристалл (ЧИП). Он является самым важным из всех элементов и предназначен для выполнения программ. Программа для Z80 представляет собой набор кодовых инструкций и согласованных с ними данных. Процессор Z80 работает на определенной частоте - 3.5 МГЦ, что позволяет выполнять 875000 простых кодовых инструкций в секунду. При подаче питания на микропроцессор, он через некоторое время переходит в рабочее состояние.

16К ROM (постоянная память)

Представляет собой ЧИП объемом 129 Кбит или 16 Кбайт памяти, и содержит программу МОНИТОРА в машинных кодах для Z80.

Программа "Монитор 16К" разбита на три части:

7 Кбайт - операционная система;
8 Кбайт - интерпретатор Бейсика;
1 Кбайт - генератор знаков.

16K RAM (оперативная память - ОЗУ)

В стандартной версии 16К Спектрум имеет 8 ЧИПов по 2 Кбайт, или 16 Кбит. Версия 48К - содержит дополнительно еще 32 Кбайт ОЗУ.

3 из 8 ЧИПов памяти образуют "область экрана" и обычно используются только для этой цели. Четвертый ЧИП предназначен для хранения всех атрибутов 768 знаков экрана (24 строки по 32 символа) и системных переменных. В версии 16К остается свободным немного более 8Кбайт ОЗУ.

ULA

Это большой ЧИП, содержащий в себе много небольших ЧИПов В Спектруме ULA связана больше всего с "областью экрана" и

"областью атрибутов" для получения ТВ сигнала.

ПАЛ - кодировщик

Этот ЧИП получает цветную информацию от ULA и использует ее в формировании требуемого сигнала для УКВ модулятора. В английской версии Спектрум сигнал, полученный от модулятора, находится на 36-м дециметровом канале.

На ПП дополнительно размещены: динамик, радиатор, регулятор напряжения, системные часы, ЧИПы выборки адресов, ЧИПы буферов и др.

1.5. Логика работы

Связи между различными элементами микрокомпьютерной системы осуществляются по дорожкам ПП или по проводам, и надо хорошо представлять себе их назначение.

Микропроцессор Z80 может обращаться к отдельным 65536 адресам памяти (64К). Ограничением на непосредственную адресацию Z80 является 64Кбайт. В версии микрокомпьютера 16К можно адресоваться к адресам от 0 до 32767, а в 48К - доступны адреса от 0 до 65535. Адрес задается в виде 16-битной комбинации. 0 представляется как 0000 0000 0000 0000, а 65535 - 1111 1111 1111 1111. Адреса, выданные Z80, поступают на "Адресную шину", состоящую из 16 линий, находящихся при высоком или низким напряжении (логический "0" или логическая "1"). Адрес можно указывать двумя байтами по 8 бит. Поэтому любые данные (машинная инструкция или данные) могут представляться как 0-255 в десятичном виде или 0000 0000 - 1111 1111 - в двоичном.

Как часть логической концепции Спектрум, также важно рассмотреть нормальный режим работы системы и обсудить "карту памяти".

Синклер Рисоч выпускает Спектрум с мониторной программой 16К и обеспечивает пользователя операционной системой (ОС) и интерпретатором Бейсик (ИБ). По существу возможно вместо программы Монитор использовать Z80 с программой в машинных кодах.

При обычном использовании операционная система не требует со стороны пользователя каких-либо сложных действий. Поэтому при работе Спектрум уже работает ИБ как часть мониторной программы. Пользователь может сразу вводить или выполнять Бейсик-программу - подпрограммой ИБ.

Заметьте, что Z80 сам по себе не может выполнять Бейсик-программу, а только мониторную программу, представленную в

машинных кодах Z80. Исключением является программа в машинных кодах, написанная пользователем.

Карта памяти стандартной версии 16К представлена на рис. 1.1 и далее коротко описана.

ROM область

16К ROM, содержащие ОС, ИБ и генератор знаков (ГЗ), располагаются в адресах 0-16383 (16-ричные 0000-3FFF). В адресе "0" хранится стартовый адрес программы в кодах.

Карта памяти экранной области

6К памяти с 16384 до 22527 (4000-57FF) обеспечивает высокое разрешение на ТВ. Важно, что расположение "экранной области" зафиксировано в аппаратуре Спектрум и не может меняться программно. Имеется взаимно-однозначное соответствие всех битов области и точек экрана ТВ, и следующие вычисления показывают, что число битов 6К памяти - 1024x8x6=49152 (битов).

Количество байтов в 24 строках экрана (по 64 точки на символ) равно 32x24x64=49152 (точек).

32767	7FFF	Графическая область пользователя	P-RAMT
32600	7F58		UDG
32599	7F57		RAMTOP
		Стек Машинный стек	
		Резервная память	STKEND
		Стек вычислителя	STKBCT
		Рабочая область	WORKSP
		Область редактора	E_LINE
		Область переменных	VARS
		Область Бейсик-программы	PROG
		Область информационных каналов	CHANS
(23734	5CB6	Память микронакопителей)	
23552	5C00	Системные переменные	
23296	5B00	Буфер печати	
22528	5800	Область атрибутов	
16384	4000	Область экрана	
0	0000	ROM область	

Рис.1.1

Экран ТВ разбит на три части. Линиям с номерами 0-7 соответствуют адреса памяти 16384-18431 (16-ричн. 4000-47FF). Линиям 8-15 - адреса 18432-2С497 (16-ричн. 4800-4FFF). Линиям 16-23 соответствуют адреса 20480-22527 (16-ричн. 5000-57FF).

Рассмотрим каждые 2К памяти, как 8 по 1/4 Кбайт областей. Каждая из 1/4 К областей задает биты одной строки (256 точек) трети экрана, начиная сверху. Это справедливо для всех 24 по 1/4 К областей экрана.

Область атрибутов.

Экран имеет 768 символов,каждый из которых имеет один из 8 цветов "бумаги" ("PAPER"), 8-и цветов "чернил" ("INK"), признак мерцания, признаки повышенной или нормальной яркости.

Область расположена в адресах 22528-23295 (5800-5AFF), где закодированы биты, последовательно определяющие атрибуты экрана.

Зависимость между площадями символа и байтами атрибута несложная, т.к. байты просматриваются подряд для каждой линии экрана сверху вниз и слева направо.

Биты 0-2 определяют цвет "чернил" , 3-5 - цвет "бумаги", 6 - яркость, 7 - мерцание.

Буфер печатающего устройства

Ячейки памяти с десятичными адресами от 23296 до 23551 (5B00-5BFF) используются как буфер печатающего устройства.

Этих 256 байт достаточно для хранения с высокой разрешающей способностью 32 символов, причем биты первых 32 байтов соответствуют символам верхней строки, следующие 32 байта соответствуют второй строке и т.д.

Отметим, что буфер печатающего устройства при необходимости можно использовать как рабочую память.

Системные переменные

В 182 ячейках памяти с десятичными адресами от 23552 до 23733 (5C00-5CB5) содержатся различные системные переменные микро-ЭВМ "Спектр". Подробное рассмотрение этих переменных будет производиться по мере возникновения необходимости.

Схемы распределения памяти микронакопителей

Эта область памяти начинается в ячейке с десятичным адресом 23734 (5CB6) и в стандартной модели микро-ЭВМ существует только теоретически. Однако, необходимо отметить, что эта область используется только после подключения микронакопителей.

Поскольку эта книга писалась до появления микронакопителей, дальше область схем распределения памяти микронакопителей не рассматривается. Задав, однако, с помощью системной переменной CHANS неиспользуемый участок памяти, можно зарезервировать в пределах постоянной памяти область для этих схем. (Интересной представляется возможность использования области схем распределения памяти микронакопителей для хранения пользовательских программ, написанных на машинном языке).

Область информационных каналов

Эта специальная область памяти начинается с ячейки, адресуемой системной переменной CHANS, которая сама хранится в ячейках с адресами 23631 и 23632 (5C4F-5C50). Это область переменного размера; заканчивается она ячейкой, в которой содержится метка конца - десятичное значение 128 (80H).

В стандартной модели микро-ЭВМ - без микронакопителей, заданы характеристики четырех каналов. Это следующие:

1. Канал K - позволяет ввод данных с клавиатуры и вывод в нижнюю часть экрана.
2. Канал S - не позволяет ввода, но предусматривает вывод в верхнюю часть экрана, которая при необходимости может быть увеличена.
3. Канал R - тоже не позволяет ввод, но допускает вывод в рабочую область памяти.
4. Канал P - тоже не позволяет ввод, но допускает вывод на печатающее устройство.

Информация о каждом канале содержит 5 байт. Это 2 байта адреса программы ввода и код (одна буква) имени канала. Четыре канала и метка конца области информационных каналов в стандартной модели микро-ЭВМ занимают 21 ячейку с адресами от 23734 до 23754 (5CB6-5CCA).

Область программы, написанной на языке Бейсик

В этой области памяти содержатся строки текущей программы, написанной на языке Бейсик (если они имеются). Размер области зависит от количества строк в текущей программе.

Начальная ячейка области адресуется системной переменной

PROG, которая содержится в ячейках с адресами 23635 и 23636 (5C53-5C54).

В стандартной модели содержимое системной переменной PROG - 23755 (5CCB), если только не используется область схем распределения памяти для информационных каналов.

В этой области строки программы, написанной на Бейсике, хранятся в следующем формате:

В первых двух байтах каждой строки содержится номер строки (первый байт - старший, второй - младший).

В третьем и четвертом байтах содержится длина остальной части. На этот раз младший байт расположен перед старшим. Длина остальной части - это число байтов с пятого по конечный символ ENTER включительно.

Дальше расположена сама строка программы. Для обозначения некоторых символов используются коды, разработанные фирмой "Синклер Рисоч"; для стандартных цифробуквенных символов - коды ASCII.

Последний байт строки всегда ENTER.

В пределах одной строки, написанной на Бейсике, отдельные операторы разделены двоеточиями - десятичный код 58 (3A). Других меток для разделения операторов не предусмотрено. Заметим, что если в строке, написанной на Бейсике, имеется десятичное число, оно хранится в виде соответствующих символов ASCII с последующим символом NUMBER - дес. код 14 (0Eh) в виде десятичного числа с плавающей запятой, независимо от того, будет ли это число с плавающей запятой или целое число из интервала - 65535 до +65535. Таким образом, для каждого десятичного числа, включенного в программу, написанную на Бейсике,требуется 6 байтов дополнительной памяти.

Приведенная ниже домонстрационная программа показывает рассмотренные выше области в своей области памяти.

Программа, демонстрирующая область программы:

```
10 FOR A=23755 TO 23817: PRINT A; TAB 9; PEEK A;
   TAB 15; CHR$ PEEK A: NEXT A
RUN
```

Область переменных

Начальная ячейка этой области всегда адресуется системной переменной VARS, которая содержится в ячейках памяти с десятичными адресами 23637 и 23638 (5C4B-5C4C).

Отметим, что в микро-ЭВМ адрес начальной ячейки области переменных не изменяется при выполнении любой программы, написанной на Бейсике. В то же время, ее размер всегда изменяется при задании новых переменных.

В последней ячейке области всегда содержится метка конца - десятичный символ 128 (80).

Приведенная выше программа показывает область используемых в ней переменных (а именно, управляющей переменной цикла FOR-NEXT).

Программа, демонстрирующая область переменных:

```
10 FOR  A = 23804 TO 23823: PRINT A; TAB 9; PEEK A: NEXT A
   RUN
```

Область редактирования

Начальная ячейка этой области, содержащей введенную или редактируемую строку, написанную на Бейсике, всегда адресуется системной переменной E-LINE, которая содержится в ячейках с десятичными адресами 23641 и 23642 (5C59-5C5A). В случае, когда на нижней части экрана телевизионного дисплея отображается только мерцающий курсор, область редактирования занимает только три ячейки. В первой ячейке, адресуемой также системной переменной K-CUR, содержится символ ENTER, во второй ячейке - метка конца (десятичный код 128, или 80H). Нижняя часть экрана получается при копировании и отображении редактируемой строки.

Подобная процедура используется при отображении строки, написанной на Бейсике в нижней части экрана с помощью клавиши EDIT. Прежде всего область редактирования расширяется таким образом, чтобы поместить всю строку. Затем производится копирование строки из области редактирования в область ОЗУ, соответствующую нижней части экрана дисплея. На последнем этапе из кодов символов формируется изображение с высокой разрешающей способностью.

Поскольку область редактирования является динамической областью, изменяющейся при каждом использовании, то на данном этапе рассмотрения микро-ЭВМ нецелесообразно приводить образец демонстрационной программы, написанной на Бейсике.

Рабочая область памяти

Эта область памяти используется для различных целей, например, для ввода данных, соединения символьных строк и т.д. Начальная ячейка этой области адресуется системной переменной WORKSP, которая содержится в ячейках с десятичными адресами 24649 и 23650 (5C61 и 5C62). При этом выделяется участок памяти необходимого размера. После использования рабочая область памяти очищается, т.е. сжимается до нуля, что помогает избежать использования большего количества ячеек, чем это необходимо для работы.

Поскольку эта область также является динамической, то на данном этапе рассмотрения невозможно привести простой пример демонстрационной программы, написанной на Бейсике.

Стек вычислителя

Начальная ячейка этой очень важной области памяти адресуется системной переменной STKBOT, которая содержится в ячейках памяти с дес. адресами 23651 и 23652 (5C63 и 5C64); конечная ячейка адресуется системной переменной STKEND, расположенной в ячейках с адресами 23653 и 23654 (5C65 и 5C66).

Стек вычислителя используется для хранения пятибайтовых чисел с плавающей запятой, а при рассмотрении символьных строк - для хранения пяти байтов параметров строки.

Стек обрабатывается по принципу "последним описан - первым считан" ("last in - first out"), и в вершине стека всегда хранится последнее поступившее значение.

Свободная память

Область памяти между стеком вычислителя и стеком микро-ЭВМ представляет собой доступный пользователю участок памяти. В стандартной модели с памятью 16К при включении системы в этой области имеется, по крайней мере, 8839 ячеек памяти. Однако, стоит помнить, что наименьшее приемлемое значение для очистки памяти равно 23821, и эта ячейка удалена от системной переменной RAMTOR на 8878 байт.

Стек микро-ЭВМ

Микропроцессор Z80 должен иметь некоторую рабочую область для использования при работе микропроцессора - так называемый стек микро-ЭВМ. Указатель стека микро-ЭВМ в микропроцессоре Z80 всегда указывает на последнюю заполненную ячейку памяти.

Стек микро-ЭВМ намного подробнее рассматривается в разделах книги, касающихся машинного языка.

Стек GO SUB

При каждом использовании цикла GO SUB номера строк цикла хранятся в стеке GO SUB.

Этот стек в памяти растет вниз, причем каждый адрес занимает в стеке три ячейки. В самой верхней ячейке содержится номер оператора в написанной на Бейсике строке, к которому будет выполнен возврат. В следующей ячейке содержится младший байт номера цикла, а третьей - старший байт.

Приведенная ниже демонстрационная программа показывает стек GO SUB, используемый для хранения номеров строк цикла для трех вложенных подпрограмм.

Программа, демонстрирующая стек GO SUB

```
10 GO SUB 20: STOP
20 GO SUB 30: RETURN
30 GO SUB 40: RETURN
40 FOR A=32597 TO 32589 STEP 1:PRINT A, PEEK A:
   NEXT A: RETURN
 RUN
```

Две ячейки над стеком GO SUB всегда содержат значения 0 и 62 (00h и 3Eh); любое значение между ними рассматривается как недействительный номер строки. В случае, если в написанной на Бейсике программе содержится лишняя команда RETURN, микро-ЭВМ осуществляет переход к этой недействительной строке, в результате чего на печать выведется сообщение об ошибке "RETURN without GO SUB" ("ВОЗВРАТ без GO SUB").

Примечание: Разработанная фирмой Sinclair Research процедура расщепления GOM SUB не полностью согласуется с общепринятой.

Системная переменная RAMTOR, расположенная в ячейках с адресами 23730 и 23731 (5CB2h и 5CB3h), адресует ячейку, в которой содержится число 62. Эта ячейка считается последней ячейкой области языка Бейсик.

Область заданных пользователем графиков

До тех пор, пока системная область Бейсика не сдвинута вниз в результате выполнения команды CLEAR, в верхних 168 ячейках памяти содержится битовое представление 21 заданного пользователем графика.

Копирование в эту область битовых представлений букв от A до U является частью процедуры инициализации микро-ЭВМ. Затем пользователь может изменить эти представления, вводя не более 21 графика.

Последняя ячейка этой области всегда адресуется системной переменной P-RAMT, которая расположена в ячейках с адресами 23732 и 23733 (5CB4-5CB5H).

В стандартной модели микро-ЭВМ с памятью 16К в системной переменной P-RAMT должно содержаться 32767 - что свидетельствует о том, что все 16К памяти находятся в рабочем состоянии.

Целесообразно иногда вводить строку:

```
PRINT PEEK 23732+ 256*PEEK 23733
```

проверяя, что результат действительно равен 32767. (В модели с памятью 48К результат должен равняться 65536).

ГЛАВА 2. Команды и функции Бейсика

2.1. Введение

Поскольку предполагается, что читатель уже достаточно близко знаком с языком Бейсик микро-ЭВМ "Спектр", при рассмотрении команд и функций языка Бейсик сосредоточимся на вопросах, которые недостаточно подробно обсуждены в двух руководствах, входящих в комплект поставки микро-ЭВМ "Спектр".

Интерпретатор языка Бейсик микро-ЭВМ "Спектр" признает 50 различных команд и 33 функции. Кратко остановимся на каждой из них. Рассмотрение проводится в алфавитном порядке.

2.2. Команды Бейсика

BEEP x,y

По этой команде подается сигнал на динамик, x - длительность в секундах. y - высота звукового тона, от ЛЯ первой октавы в десятичном диапазоне чисел -60 до +69,8. Каждый из операндов, x или y, либо оба вместе могут быть выражениями.

Интересно отметить, что BEEP не прерывается по клавише BREAK.

BREAK возможен только в конце оператора, содержащего команду BEEP.

BORDER m

Имеется 8 возможных цветов, которые могут быть переданы граничной области (бордюру) ТВ-экрана. Используются целые числа от 0 до 7. Однако, допускается интервал от -0.5 < m < 7.5, где m - результат выражения, поскольку m округляется до целого.

По команде BORDER выходной сигнал поступает на порт 254 и это может быть выражено оператором:

OUT 254 m (где, например, m=2 - дает красный бордюр).

Но OUT и BORDER различны. Цвет, получаемый бордюром по команде OUT - "временный цвет", в то время как BORDER дает "постоянный" цвет, хранящийся в системной переменной BORDER (десятичный 23624,16-ричный 5C48).

BORDER и ENTER - делает бордюр красным.
OUT 254,1 и ENTER - бордюр будет временно голубым, а затем опять вернется в красный цвет.

BRIGHT m

Это первая команда из команд управления цветом. Все они могут использоваться как команды в операторах Бейсика (в этом случае их действие постоянно), либо включаться в операторы печати, и тогда их действие распространяется только на данный оператор печати; m может быть выражением, но результат должен быть целым числом 0, 1 или 8.
При m=8 "высветленным" будет текст, подлежащий печати.
Следующие примеры показывают 4 различных способа использования команды BRIGHT.
- два изменения яркости имеют постоянный характер:

```
10 BRIGHT 1: PRINT "Bright": BRIGHT 0: PRINT "Normal"
```

- изменения яркости временны в течение оператора:

```
20 PRINT BRIGHT 1; "BRIGHT -"; BRIGHT 8; "Normal"
```

- команда CHR$ - эквивалентна:

```
40 LET A$ = CHR$ 19 + CHR * : LET B$ = CHR$19 + CHR$ 8;
   PRINT A$; "Bright -"; B$; "Normal"
```

-помещает команду цвета в строку переменных.
Также возможно использование последовательности:
- режим курсора "E" и клавиша "9" - для получения BRIGHT 1;
- режим курсора "E" и клавиша "8" - для получения BRIGHT 0.

Этот набор может быть размещен внутри скобок, т.е. между открывающей скобкой и первым символом или как единственный символ строки. Например:
LET a$ = "": REM - внутри кавычек набрана вышеприведенная последовательность с клавишей "9".
Строка a$ будет напечатана как при BRIGHT 1.

CAT

Используется с микроприводом (подробности отсутствуют).

CIRCLE x,y,z

Эта команда рисует окружность радиусом Z с координатами центра x,y. Z берется как абсолютное целое значение, а x и y используются как числа с плавающей запятой. Самая большая

окружность при радиусе 88 единиц получается оператором CIRCLE 127.5,87,88, а окружность с нулевым радиусом - представляет собой простую точку. Любая команда цвета может быть включена в оператор CIRCLE и ее влияние всегда будет временным. По некоторым стандартам команда CIRCLE довольно медленна и не точна, но тем не менее, она очень полезна.

CLEAR, CLEAR N

Система Спектрум имеет такое большое количество области RAM, доступной для пользователя, что выполнение команды CLEAR самой по себе вряд ли будет полезным. Однако, расширение команды возможностью перемещать указатель RAMTOR делает ее очень сильной. RAMTOR указывает на верхнюю границу системы Бейсик и содержимое адресов ниже RAMTOR может изменяться, в то время как любое содержимое выше RAMTOR остается невредимым, даже после выполнения команды NEW.

Нижнюю границу для N можно выбирать не меньше, чем 23821. Иначе Спектрум выдает звуковой сигнал. Это говорит о том, что отведенного RAM не хватает на задачу. Верхней границей N является для системы 16К - 32767, а для системы 48К - 65535. Использование CLEAR N с соответствующим значением вызывает помещение стека GO SUB в область, отведенную для графики пользователя. Поучительно выполнить следующее упражнение:

Ввести CLEAR 32767 (или 65535).

Изменить курсор на G и ввести литеры с L по U.

Нажать клавишу SPACE на несколько секунд и наблюдать изменение графики пользователя, поскольку каждое нажатие клавиши приводит к использованию машинного стека.

CLOSE

Используется с микроприводом.

CLS

Это очень простая команда, которая меньше, чем за 1/10 секунды выполняет много работы. Команда CLS очищает дисплейную область. Она зануляет адреса с 16384 по 22527 (4000-5800h) и сбрасывает байты атрибутов по адресам 22528 - 23295 (58CC-5AFFh). Байты атрибутов не зануляются, а копируются из системной переменной ATTR-P, которая перемещается по адресу 23693 (5C8Dh) и может быть изменена. В переменной ATTR-P бит 7 отвечает за "мерцание", бит 6 - за "яркость", биты 3-5 - за цвет "бумаги" и биты 0-2 - за цвет "чернил".

CONTINUE

В большинстве требуемых случаев эта команда работает хорошо. Но когда пользователь выполняет несколько команд непосредственно, компьютер зацикливается и сможет восстановить работу только по клавише BREAK. Существует два особых аспекта команды. Первый позволяет пользователю использовать в программе Бейсика оператор STOP, и тогда не требуется использование клавиши BREAK. Пользователь в этом случае может проверить переменные, установить их, и изменить программу. Использование операторов STOP и CONTINUE существенно облегчает отладку программ.

Второй аспект позволяет пользователю повторять выполнение после исправления ошибки. Например, если программа остановилась по ошибке "variable not found", то переменная может быть определена в непосредственном режиме и оператор заново выполнен по команде CONTINUE.

Шесть системных переменных - NEWPPC, NSPPC, PPC, SUBPPC, OLDPPC и OS PEC - определенным образом включаются в выполнение команды CONTINUE.

COPY

Это очень простая команда. Поскольку принтер управляется SP, верхние 22 строки ТВ экрана посылаются на принтер. Линии ТВ экрана выводятся по очереди. Команда COPY одна из немногих команд, которые останавливают часы реального времени на время выполнения команды. Это можно видеть из системной переменной FRAMES до и после использования команды.

DATA e, e...

Это команда, которая может использоваться только в программных строках, определяет список данных. Хотя это и упоминается в руководстве, но не становится ясным, что рассмотрение элементов в операторе DATA как выражений является особенностью, делающей эту команду очень полезной. Более детально см. READ и RESTORE.

DEF FN a (a, ..., z) = e и DEF FN a$(a, ...,z)=1

Команда DEF FN очень мощная. Пользователь может определять до 52 функций - 26 числовых и 26 строковых. Имена, используемые для функций, всегда должны быть одиночного характера (+$ - для строк), и они могут быть именами, используемыми где-нибудь еще как простые переменные.

Существует небольшое ограничение на имена аргументов, так как они тоже должны быть одинарными литералами (+$ для

строк). Поэтому может быть не более 52 аргументов. Выражение, определяемое функцией, может быть любым, дающим числовой или строковый результат. Однако, рекурсивное использование приводит к неудаче (это известно автору как единственный путь к краху Бейсика для Спектрум).

DIM a(e ... l) и DIM a$(e ... l)
1 k 1 k

Команда DIM удаляет существующий массив с тем же именем и затем устанавливает новый. Числовые массивы обнуляются, а строковые заполняются пробелами.

В системе Спектр вся индексация начинается с 1, или более точно, с 0,5<e<1,5. Достижение индексом нулевого значения является обычной ошибкой, и приводит к сообщению "subscript wrong". Использование в Спектруме в литеральных массивах строк фиксированной длины, тогда как простая литеральная переменная изменяет длину, может привести к неудаче. Но наиболее просто, используя в определении литерального массива последний индекс, всегда фиксировать длину строки. Эта особенность может быть наиболее полезна при установлении формата ТВ экрана.

DRAW x,y DRAW x,y,z

Эта команда рисует линию от текущей позиции (но не включая ее) на расстояние x - по горизонтали, и y - по вертикали. Если аргумент z определен, рисуется дуга вместо прямой линии. Аргумент z должен быть определен в радианах, при z=PI получается полуокружность. При положительных значениях z, дуги рисуются справа от того места, где была бы прямая линия, а при отрицательных - слева. Любая из команд управления цветом может быть включена в оператор DRAW, и ее действие будет распространяться на время выполнения команды.

ERASE

Используется с микроприводом.

FLASH m

Это вторая команда управления цветом. Когда она используется в Бейсике, ее влияние постоянно, но когда используется при включении в оператор печати, ее влияние - только на данный оператор.

Как и BRIGHT, может принимать значения 0,1 или 8. При m=0, изображение статично, при m=1 - включено мерцание. При

m=8 значение не изменяется.

FOR A=x TO y STEP c

Команда FOR - наиболее интересная команда, и обычно она плохо понимается, хотя и широко используется.

Задачи, стоящие перед интерпретатором, при обработке команды FOR, таковы:

а) удалить все переменные с тем же именем, и любые управляющие переменные с тем же именем;

б) добавить к существующим переменным новую управляющую переменную.

Эта переменная требует 19 адресов памяти. Первый адрес содержит имя переменной, соответствующим образом помеченное. Следующие 5 байт содержат начальное значение цикла, хранящееся как действительное число в 5 байтах. Следующие 5 адресов - конечное значение счетчика цикла, затем - 5 байт приращения. Если приращение не задано, они содержат единицу в плавающем виде. Последние 3 байта содержат информацию о цикле: первые два указывают номер строки, в которой записан оператор FOR, а последний - номер оператора в строке, увеличенный на единицу. Таким образом, после окончания цикла, будет выполняться следующий оператор, вне зависимости от того, находится ли он в той же строке, или нет.

Если начальное значение, конечное значение и шаг приращения - целые числа из интервала -65535 до +65535, то операции выполняются на 20% быстрее.

Следующая программа демонстрирует содержимое управляющей переменной команды FOR. Заметьте, что оператор FOR в строке 10 устанавливает пустую переменную под именем A, которая в дальнейшем в программе не используется.

```
10 FOR A=1.6 TO 2.1 STEP 0.1
20 LET V= PEEK 23627 + 256*PEEK 23628 -1
30 FOR B=1 TO 19
40 PRINT B, PEEK (V+B)
50 NEXT B
   RUN
```

Добавьте 25 LET V=V+25 и увидите контрольную переменную B.

в) третье действие, предпринимаемое интерпретатором, является весьма специфическим. В Спектрум не существует ошибок вызванных установлением шага в неправильном направлении. Это значит, что строка, подобная FOR A=2 TO 0 STEP 1 допустима. Однако, в действительности это приведет к полному игнориро-

ванию FOR-NEXT цикла. Это может быть показано изменением продемонстрированной выше программы следующим образом:

```
10 FOR A=1.6 TO 2.1 STEP -0.1
60 NEXT A
```

и программа будет успешно работать, но печати никакой не будет. Эта возможность - перепрыгивать через целые циклы - может быть преимуществом, ведь это позволяет распространить ограничение на 0, и может быть недостатком - ожидаемый результат не появится вообще.

FORMAT f

Используется с микроприводом

GO SUB n

Эта команда приводит к выполнению подпрограммы, которая начинается с номера n или с первого номера строки, имеющегося в программе и большего n. Переходы могут быть сделаны и REM-оператором, хотя делать этого, вероятно, не следует.

Команда GO TO помещает адрес перехода (номер строки) в системные переменные NEWPP и NSPPC, расположенные в 23618-23620 (5C42h-5C44h). Переходы к операторам, не являющимся первыми в строке, в обычной практике недопустимы.

IF x THEN s

В системе Спектрум значение нуль соответствует логическому значению "ложь", а любое значение, отличное от 0 - логическому значению "истина". Команда выполняет "s" и любые другие операторы в строке только, если значение x - "истина". Команда IF-THEN работает хорошо, но следующая небольшая программа написана, чтобы показать, используя повторное деление, что происходит, когда значение никогда не становится "ложью". Проблема, появляющаяся в этом случае, связана с получением в Спектруме величины 2**128.

```
10 LET A=1
20 IF NOT A THEN PRINT A:STOP
30 PRINT A
40 LET A=A/2
50 GO TO 20
   RUN
```

"A" будет распечатываться (более 5 экранов) до тех пор,

пока оно не достигнет значения 2**128. Однако, если строку 40 поменять на LET A=A*0.5, то значение "ложь" будет достигнуто.

INK n

Это третья из команд управления цветом. Когда используется самостоятельно в операторе Бейсика, ее действие постоянно, а когда находится в операторе печати - временно. Значение n может быть 0<= n <=7. Восемь основных цветов, используемых в Спектрум, закодированы и это ясно видно на клавиатуре. Использование INK 8 при печати дает цвет, хранящийся в атрибутах.

Использование INK 9, однако, немного более сложно: цвет чернил становится черным или белым в зависимости от цвета бумаги. Если цвет бумаги темный, т.е. черный, синий, красный или фиолетовый - цвет чернил - белый, а при светлом цвете бумаги - зеленом, голубом, желтом или белом - цвет чернил черный. Действие постоянной команды INK состоит: для N=0-7 устанавливаются биты 0-2 переменной ATTR-P; и для N=8 - устанавливается бит 5 переменной MASK-P. Если команда INK имеет временное действие, устанавливаются соответствующие биты временных системных переменных.

INPUT ...

Это очень мощная команда, позволяющая набором операторов INPUT совместно с элементами управления печатью добиваться желаемых результатов. Следующая программа показывает использование составных операторов INPUT. Заметьте, что команда INPUT работает с нижней (редакторской) областью экрана.

```
10 INPUT "Name please",A$; CHR$ 13; "Age please"; B$
20 PRINT AT 5,0; "Name"; TAB 7; A$
   CHR$ 13; "Age"; TAB 7; B$
```

Это четвертая команда управления цветом. Ее эффект может быть временным или постоянным. При INVERSE 1 цвета чернил и бумаги меняются местами, а при INVERSE 0 - нормальное значение цвета чернил и бумаги. При постоянном режиме по команде INVERSE устанавливается 5 бит P-FLAG, а при временной - 4.

LET v=e

Это основная команда в Бейсике. Переменная, выбранная пользователем, определяется интерпретатором либо как простая переменная, либо как управляющая переменная FOR-NEXT цикла, либо образует часть массива. Простая переменная может иметь любые существующие ссылки для переопределения и распределения нового места. Управляющая переменная и переменная массива будет размещена, но не переопределена. Далее определяется выражение, которое должно определить значение переменной и копируется в соответствующее место.

Все численные переменные имеют пять адресов, распределенные для определяемой величины. Эта величина затем сохраняется как число с плавающей запятой или целое. Простые строки имеют переменную длину, она представляет собой число адресов, отведенных для простой строки, и зависит от количества символов в строке в текущий момент. Все массивы строк имеют фиксированную длину, и строка, предназначенная для массива, будет уменьшаться, если она превышает длину элемента массива, и дополняться пробелами, если она короче фиксированной области, отведенной для нее.

Следующий пример показывает простой путь, которым можно проследить за областью переменной, и позволяющий пользователю ввести ряд переменных для наблюдения.

```
10 REM enter your variable.
20 LET V=PEEK 23627 + 256 - PEEK 23628
30 PRINT V, PEEK V: LET V=V+1: GO TO 30
```

Следует сделать несколько предложений для строки 10.

```
LET A=0
LET A=9E4
LET A$="AAA"
DIM A(2): LET A(1)=9E4
DIM A$(5):LET A$="1234567"
DIM A$(2,5): LET A$(1)="AAA"
```

LIST n

Эта команда допускает, что величина n должна быть равна нулю до тех пор, пока она не будет определена другим образом.

Если строка с номером n существует, она становится текущей, и помечается стрелкой >.

Интересной особенностью системы является то, что LIST 49172, например, будет такой же, как и LIST 20.

LLIST n

Эта команда посылает листинг программы, написанной на Бейсике, на принтер. Также n становится текущей строкой. Для тех читателей, которые не лишены чувства юмора, интересно использовать LIST <> 3 и LLIST <> 2 и включить их в программу.

LOAD

Команда LOAD имеет много различных оттенков, и позволяет загружать программы Бейсика, массивы переменных и блоки кодов.

Вообще, команда LOAD хорошо освещена в инструкции "Программирование на языке Бейсик", поэтому далее приводится обсуждение вопросов, которые не приведены в инструкции.

Информация (код или программа), записанные на магнитной ленте, содержатся в двух частях. Первая представляет собой 17-байтовый заголовок, вторая - "блок кодов".

Каждая из этих частей использует одинаковый формат единичных битов, сопровождаемых звуковым сигналом, который в два раза длиннее нулевых битов. В Спектрум используется головной маркер (+D0 для заголовков и +FF для блока кодов), и завершающий байт четности после заголовка и после блока кодов.

Блок заголовка разбит на пять частей следующим образом:

1. Одиночный байт информации, который равен "0" для программы Бейсика, "1" - для числового массива, "2" - для массива символов и "3" - для блока кодов.

2. Следующие 10 байт содержат название файла. Название, которое имеет более 10 символов, отвергается.

3. Два байта, содержащие общую длину блока кодов. В случае использования Бейсика, область программы и область переменных содержатся на магнитной ленте.

4. Два байта, которые для программы на Бейсике содержат начальный номер строки или начало для блока кодов.

5. Два байта, содержащие для Бейсик-программы длину области программы. "Блок кодов" просто загружается в требуемую область ОЗУ, как указано в заголовке.

LPRINT

Эта команда вызывает посылку элементов печати на принтер. Для дальнейшего обсуждения см. ниже команду PRINT.

MERGE f

Эта команда позволяет загрузить Бейсик-программу и ее

переменные с магнитной ленты и соединить с существующей программой и переменными. Там, где имеет место то же количество номеров строк в обоих программах, сохраняется только новая версия. Это относится также и к переменным с тем же названием и типом.

Команда выполняет обработку новой программы как блока данных, который должен быть загружен в рабочую область. Далее, области программ сравниваются строка за строкой, и новые строки копируются из рабочей области в основную область программы. Область программы "переопределяется" и "расширяется", если необходимо. Поскольку строки Бейсика соединены, подобная операция соединяет и область переменных.

MOVE f1, f2

Для использования с микроприводом.

NEW

Это команда повторного запуска системы без изменения системных параметров RAMTOP, P-RAMT, RASP, PIP, и UDG. Определенные пользователем графические символы, также не сбрасываются.

NEXT a

Эта команда должна рассматриваться как команда, выполняющая работу в цикле FOR-NEXT, в котором управляемая переменная установлена (см. FOR выше). Этапы, включающие в себя выполнение команды NEXT, следующие|

а) управляющая переменная расположена в области переменных, и величина STEP добавляется к VALUE, вне зависимости от того, является STEP положительным или отрицательным.

б) далее новая VALUE проверяется относительно LIMIT, но здесь следует обратить внимание на знак STEP. Если LIMIT превышается, то невозможна никакая дальнейшая организация циклов - команда NEXT закончена. Однако, если LIMIT не достигнут, выполняется дальнейшее прохождение цикла FOR-NEXT.

В Спектрум VALUE всегда увеличивается до того, как выполняется проверка ограничителя, следовательно, конечная VALUE после окончания цикла FOR-NEXT, будет всегда больше, чем LIMIT для отрицательной.

OPEN

Для использования с микродрайвом. Однако, возможно открывать и закрывать потоки данных следующим образом:

Используйте строку PRINT #5; "works?" и она не будет ра-

ботать до тех пор, пока поток данных не открыт использованием строки OPEN #5,"S". Строка CLOSE #5 закрывает поток. Следующие строки показывают ввод с клавиатуры и вывод на экран ТВ.

```
10 OPEN #5, "K"
20 INPUT #4; A$
30 OPEN #5; A$
40 PRINT #5; A$
```

OUT m,n

Эта команда позволяет пользователю посылать сигналы на выходной порт Спектрум из Бейсика.

В пункте BORDER (см. выше) было показано, как порт 254 управляет цветом рамки; разные цвета приводились для случаев от n=0 до n=7. В то же время, порт 254 также может использоваться для управления звуковым динамиком, и следующая программа показывает, как это делать. Программа интересно показывает, как возможно получить оценку времени, требующегося Спектрум для интерпретации оператора, так как быстрый оператор даст высокий звук, а медленный - ряд щелчков.

```
10 OUT 254,23: _____________ ; OUT 254,7; GOTO 10
```

В программе оператор, проверяемый пользователем, может быть, например, таким: PRINT;:, RAND;, LET A=0:, или POKE 00:.

В программе OUT 254,23 не используется звуковой динамик, а в OUT 254,7 - используется. Операция повторяется очень быстро, что дает звук.

В дальнейшем, при наличии устройств ввода/вывода, команда будет использоваться более часто.

OVER n

Это пятая из команд элементов цвета.

Ее действие может быть постоянным или временным. Если используется команда OVER 1, то последовательная печать будет выполняться об'единением по команде XOR с уже существующим символом. При выполнении XOR бит перебрасывается - установленный бит сбрасывается.

При использовании OVER 0 все биты в данной области символа будут графическими.

Заметьте, что при OVER 1, если символ или строка напечатаны дважды, они исчезнут, т.е.:

```
10 OVER 1: PRINT "A"; CHR$ 8; "A"
```

где CHR$ 8 - возврат на одну позицию в строке. Бит 1 переменной P-FLAG - управляющий флаг OVER.

PAPER n

Это шестая и последняя команда управления цветом. Она может быть постоянной или временной. Как и в случае INK, команда PAPER может использоваться от n=0 до n=9.

При n=8 устанавливается режим высвечивания, и цвет бумаги области передаваемого символа будет оставаться неизменным. Использование команды PAPER 9 является очень полезным при оформлении заглавий. Команда PAPER 9 устанавливает, что цвет бумаги в области вывода информации будет контрастным по отношению к используемому цвету чернил. Следовательно, при светлых чернилах - зеленых, голубых, желтых и белых - бумага будет черной, но при темных чернилах - черных, синих, красных и фиолетовых - бумага будет белой.

Фактическое действие "постоянной команды" PAPER является: для n=8 установить биты 3, 4, 5 переменной MASK-P, и для n=9 - установить бит 7 переменной P-FLAG, а для n=0-7 - установить биты 3, 4, 5 переменной ATTR-P.

PAUSE n

Эта команда HALT останавливает Z80 на n прерываний. Единственная работа, которая выполняется за время паузы, заключается в выполнении программы прерывания клавиатуры. Период паузы закончится после n прерываний или при нажатии клавиши.

PLOT x,y

Экран Спектрум представляет собой 256176 элементов изображения, каждый из которых может управляться отдельно командой PLOT. В обычной работе команда PLOT установит бит ОЗУ, соответствующий биту элемента изображения. Команда делает также этот элемент изображения текущим в системной переменной COORDS, который записывает координаты последнего элемента изображения.

Любой из цветовых элементов может использоваться в команде PLOT и использование OVER 1 может привести к "UNPLOT", а INVERSE 1 - к "NOPLOT".

POKE m,n

Эта команда позволяет пользователю выводить значения по адресам непосредственно в память Спектрум. Значения m может принимать от 0 до 65535, но ошибка не фиксируется, если по-

льзователь пытается ввести значения в ПЗУ. n может принимать значения от -256 до +255. Значения от 0 до 255 вводятся непосредственно, а к значениям от -255 до -1 прибавляется +256.

PRINT ...

Эта команда позволяет пользователю выводить символы на экран ТВ. Элементы отделяются друг от друга разделителями: ";" - элементы выводятся подряд, "," - выполняется табуляция к следующей половине строки, "/" - печать в начале следующей строки. Позицию экрана можно также указывать с помощью TAB и AT.

Элементы печати могут быть элементами цвета, действие которых будет временным, числовыми выражениями, или наборами символов. Заметьте, что все элементы цвета, разделители и управление табуляцией могут быть выполнены использованием соответствующих значений CHR$ и это может оказаться полезным.

RANDOMIZE n

Эта команда устанавливает значение системной переменной SEED, расположенной в адресах 23670 и 23671 (5C76,5C77). Если n не задано, значение для SEED берется из двух младших байтов FRAMES и может рассматриваться как случайное число. Если n задано, тогда это значение копируется в SEED. Для дальнейшего уточнения работы генератора псевдослучайных чисел см. ниже функцию RND.

READ v1, v2, ...

Эта команда используется вместе со списком DATA и может рассматриваться как несколько команд LET. Элементы в DATA являются выражениями, а элементы в READ - переменными, которым это выражение присваивается.

Сообщение об ошибке появляется, если нет соответствия между переменными и выражениями.

Заметьте, что в оператор READ можно включить неописанные ранее переменные. Системная переменная DATADD, расположенная в адресах 23639 и 23640 (5057 и 5058h) используется для указания в списке DATA. Первоначально указатель устанавливается на адрес начала области программы и сбрасывается в это значение по RUN.

REM

Это полезная команда, так как позволяет вносить коммен-

тарии. Вся строка после нее рассматривается как комментарий и не интерпретируется.

RESTORE n

Эта команда используется вместе со списком DATA и относится к одному или нескольким DATA.

Если n=0, то переменная DATADD указывает на адрес перед областью программы. Если n определено, то указывает на адрес строки Бейсик с номером n, если она существует или на ближайшую после этого номера.

Если n>9999 (самый большой разрешенный номер строки Бейсик), или "ушел" за последний номер программы, то DATADD указывает на последний адрес в области программы.

RETURN

По этой команде извлекается верхнее значение из стека GOSUB. Если обращение корректно, интерпретатор будет выполнять это утверждение следующим. Иначе будет выдано сообщение об ошибке: "RETURN" без "GOSUB".

RUN n

Эта важная команда позволяет пользователю выполнять программы Бейсик. Если n не задано, принимается значение 1. Когда n определено, интерпретатор находит область программы строки Бейсик с этим номером или первую строку после нее, если строка n не существует.

Команда RUN инициирует требуемые указатели путем выполнения RESTORE и CLEAR до интерпретации какой-либо строки.

SAVE

Эта команда хорошо описана в инструкции "Программирование на языке Бейсик", а подробности представлены после LOAD (см. выше).

STOP

По этой команде появляется сообщение "STOP". Использование CONTINUE как прямую команду позволяет затем продолжить программу со следующего оператора. Как и в любой ошибочной ситуации, команда STOP передает номер ошибки в системную переменную ERR-NR, которая располагается по адресу 23610 (5С3АН), номер ошибки всегда на 1 меньше, чем код ошибки, который появляется перед сообщением об ошибке.

Поскольку ошибка имеет место, интерпретатор останавли-

вает выполнение программы и переходит на программу обработки ошибки. Затем номер ошибки передается в ERR-NR, выбирается адрес возврата в командный режим (адрес расположен в 4367, или 1303H). Этот адрес возврата всегда существует и расположен в двух адресах ниже стека GOSUB. Если отсутствуют текущие циклы GOSUB, стек GOSUB будет пустым, но, т.к. вызываются различные подпрограммы, адрес возврата опускается в стек и затем извлекается опять, когда программа завершается. Адрес возврата всегда указывается системной переменной ERR-NR, которая расположена в адресах 23613 и 23614 (5C3DH и 5C3EH).

Заметьте, что команды RUN и GOTO не очищают стек GOSUB и, следовательно, если программа останавливается без завершения выполнения подпрограммы, тогда ERR-SP постепенно указывает на все более низкие адреса в памяти (на три байта для каждой подпрограммы). Однако, команда CLEAR восстанавливает значение ERR-SP.

VERIFY

Наличие этой команды вселяет уверенность пользователя в Спектрум. Команда VERIFY позволяет сверить с оригиналом любую программу, выведенную на магнитофон с помощью SAVE. Процесс проверки осуществляется для обеих частей программы - заголовка и блока данных. Ошибка "R" - ошибка при загрузке с ленты - появляется, если запись на ленте не точно соответствует оригиналу.

2.3. Функции Бейсика.

ABS

Все отрицательные числа становятся положительными, в то время, как положительные не изменяются.

ACS

Аргумент "x" берется как косинус и функция возвращает величину данного угла в радианах.

AND

Это двоичная операция, поэтому она требует два операнда. Если оба операнда логически "истинны", операция полностью "истинна". Однако, если один или оба логически "ложные", операция дает "ложь". Числовое выражение, которое является "ложным", будет иметь нулевую длину, т.е. пустую строку. Когда результат "истина", первый операнд возвращается по-

льзователю.

ASN

Аргумент берется как синус и функция возвращает величину данного угла в радианах.

ATN

Аргумент берется как тангенс и функция возвращает величину в радианах. ATN является одной из четырех функций,определенных в Спектрум с использованием полиномов Чебышева.

ATTR

Эта функция имеет форму ATTR (строка, столбец). Она возвращает пользователю значение байта атрибутов.

Функция эквивалентна

PEEK 22528 + строка*32 + колонка

Величина признака может рассматриваться следующим образом:

```
INK + PAPER*8 + BRIGHT*64 + FLASH*128
```

BIN

Это интересная функция, т.к. она позволяет любое целое число в диапазоне 0-65535 вводить в двоичной форме. Разрешается использовать любое число двоичных цифр до 16. Следующая строка Бейсика поэтому законна и дает числа от 1 до 10.

```
10 FOR A=SIN 1 TO BIN 1010 STEP 00000001:
   PRINT A: NEXT A
```

Хотя Спектрум позволяет ввести числа в двоичной форме, они не могут печататься в этой форме без использования Бейсик-программы.

CHR$

Эта функция возвращает пользователю символ, как строку для данного кода.

Следующая строка Бейсик показывает, как распечатать набор символов. Клавиши CONTINUE и ENTER должны будут использоваться 6 раз, чтобы установить коды управления цветом:

```
10 FOR A=0 TO 255: PRINT CHR$ A: NEXT A
```

CODE

Эта функция возвращает пользователю код первого символа в аргументе x$; код будет равен нулю, если x$ является нулевой строкой.

COS

Аргумент рассматривается в радианах,и функция возвращает соответствующий косинус для данного угла.

EXP

Для данного аргумента x функция возвращает величину "e" в степени "x", где e=2.718281828...

EXP является второй функцией, определенной в Спектрум с использованием многочленов Чебышева.

FN

В Спектрум около 26 числовых и 26 строчных функций. За знаком функции FN должна следовать одна буква или одна буква и знак $, а затем - требуемые аргументы, заключенные в скобки.

IN x

Аргумент x используется как адрес порта, и считываемое значение будет в диапазоне 0-255. Если порт не используется, величина будет 255.

INKEY$

Эта функция позволяет опрашивать клавиатуру без обращения к INPUT. Если нажата одна клавиша, то длина входной строки определяется этой клавишей. INKEY$ распознает как символ на клавише с CAPS LOCK или без нее, а также символ, нажатый с SYMBOL SHIFT. Другие комбинации дают пробел или "?". Если ни одна клавиша не нажата или нажата более, чем одна, длина входной строки равна 0.

INT

Для данной величины x эта функция возвращает только целую часть. У отрицательного числа прежде всего отбрасывается дробная часть, затем вычитается 1, чтобы "округлить" результат. Например, +4.66 даст +4, в то время как -5.66 получит выражение -5 и затем уменьшится до -6.

LEN

Эта функция определяет длину заданной строки. Нулевая строка дает нулевой результат.

Однако, заметьте, что элементы цвета, полученные нажатием клавиш, могут включаться в строку. Эти элементы будут иметь длину два.

LN

Для заданного аргумента x функция возвращает логарифм по основанию e для величины x.

Представляет собой еще одну функцию, вычисляемую в Спектрум с использованием полиномов Чебышева.

NOT

Это интересная функция, т.к. она является единственной функцией, которая дает "логическое" состояние значения в системе Спектрум. Однако, важно отметить, что результат инвертируется и что "истина", логический результат, дается довольно трудным просмотром "NOT NOT x".

Для x=2 NOT x даст "ложь", а NOT NOT x - "истина".

Для x=0 NOT x даст "истина", а NOT NOT x - "ложь".

OR

Является двоичной операцией и требует два числовых операнда. Если любой из операндов является логически "истинным", тогда операция - "истинная".

Однако, если ни один из операндов не является "истиной", операция дает "ложь". "Истинный" результат будет равен единице, если второй операнд не равен нулю, и величине первого операнда, если второй операнд равен нулю.

PEEK

Это полезная функция возвращает величину, находящуюся по адресу x. Результат будет всегда выражен целым числом в десятичном диапазоне 0-255 включительно.

PI()

Самая простая функция. Представление с плавающей запятой PI/2 является константой в "таблице постоянных" в ПЗУ. Эта величина выбирается, удваивается и передается пользователю. Величина PI приблизительно равна 3.141592653...

POINT

Эта функция имеет форму POINT (x-точка,y-точка), и возвращает величину 1, если элемент изображения в этом положении установлен, и величину 0, если сброшен.

RND

В Спектрум случайные числа выдаются генератором псевдослучайных чисел. Значение системной переменной SEED, находящейся в 23670 и 23671 (5076 и 5077), выбирается, модифицируется и возвращается при каждом обращении к функции RND. Случайное число, возвращенное пользователю, является новой величиной SEED, деленной на 65536.

Величина SEED сбрасывается в нуль при инициализации системы. С этого времени она является последовательностью целых чисел от 0 до 65535. Наконец, величина не будет повторяться до тех пор, пока все числа от 0 до 65535 не использовались.

Нижеприведенных два примера показывают эти особенности.

а) 10 PRINT RND: FOR A=1 TO 65535: POKE 0, RND: NEXT A: PRINT RND

Эта программа занимает 20 минут работы, но она показывает, что те же самые числа RND получаются каждые 65536 вызовов (POKE 0, RND - является фиктивным оператором).

```
10 POKE 23670,0: POKE 23671,0
20 LET SEED = 0
30 LET SEED = SEED + 1
40 LET SEED = SEED*75
50 LET SEED = SEED-INT (SEED/65537)*65537
60 LET SEED = SEED-1
70 PRINT SEED/65536, RND
```

Строка 10 устанавливает системную переменную SEED в нуль. Программа показывает, что такой же результат получается с помощью модифицированных шагов (строки 30-60), как и в случае вызова RND.

SCREEN$

Это самая интересная функция и очень слабо освещена в инструкциях. Форма этой инструкции - SCREEN$(строка,столбец) и она возвращает строку, содержащую символ для заданной позиции. Эта функция распознает символы с десятичными кодами 32-127 (20-7FH), когда они представлены в прямом или инвертированном виде.

Работа функции заключается в сравнении содержимого 8 байтов для области данного символа относительно 8 байтового

представления, имеющего место в генераторе символов. Функция возвращает нулевую строку, если символ не является ни одним из 96 символов генератора. Нежелательно, чтобы функция исключала из сравнения символы, определенные пользователем, т.к. вполне возможно, что они могут появиться. Однако будет весьма трудно проверить обычные графические символы, которые не хранятся в 8-байтовой форме, а создаются, когда требуется.

Несмотря на трудоемкую работу, которая включает в себя вычисление этой функции, она выполняется удивительно быстро. Это можно показать довольно просто следующим образом:

```
10 PRINT " "; SCREEN$ (0,0)
```

результат появится почти мгновенно.

SGN

Все положительные величины возвращают величину "+1". Все отрицательные - "-1", в то время, как нуль - дает "0".

SIN

Аргумент x рассматривается в радианах и функция возвращает соответствующий синус для данного угла. Эта функция является четвертой и последней, вычисляющейся по полиномам Чебышева.

SQR

Эта функция возвращает пользователю квадратный корень аргумента x. Альтернативой для SQR - использование "x**.5" и это покажет по существу каким образом Спектрум представляет SQR в своем калькуляторе. Квадратные корни отрицательных величин не вычисляются и дают сообщение об ошибке "неправильный аргумент".

STR$

Аргумент для этой функции рассматривается как числовое выражение. Операция STR$ включает в себя вычисление выражения, печать его в рабочей области так, как она будет выведена на экран ТВ, если требуется, и затем возвращает пользователю параметры строки.

TAN

Аргумент x рассматривается в радианах, и функция возвра-

щает соответствующий тангенс этого угла. Тангенс находится путем определения синуса и косинуса угла отдельно и последующего деления одного на другое. Вычисление тангенса поэтому занимает в 2 раза больше времени, чем определение косинуса или синуса.

USR

В Спектрум команда USR со следующим за ним числовым значением отличается при использовании от той же команды, но со строковым выражением.

USR-число

Эта важная функция может быть рассмотрена по существу как команда. Аргумент x, являющийся числовым выражением, принимается как адрес программы в машинных кодах, написанной пользователем. Когда "USR-число" вызывается, то Z80 останавливает выполнение программы монитора и вместо этого выполняет команды с адреса "x" и далее. Команда машинного кода RET вызовет возврат в программу монитора. Заметьте, что в программе, написанной пользователем в машинном коде, можно модифицировать регистр IY, но его содержимое необходимо восстановить перед возвратом в программу монитора. (Маскируемые прерывания должны быть запрещены, пока пара регистров IY содержит любую величину, отличную от +5C3AH). Однако, значение в альтернативной паре регистров HL должно быть сохранено и остановлено правильно, если требуется успешный возврат к Бейсику. Использование команды "USR-число" позволит получить значение, которое может быть возвращено. В Спектрум числовые значения передаются через пару регистров BC.

USR-строка

Эта функция возвращает адрес требуемого символа из графики, определенной пользователем. Аргумент "USR-строка" состоит из одного символа и этот символ должен быть в диапазоне "A-U" или "a-u".

В стандартной версии 16K графическая область, определяемая пользователем, будет обычно начинаться с адреса 32600 (7F58H), но, т.к. она может быть перемещена, лучше всегда рассматривать область, как область, начинающуюся с адреса, определяемого системной переменной UDG, которая располагается в адресах 23675 и 23676 (5C7B и 5C7C).

Если область графики, определенной пользователем, начинается с адреса 32600, то этот адрес задается с помощью USR-"A". Восемь адресов 32600 - 32607 содержат представление

символа "A", который будет "A" или как определит пользователь. USR "B" возвращает адрес 32608, USR "C" дает 32616 и так далее, для двадцати одного графического символа.

Простого метода определения символов еще нет, но многократное использование операторов DATA и BIN дадут, по крайней мере, прекрасную идею. Нижеприведенная программа иллюстрирует эту точку зрения.

```
10 DATA BIN 00000001
11 DATA BIN 00000010
12 DATA BIN 00000100
13 DATA BIN 00000100
14 DATA BIN 00001000
15 DATA BIN 01001000
16 DATA BIN 00101000
17 DATA BIN 00010000
18 FOR A=0 TO 7: READ B: POKE USR "A"+A,B: NEXT A
19 PRINT CHR$ 144: REM "A"
```

VAL

Эта функция требует строку в качестве аргумента. Строка затем рассматривается как числовое выражение, вычисленное и возвращенное пользователю. Эта функция по существу полезна, поскольку она позволяет вычислять выражения, которые содержатся в строках.

VAL$

Может показаться, что эту функцию всегда можно обойти. VAL$ требует строку в качестве аргумента, и эта строка должна быть заключена в кавычки, и даже простой оператор, содержащий VAL$, может иметь двойные или тройные кавычки. Функция затем возвращает пользователю результат в виде строки.

Как VAL, так и VAL$, могут в действительности оказаться более полезными, если они не дают сообщения об ошибке: "NONSENCE IN BASIC", которое имеет место, когда функция повреждена, но дает нулевой результат.

2.4. Управляющие символы.

В Спектрум имеется 11 управляющих символов, которые могут быть использованы.

Процедура управляющего символа возможно не очень полезна для программиста на Бейсике, но она может быть очень полезной, когда вызывается из программы в машинных кодах.

CHR$6 - печать запятой.

Печатаемая позиция устанавливается в положении "следующая половина экрана". Внутри оператора PRINT использование "CHR$6" и "," взаимозаменяемо.

CHR$8 - перемещение на одну позицию назад.

Печатаемая позиция модифицируется для возврата на один символ. Эта операция работает даже когда используется возврат к предыдущей строке экрана.

Особый случай использования CHR$8 позволяет "подчеркивать" как показано ниже.

```
10 PRINT "*"; CHR$8; OVER 1; "_"; CHR$0 и т.д.
```

CHR$9 - перемещение вправо

CHR$13 - следующая строка экрана

Печатаемая позиция устанавливается в начальное положение следующей строки. В операторе PRINT использование "CHR$13" и " " взаимозаменяемо, но использование CHR$13 до некоторой степени понятнее в листинге.

CHR$16 до CHR$21 - элементы цвета.

Эти управляющие символы дают "временные цвета". Они соответственно эквивалентны следующим операторам: INK, PAPER, FLASH, BRIGHT, INVERSE, OVER.

Когда один из этих управляющих символов элемента цвета используется, за ним должен следовать соответствующий "номер цвета".

Следующая строка показывает, как цвет INK установить в RED.

```
10 PRINT "XXXX", CHR$16; CHR$2; "YYYY"
```

CHR$22 - AT

Печатаемая позиция видоизменяется в позицию, данную следующими двумя символами. Первый символ дает номер строки, второй - номер столбца.

Следующая строка показывает эквивалентность оператора

```
10 PRINT AT 12,6; "Here"
```

оператору

```
10 PRINT CHR$22;  CHR$12;  CHR$6;  "Here"
```

CHR$ 23 - TAB

Печатаемая позиция устанавливается в столбец, заданный следующими двумя символами. Если текущий номер столбца равен или превышает новое значение, печатаемая позиция определяется на следующей строке экрана.

Особо обратите внимание на то, что TAB или в форме ключевого слова, или в форме управляющего символа дает знаки пробела от старой позиции до новой. Эта особенность может быть как совершенно бесполезной, так и очень полезной.

Нижеследующая программа показывает эту возможность.

```
10 FOR A=0 TO 20
20 PRINT PAPER 7; TAB RND 7;
   PAPER RND *6.5; TAB RND*20+9; A
30 NEXT A
```

Строка 20 может быть:

```
20 PRINT PAPER 7, CHR$23, CHR$ (RND 7); CHR$0;
   PAPER RND 6.5; CHR$ 23,
   CHR$ (RND 20 9); CHR$0; A
```

Глава 3. Микропроцессор Z80

3.1. Введение

Микропроцессор Z80 является наиболее важной микросхемой в микрокомпьютерной системе Спектрум, Z80 разработала фирма ZILOG INC (California. U.S.A.), и доказал, что он является наиболее удачным из когда-либо созданных микропроцессоров. В Спектрум использован микропроцессор Z80A, которые изготовляется по лицензии Zilog INC.

Обсуждение его возможностей следует разделить на две части. В первой части дана "физическая" точка зрения на микросхему, во второй - "логическая". Вслед за логической структурой будут рассмотрены машинные коды.

3.2. "Физический" взгляд на Z80.

Z80 - микросхема с 40 выводами, пронумерованными 1-40.

Рассмотрим функции выводов.

Выв. 11 - напряжение питания. Z80 требует одного источника +5v.

Выв. 29 - общий вывод.

Выв. 6 - тактовый вход. В Спектрум используется тактовая частота 3.5 Мгц, т.е. тактовый импульс следует каждые 0.000000286 сек.

Выв. 7-10, 12-15 - эти 8 линий образуют информационную шину, которая несет информацию как в микропроцессор, так и из него.

Выв. 1-5, 30-40 - эти 16 линий образуют адресную шину, которая доставляет адреса из микропроцессора в память.

Оставшиеся 13 выводов присоединены к линиям, которые несут контрольные сигналы.

Выв. 21 - линия считывания, RD. Эта линия становится активной, когда байт информации должен быть считан микропроцессором из памяти.

Выв. 22 - линия записи, WR. Эта линия становится активной, когда байт информации должен быть записан микропроцессором в память.

Выв. 19 - запрос памяти MREQ. Эта линия становится активной в тех случаях, когда байт информации должен следовать как из микропроцессора, так и в него.

Байт информации следует из памяти в соответствии с адресом, помещенным на адресной шине. Далее,в соответствии с откликом на сигналы RD и MREQ соответствующая микросхема памяти помещает байт информации на информационную шину, с которой эта информация в дальнейшем считывается в микропроцессор. Байт данных, записывается в память микропроцессором, помещает требуемые адреса на адресную шину. Шины MREQ, WR активизируются, и байт данных выставляется на шину данных. В дальнейшем, если к микросхеме памяти продолжается обращение, байт данных будет скопирован в память.

Выв. 28 - линия регенерации ОЗУ (RFSH). Эта шина используется для регенерации динамической памяти. В Спектрум эта линия частично используется для регенерации ТВ сканирующих сигналов.

Выв. 27 - выборка памяти M1. Наиболее важная линия, которая активизируется либо по машинному коду, либо по соответствующему байту данных, выбираемому из памяти.

Выборка инструкции или байта информации требует, чтобы все три линии M1,MREQ и RD активизировались. В то же время, выборка байта информации из ячейки памяти, которая является другой частью памяти, требует, чтобы только MREQ и RD были активны. Время, необходимое для выборки инструкции, составляет 1.14 мксек, что является 4- тактовыми циклами, или Т-состояниями.

Выв 20 - линия вход/выход IORQ. Эта линия активна при выполнении команды IN или OUT.

Выв. 18 - останов HALT. Линия активизируется при выполнении команды HALT.

Выв. 25 - линия запроса, BUSRQ. Z80 позволяет внешним устройствам использовать адресную и информационную шины в режиме пропуска цикла. Запрос микропроцессора пропустить следующий цикл выполняется внешними устройствами путем активизации этой линии.

Выв. 23 - линия подтверждения, BUSAK. Микропроцессор подтверждает запрос остановки выполнения дальнейших команд и активизирует эту линию.

Оставшиеся 4 вывода находятся под контролем пользователя.

Выв. 26 - линия "сброс", RESET. Эта линия используется для инициализации процессора. Она активизируется при включении питания. Сброс может быть использован в Спектрум соединением линий RESET и GND.

Выв. 24 - линия "ожидание", WAIT. "Медленная" память может требовать большего времени для циклов считывания или записи. И об этом сообщает микропроцессору путем активации линии WAIT.

Выв. 17 - "немаскируемое прерывание". NMI. Активация этой линии приводит к остановке выполнения микропроцессором

текущей команды. Вместо этого микропроцессор выполняет программу пользователя, специально составленную для этих целей. В Спектрум немаскируемое прерывание требует системного сброса, который выполняется записыванием 0 по адресу 23728.

Выв. 16 - "маскируемые прерывания", INT. В Спектрум сканирование клавиатуры и обмен в режиме реального времени называют "управляемым прерыванием". Это означает, что электроника системы каждые 1/50 сек активизирует INT, вызывая остановку выполнения микропроцессором основной программы и, вместо этого, выполнение подпрограммы сканирования клавиатуры. Способность Z80 реагировать на INT может управляться программистом специальными машинными командами.

3.3 Логический взгляд на Z80.

Внутренняя структура микропроцессора Z80 удивительно сложна, но, к счастью, может быть разделена на 5 функциональных частей. Это - устройство управления, регистр команд, программный счетчик, 24 регистра пользователя и арифметическое и логическое устройство. Обсудим каждую часть по очереди.

Устройство управления

Упрощенно его можно представить как "руководителя работ". В функции устройства управления входит пересылка байтов данных в Z80, который, в свою очередь, результаты (байты данных) посылает из процессора в правильном направлении и убеждается, что операция успешно завершена.

В Z80 устройство управления выдает огромное количество внутренних сигналов, которые проходят к различным частям внутренней структуры, а также управляющие сигналы на линии RD, WR, MREQ и т.д.

Важно подчеркнуть, что устройство управления подобно менеджеру производства - не оказывает влияния на вид работ, но только на их порядок. Z80 следует программе, которая написана программистом, а менеджер следует программе директоров компании.

Регистр команд (РК)

Термин "регистр" используется для описания простой ячей- внутри Z80. Это то место, где 8 бит байта могут держаться вместе. В Z80 имеется большой блок регистров и пересылки байтов данных к регистрам и от них, являются простейшей наиболее важной чертой программирования в машинных кодах.

РК - специальный регистр, где микропроцессор содержит копию выполняемой текущей команды. Одной из черт набора ко-

манд Z80 является то, что определенные инструкции содержатся в 2 байтах данных. В этих случаях РК содержит эти байты по-очереди. Во время выполнения программы РК содержит каждую команду по очереди.

Программный счетчик (ПС)

ПС - это не один, а пара регистров, которые используются совместно. Таким образом, ПС может содержать 16-битные значения. ПС имеет специфическую цепь в сохранении адресов, либо текущей выполняемой команды, либо следующей выданной команды, в зависимости от того, был ли изменен программный счетчик или нет.

Когда инструкция выбирается, устройство управления использует адреса в программном счетчике как адрес памяти, содержащей команду, которая должна выполняться следующей. И затем устройство управления наращивает программный счетчик.

Действие программного счетчика очень похоже на переменную PPC интерпретатора Бейсик, которая содержит номер текущей строки Бейсика и также наращивается.

Регистры пользователя (основные регистры)

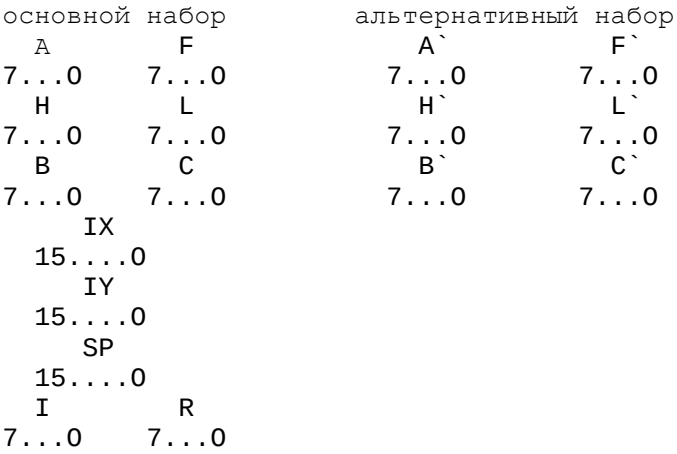
В Z80 существует 24 пользовательских регистра. Они названы пользовательскими, поскольку они могут быть заполнены информацией программистом (пользователем).

Имена, данные этим регистрам, не кажутся на первый взгляд логически обоснованными. Причина этого заключается в том, что Z80 развился из более ранних и менее сложных моделей. Некоторые из этих названий напоминают нам о первых, изготовленных в свое время микропроцессорах, тогда как в последующем новые названия были добавлены (к данному перечню) произвольно, причем некоторые из них оказались более подходящими и более информативными, чем остальные.

Все регистры - однобайтовые, но обычно они используются как пара регистров.

Диагр. 3.1 показывает 24 пользовательских регистра микропроцессора Z80, представленных как 12 регистровых пар. Номера битов также показаны.

Каждый из регистров коротко будет обсужден ниже.



Диагр. 3.1

Регистр A. Этот регистр является наиболее важным регистром Z80. Его часто называют аккумулятором. Название это пришло из моделей, в которых существовал единственный регистр для накопления результата.

В Z80 регистр A широко используется в арифметических и логических операциях, и, конечно, существует много операций, которые могут выполняться только используя A - регистр.

Существует огромное число различных возможностей помещения байта данных в регистр A и следующие. Существует много машинных команд, которые включают регистр A.

Регистр F. Это флаговый регистр, и он часто рассматривается как набор 8 флаговых битов, связанных вместе, а не как отдельный регистр.

Понятие флагов рассмотрено в главе 5, но теперь надо сказать, что флаг может быть установлен (т.е. содержать 1), или сброшен (содержать 0).

Флаговый регистр содержит 8 бит, но программист обычно имеет дело с четырьмя основными флагами. Это флаг нуля, флаг знака, флаг переноса и флаг четности/переполнения. Дополнительные флаги используются устройством управления и не могут быть использованы программистом непосредственно.

Регистровая пара HL. В ранних микропроцессорах существовал ординарный адресный регистр, который мог адресовать 256 адресов памяти. Однако, когда была введена двухбайтовая адресация, стало возможным адресовать 65536 адресов памяти непосредственно. В микропроцессоре, содержащем адресный ре-

гистр, один из регистров является старшим, а другой - младшим. Имена H и L в Z80 произошли от слов "high" и "low". Интересно отметить, что старший регистр, будучи более поздним достижением, привел к ситуации, в которой при адресации сначала указывается младшая часть, а затем старшая.

Память 64 К байт может быть рассмотрена как 256 страниц по 256 адресов в каждой, и в таком случае значение старшего байта указывает на используемую страницу.

В микропроцессоре регистровая пара HL является одной из трех регистровых пар, которые обычно используются в качестве адресных регистров. Однако, HL - пара является наиболее важной. HL-пара также может быть использована для хранения 16--битного числа, и существует ряд арифметических операций, которые могут быть выполнены над этими числами. H-регистр и L-регистр могут также использоваться как одинарные регистры, хотя с ними может быть выполнено ограниченное число операций.

Регистровые пары BC и DE. Эти регистровые пары используются, главным образом, как адресные регистры. Казалось бы, что их имена произошли единственно из-за существования A-регистра.Хотя, к DE ясно подходит аббревиатура слова "destination".

Все регистры могут быть использованы как одинарные, и особенно употребительно использование регистра B в качестве счетчика цикла.

Набор альтернативных регистров.

Z80 -специфический микропроцессор, так как он имеет альтернативный набор регистров для A, F, H, L, B, C, D и E. Эти альтернативные регистры обозначаются A`, F`, H`, L`, B`, C`, D`, и E`, произносятся как A прим и т.п.

Существует две специальные машинные команды, которые позволяют обменивать основной и альтернативный наборы регистров. После обмена Z80 будет работать с альтернативными регистрами, полагая, что это основной набор. Бывший основной, теперь будет использоваться как альтернативный.

Программист может обменивать наборы регистров (все вместе или частично) столько раз, сколько необходимо. Концепция альтернативных регистров кажется очень простой, но на практике использование альтернативных регистров может быть путаным. Самые большие проблемы те, что программист должен помнить, какой набор регистров используется, так как не существует машинных команд, которые работают на одном регистре и не работают на дополнительном ему.

Альтернативные регистры часто используются для сохранения среды, могла запускается независимая задача.

Особенным примером для программ Спектрум с 16К монитором является ситуация, когда при вычислении с плавающей запятой используются H и L регистры для сохранения адреса возврата. Поэтому, если эти регистры испорчены, то возвращение в Бейсик невозможно.

Регистровые пары IX и IY

Эти две регистровые пары используются для выполнения операций, которые включают индексацию. Это дает возможность работать со списком или таблицей. Начальный адрес списка или таблицы должен быть первоначально занесен в подходящую пару регистров IX или IY. В программе Спектрум регистр IY содержит адрес 23610 (5C3AH), указывающий на таблицу системных переменных. Регистр IX широко используется как указатель в программах обработки команд LOAD, SAVE, VERITY и MERGE.

Указатель стека

Это еще один адресный регистр. Он используется для указания в памяти области машинного стека и всегда рассматривается как цельный 2-х байтовый регистр.

Микропроцессор Z80 использует стек, заполненный в памяти сверху вниз. Аналогией может служить многоэтажный дом, в котором первый жилец поселяется в верхнем этаже, следующий-этажом ниже и т.д. Стек используется по принципу "последним пришел - первый вышел", так что первым жильцом, который выезжает, будет последний вселившийся.

Указатель стека используется для указания различных размещений в области стека в каждом случае. Указатель стека всегда содержит адрес, куда последний раз была произведена запись. Поэтому устройство управления сначала уменьшает значение указателя стека, а затем помещает туда значение. В системе Z80 каждая пересылка в стек или из него включает в себя 2 байта данных, и поэтому указатель стека должен дважды уменьшиться при помещении в стек, и дважды увеличен при выборе стека.

Машинный стек обычно используется микропроцессором как место сохранения адресов возврата из подпрограмм, но программист может помещать в стек и числовые данные, и таким образом использовать его как рабочую область. Однако, обычная ошибка программистов состоит в попытке использовать эти данные как адреса возврата, без проверки того, указывает ли стек на требуемую программу.

Регистр I

Это регистр вектора прерывания. В системах, основанных на Z80, и отличных от Спектрум, этот регистр обычно используется для размещения адресов обслуживания устройств ввода/-вывода. Однако, в Спектрум эта возможность не используется, и регистр I используется для генерации яркостного ТВ сигнала.

Регистр R

Это регистр регенерации памяти. Этот регистр является простым счетчиком, который увеличивается каждый раз при выполнении цикла регенерации. Значение в регистре циклически меняется от 0 до 255.

Регистр R используется для выполнения регенерации динамической памяти.

Арифметически-логическое устройство (АЛУ)

Эта часть Z80 имеет дело, как следует из названия, с арифметическими и логическими операциями. Важно понять, что операции, которые могут быть выполнены АЛУ, сильно ограниченны. Простое двоичное сложение или вычитание возможно, а умножение или деление - нет. Можно выполнить прибавление 1 или вычитание 1. Устройство способно выполнять большое число битовых операций и установить флаги, чтобы показать результат.

3.4. Структура программы в машинных кодах.

Как установлено выше, микропроцессор Z80 работает как компьютер, поскольку он представляет машину, способную запоминать последовательность хранимых команд. Программа должна существовать как набор машинных команд и данных, размещенных в последовательных адресах памяти. В микропроцессорных системах, основанных на Z80, единицей памяти является 8 бит или 1 байт данных. Поэтому программа в машинных кодах является последовательностью 8-битовых значений.

Десятичный результат может быть представлен следующей программой, которая печатает адреса с 0-7.

```
10 FOR A=0 TO 7
20 PRINT "LOCATION"; TAB 10; A; TAB 15; PEEK A
30 NEXT A
```

Программа на Бейсике, показывающая 16-ричные значения, немногим более сложна

```
10 FOR A=0 TO 7
20 LET H= INT (PEEK A/16)
30 LET L= PEEK A-H*16
40 PRINT "LOCATION"; TAB 10; TAB 15; CHR$(48+H+7 H 9)
   CHR$(48+L+7*(L  9))
50 NEXT A
```

Эти две программы включены, чтобы показать способ получения десятичного или 16-ричного листинга программы в машинных кодах.

Рассмотрим шаги получения документированного листинга.

Используя таблицу машинных команд (приложение F "BASIC programming") можно представить эти машинные команды в следующем виде:

мнемоника	комментарии
0 - DI	запрещение маскированных прерываний
1 - XOR A	сложение по модулю2 регистра A
2-4 - LD DE,+FFFF	загрузка регистровой пары DE
5-7 JP +11CB адр	переход по абсолютному адресу

В вышеупомянутом описании мнемонически записана каждая машинная команда. Мнемоника - стилизованный путь представления команд в полезной форме. Все машинные команды из набора команд Z80 имеют свою мнемонику и программы в машинных кодах записываются обычно с использованием мнемоники, а не двоичного, десятичного или 16-ричного представления.

Заметим, что в вышеупомянутом описании две строки программы используют по одному байту памяти, две следующие - по 3 байта. В последнем случае первый из 3 содержит код выполняемой операции.

Обычная форма для представления программ в машинных кодах дана в примере.

адрес	машинный код	мнемоника	комментарии
0000	F3	DI	запрет прерываний
0001	AF	XOR A	исключающее ИЛИ
0002	11 FF FF	LD DE,+FFFF	предельный адрес
0005	C3 CB 11	JP +11CB	переход вперед

Эта форма называется "форматом ассемблера", и обычно имеет адреса расположения кодов в 16-ричном представлении, машинные коды в 16-ричном представлении, мнемоническую запись команды и комментарии.

Вышеупомянутый пример показал, как может быть получен формат ассемблера для данного блока машинных кодов Z80. Этот

процесс называется дизассемблированием, а программа, решающая эту задачу - дизассемблером.

Операции, детально рассмотренные выше, приведены в обратном порядке по отношению к процессу написания программ.

Прежде всего, программист будет писать программу, используя мнемонику. Затем будут найдены двоичные, десятичные и 16-ричные эквиваленты. Программа, которая ставит в соответствие мнемонической записи машинные коды, называется ассемблер.

Отметьте, что формат ассемблера, приведенный выше, часто дополняется и включает в себя метки и переменные, а именно:

```
STARI equ        0000            нулевой адрес
START/NEW  equ   11CB            продолжать здесь
TOP-MEM    equ   FFFF            =65535

адрес            мнемоника
0000    START     DI               запрет прерываний
0001               XOR A            исключающее ИЛИ
0002               LD DE,+TOP-MEM   старший байт памяти
0005               JP START/NEW     переход вперед
```

При обсуждении формата ассемблера должно быть отмечено что он не является полностью определенным форматом, и что различные ассемблеры будут иметь различные требования и ограничения.

ГЛАВА 4. Математические основы программирования на машинном языке.

4.1. Введение.

В микропроцессорных системах, основанных на Z80, какой является СПЕКТРУМ, все данные передаются восьмибитовыми комбинациями.наиболее правильно описывать представление этих битов 8-ми битовыми двоичными числами. Но такое представление трудно для работы, и поэтому программисты, пишущие программы на уровне машинных кодов, обычно используют шестнадцатиричное представление.

В этой главе по очереди будут рассмотрены шестнадцатиричное представление, абсолютная двоичная арифметика, интегральное представление и представление чисел с плавающей запятой. Первые три раздела применимы к любым 8-ми битовым микропроцессорным системам, в то время как последние два специфичны для "СПЕКТРУМа".

4.2. Шестнадцатиричное кодирование.

Принципы, лежащие в основе шестнадцатиричного кодирования, состоят в описании чисел от 0 до 9 и добавлении 6 знаков, обозначаемых буквами от A до F. Следующая таблица показывает соответствие двоичного, десятичного и шестнадцатиричного представления для чисел 0 - 15.

двоичные	десятичные	шестнадцатиричные
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Как можно видеть из таблицы, одна шестнадцатиричная цифра записывается 4 битовыми двоичными. Поэтому восьмибитовое двоичное число записывается парой шестнадцатиричных знаков , а шестнадцатибитовое -- 4-мя шестнадцатиричными знаками.

Следующие примеры это иллюстрируют:

0000 0000 (двоичное) = 00 (шестнадцатиричное)
0100 1111 (двоичное) = 4F (шестнадцатиричное)
0000 0000 0000 0000 (двоичное) = 0000 (16-ричное)
0100 1100 1010 1111 (двоичное) = 4CAF (16-ричное).

Преобразование чисел из одной системы в другую для большинства людей трудно даже после месяцев и лет тренировки, но это очень полезное искусство. Тем не менее, очень просто написать программу, выполняющую эту работу.

Следующая программа на БЕЙСИКЕ переводит числа из десятичной системы в шестнадцатиричную:

```
 10 INPUT "Decimal number?", D
 20 IF D > 65535 THEN GO TO 170
 30 PRINT "Decimal", D
 40 DIM H(4)
 50 DIM H$(4)
 60 LET H(1)=INT (D/4096)
 70 LET D = D - H(1)*4096
 80 LET H(2) = INT (D/256)
 90 LET D = D - H(2)*256
100 LET H(3) = INT (D/16)
110 LET D = D - H(3)*16
120 LET H(4) = D
130 FOR A = 1 TO 4
140 LET H$(A) = CHR* H(A) + 48 + 7*H(A) + 9))
150 NEXT A
160 PRINT "Hexadecimal" H$
170 PRINT
180 GO TO 10
```

В программе десятичное значение последовательно уменьшается для поиска шестнадцатиричного эквивалента. Строки 130-150 переводят эти значения в код ASCII.

Следующая программа показывает шестнадцати-десятиричное преобразование:

```
 10 DIM H*(4)
 20 INPUT "Hex characters", H$
 30 IF CODE H* = 32 THEN GO TO 20
 40 LET D = 0
 50 FOR A = 1 TO 4
```

```
 60 IF H$(1) = CHR$32 THEN GO TO 100
 70 IF H$(A)<"0" OR H$(A)>"9" AND H$(A)<"A" OR H$(A)>"F"
    THNEN GO TO 120
 80 LET D=D+16**(4-A)*(CODE H$(A)-48-7*CODE H$(A)  57))
 90 NEXT A
100 PRINT "Hexadecimal", H$
110 PRINT "Decimal", D
120 PRINT
130 GO TO 20
```

В программе строка шестнадцатиричных знаков не всегда имеет 4 знака. Если значение не задано пользователем,требование всегда повторяется.

Последний пример в этом разделе показывает, как преобразование шестнадцатиричного в десятичное может быть выполнено вручную.

Например, шестнадцатиричное число: 789A

```
Десятичный эквивалент = 7 * 4096 = 28672
                        8 *  256 =  2048
                        9 *   16 =   144
                        A *    1 =    10
                        ----------------
                        789A     = 30874
                        ----------------
```

Или попарно:

```
Десятичный эквивалент = 78 * 256 = 30720
                        9A *   1 =   154
                        ----------------
                        789A     = 30874
                        ----------------
```

4.3. Абсолютная арифметика.

Одинарный регистр или адрес памяти содержит восьмибитовое двоичное число. Это число может быть в двоичном диапазоне 0000 0000 - 1111 1111, десятичном 0 - 256, или шестнадцатиричном 00 - FF. Ни в одном из этих случаев число не может быть ни отрицательным, ни дробным, и это образует основу абсолютной двоичной арифметики -- значение в памяти или регистре всегда положительное и целое.

Также важно представлять себе, что значение, содержащееся в памяти или регистре изменяется циклически в пределах 0 - 255. Это означает,что, если в операции сложения полученный результат превышает 255, окончательный результат записывается после вычитания 256; при вычитании к значению меньше нуля прибавляется 256.

Это показано на следующих примерах:

```
    252 + 44     дает    40
```

или

```
    FC  + 2C     дает    28
    87 - 200     дает   143
```

или

```
    57 - C3      дает    8F
```

Флаг переноса действует в большинстве операций, требующих переноса. Для дальнейшей детализации см. главу 5.

В системах, основанных на Z80, все числа представлены в абсолютной двоичной арифметике, но программист часто нуждается в различных интерпретациях чисел, таких, как положительное или отрицательное целое или дробное. Следующие три раздела главы рассматривают использование в системе "СПЕКТРУМа" различных видов интерпретации чисел.

4.4. Дополнительная арифметика.

Принцип дополнительной арифметики очень прост. Но при использовании ее в программах результат может быть очень неожиданным.

Метод позволяет программисту рассматривать двоичные значения в интервале 0000 0000 - 0111 1111 как эквивалент десятичных значений 0 - 127, а двоичные значения в интервале 1000 0000 - 1111 1111 как эквивалент десятичных значений -128 - -1

В результате такой интерпретации бит 7 (левый крайний бит восьмибитового числа рассматривается как знаковый. Он будет нулевым для положительных чисел, и будет иметь значение 1 для отрицательных.

На диагр. 4.1. показывается принцип дополнительной арифметики для восьмибайтовых чисел.

Заметим, что возможно распространить принципы дополнительной арифметики и на шестнадцатибитовые числа. В этом случае интервал возможных значений чисел будет от -32768 до 32768 десятичных.

Преобразование отрицательных десятичных чисел в дополнительную двоичную или шестнадцатиричную форму просто, но часто легче сослаться на соответствующую таблицу.

1. Найдите двоичную форму для абсолютного двоичного значения, например -54 -- 0011 0110.

2. Получите обратный код - надо изменить нули на единицы и обратно: 0011 0110 -- 1100 1001.

3. Прибавьте единицу.

   1100 1001 + 1 = 1100 1010.

4. Используйте двоичную форму или переведите его в шест-

надцатиричную: -54 в дополнительном коде: 1100 1010 или CA.

Операции можно провести в обратном порядке для перевода числа в дополнительном коде в десятичный вид.

4.5. Интегральное представление.

Интерпретатор Бейсика в системе СПЕКТРУМа использует 5 байт для представления чисел. Целые числа, например, находящиеся в интервале -65536 -- +65536 включительно записывают в целой форме, в то время как дробные или целые вне этого диапазона записываются в 5 байтах числа с плавающей запятой.

В целой форме первый байт всегда нулевой. Второй байт содержит 0, если число положительное, и 255 (FFh), если оно отрицательное. Третий и четвертый байты содержат целое значение как шестнадцатибитовое беззнаковое число в дополнительном коде, но третий байт содержит всегда младшую часть, а четвертый -- старшую. Пятый байт не используется, но всегда содержит 0.

Следующая программа демонстрирует целую форму (в десятичном виде для любых целых чисел, вводимых пользователем. Строка 20 программы гарантирует получение целой формы).

```
10 INPUT N
20 IF N <> INT N OR N < -65535 OR N > 65535 THEN GO TO 10
30 PRINT "Number chosen=", N
40 LET V = PEEK 23627 + 256 * PEEK  23628
50 FOR A = 1 TO 5
60 PRINT A; "."; TAB 5; PEEK (A + V)
70 NEXT A
80 GO TO 10
```

Диаграмма 4.1.

```
   двоичное      десятичное      16-ричное
   0111 1111       +127              7F
   0111 1110       +126              7E
Положительные значения
   0000 0010        +2               02
   0000 0001        +1               01

   0000 0000         0               00

   1111 1111        -1               FF
   1111 1110        -2               FE
Отрицательные значения
   1000 0001       -127              81
   1000 0000       -128              80
   ^
   L знаковый бит
```

Диагр. 4.1

В приведенной выше программе указатель указывает на начало поля переменных и адреса с +1 до +5 будут содержать 5 байт введенного числа.

Программа дает результаты в виде:

```
Выбрано значение 0
1. 0
2. 0
3. 0
4. 0
5. 0
так как 0 положителен, представляется он как 0*1 +
0*256.
Выбрано значение 1516
1. 0
2. 0
3. 236
4. 5
5. 0
так как 1516 --положительное, представляется оно
как 236* 1 + 5*256
Выбрано значение -1
1. 0
2. 255
3. 255
4. 255
5. 0
```

так как -1 -- отрицательное и представляется оно FFFF.

Замечание: ошибкой программирования будет перевод 0 как -65536. Это значение не соответствует целой форме 0,255,000 (см. приложение IY).

4.6. Представление с плавающей запятой.

Пять байтов, используемых в СПЕКТРУМе для предоставления числа с плавающей запятой, позволяют хранить числа в диапазоне 29E-38 - 1.7E38 (приблизительно).

Нулевое значение всегда хранится в памяти как пять нулевых байтов. Все остальные значения хранятся: экспонента в первом байте, мантисса -- в остальных четырех. Теория описания чисел в терминах экспоненты и мантиссы первоначально появилась в десятичной записи,поскольку в ней легче работать

Нахождение экспоненты и мантиссы для десятичного числа это, в сущности, перевод числа в E-формат. Рассмотрим число 12345, которое в E формате может быть представлено как .12345E+5.Для получения мантиссы десятичная точка сдвигается влево до тех пор, пока она установится перед старшей знача-

щей цифрой. Экспонента -- число необходимых сдвигов, мантисса -- дробное десятичное число. Экспонента равна +5; мантисса -- .12345.

Поскольку система СПЕКТРУМ имеет дело скорее с двоичными числами, чем с десятичными, подобные операции могут быть рассмотрены для двоичного числа.

Рассмотрим двоичное число 0001 1111, что эквивалентно +31.

Двоичная точка лежит справа от 3 бита и требует 5 сдвигов, чтобы перевести ее перед старшим значащим битом. Поэтому экспонента = +5, мантисса = .1111 1000.

В СПЕКТРУМе после этого производится еще некоторые действия. Так, для экспоненты: правильная экспонента, как выше +5 увеличивается на десятичное значение +128 (16-ричное 80). В нашем примере +5 + 128 = 133.

Для мантиссы:

первый бит мантиссы всегда будет единичным, поэтому его можно использовать как знаковый -- для положительного числа он будет нулевым, для отрицательного -- единичным. В нашем примере мантисса становится равной .0111 1000.

И теперь остается поместить число в 5 байтов. Неиспользуемые байты мантиссы обнуляются.

В десятичном виде для плавающей арифметики +31 записывается как 133,120,000

В шестнадцатиричном как 85,78,00,00,00.

И, если хотите, в двоичном 1000 0101 0111 1000 0000.

Подобная операция позволяет переводить отрицательные и дробные числа, хотя и не так просто.

Следующая программа показывает десятичную форму плавающих чисел. Включение строки 30 гарантирует, что не всегда получится плавающая форма числа.

```
 10 INPUT N
 20 IF N = 0 THEN GO TO 40
 30 LET N=N+.2E-38
 40 PRINT "Number chosen =";N
 50 PRINT
 60 PRINT "Exp="; TAB 9; "Mantissa"
 70 LET V=PEEK 23627 +256*PEEK 23628
 80 PRINT PEEK (V+1); TAB 9;
 90 FOR A=2 TO 5
100 PRINT PEEK (V+1); CHR$32
110 NEXT A
120 PRINT
130 GO TO 10
```

Приведенная выше программа даст следующие результаты:

```
1 = 129 0000
2 = 130 0000
35456 = 144 1012800
-1    = 129 128000
-35456= 144 13812800
0.333 = 131 74167239158
```

Интересно заметить, что .5 и 1/2 дадут различные значения (см. приложение IY для дальнейшей детализации).

```
.5  = 127 127255255255
1/2 = 128 000
```

ГЛАВА 5. Система команд микропроцессора Z80.

5.1. Команды и данные.

Теперь мы достигли стадии, когда можно обсудить команды машинного языка.

В этой книге все команды разделены на 18 групп, каждая из которых состоит из команд, имеющих много общего. Однако, прежде чем обсуждать эти команды, рассмотрим 6 классов данных, которые могут следовать за командой.

1. Однобайтовая константа (+dd).

Это число A в диапазоне 00-FFh, 0-255 десятичное. Те команды, которые требуют за собой однобайтовую константу имеют мнемоническую приставку +dd, например, LD A,+dd.

2. Двухбайтовая константа (+dddd).

Это число A в диапазоне 0000-FFFFh (десятичное 0-65535). Те команды, которые требуют за собой двухбайтовую константу, имеют мнемоническую приставку +dddd, например, LD HL,+dddd.

3. Двухбайтовый адрес (addr).

Это число A в диапазоне 0000-FFFFh (десятичное 0-65535), т.е. число, которое используется как адрес в памяти. Те команды, которые требуют за собой двухбайтовый адрес, имеют мнемоническую приставку addr, например, JP addr.

4. Однобайтовая константа смещения (e).

Это число A в диапазоне 00-FFh, -128 - +127 десятичное. Число всегда представлено в дополнительной арифметике. Те команды, которые требуют за собой однобайтную константу смещения, имеют мнемоническую приставку e, например, JP e.

5. Однобайтовая индексирующая константа смещения (+d).

Это число A в диапазоне 00-FF (десятичное -128- +127). Число всегда представлено в дополнительной арифметике. Те команды, которые требуют за собой однобайтовую индексирующую константу, имеют мнемоническую приставку +d, например, LD A, (IX-d).

6. Однобайтовая индексирующая константа смещения и однобайтовая константа (+d, +dd).

Т.е. числа в диапазоне 00-FFh (первое из которых рассматривается как десятичное -128 - +127, а второе - как десятичное 0-255.

Команды, требующие за собой двух байт данных, сопровождаются мнемониками +d и +dd.

5.2. Группы команд.

Существует много путей для разделения на группы сотен различных машинных команд. Однако, метод, выбранный здесь, разделяет команды на 18 функциональных групп.

После изучения команд по группам, читатель выполнит программы БЕЙСИКА из следующей главы, которые иллюстрируют эти команды.

Группа 1. Команда НЕТ ОПЕРАЦИИ.

Мнемоника	Шестнадцатиричный код
NOP	00

Выполнение команды в Z80 требует 1.14мкс. Ни один из регистров или флагов не меняется. Команда НЕТ ОПЕРАЦИИ используется программистом для организации задержек, но чаще для удаления ненужных команд из программы.

Группа 2. Команды загрузки регистров константами.

Мнемоника	Шестнадцатиричный код
LD A,+dd	3E dd
LD H,+dd	26 dd
LD L,+dd	2E dd
LD B,+dd	06 dd
LD C,+dd	0E dd
LD D,+dd	16 dd
LD E,+dd	1E dd

Каждая строка этих команд требует двух байт памяти: один для кода операции,второй для константы. Команды записывают в регистры соответствующие значения. Старые значения регистров пропадают.

Следующие команды заключают загрузку пары регистров двухбайтовыми константами:

Мнемоника	Шестнадцатиричный код
LD HL,+dddd	21 dd dd
LD BC,+dddd	01 dd dd
LD DE,+dddd	11 dd dd
LD IX,+dddd	DD 21 dd dd
LD IY,+dddd	FD 21 dd dd
LD SP,+dddd	31 dd dd

Строка команды требует три или четыре байта памяти. Код операции занимает один или два байта памяти и два байта занимает константа.

Первый байт константы всегда загружается в младший ре-

гистр пары, т.е. L, C, E, X, Y, или P, а второй байт -- в старший регистр, т.е. H, B, D, I, или S.

Эти команды записывают в регистровые пары значения данных. Эти значения часто рассматриваются программистом как двухбайтовый адрес, но могут быть также и двухбайтовым числовым значением и двумя отдельными однобайтовыми числовыми значениями.

Группа 3. Команды копирования регистров и обмена.

В Z80 существует 59 команд, которые выполняют копирование регистров и регистровых пар. Эти команды лучше всего делить на 4 подгруппы.

Подгруппа a. Команды копирования регистр-регистр.

Следующая таблица даст коды операций команд, выполняющих копирование содержимого регистра в другой регистр

	LD	LD	LD	LD	LD	LD	LD
A	7F	67	6F	47	4F	57	5F
H	7C	64	6C	44	4C	54	5C
L	7D	65	6D	45	4D	55	5D
B	78	60	68	40	48	50	58
C	79	61	69	41	49	51	59
D	7A	62	6A	42	4A	52	5A
E	7B	63	6B	43	4B	53	5B

Ни одна из команд, приведенных в таблице, не изменяет состояния флагов. Существует, кроме этого, 4 команды для I и A регистров.

Мнемоника	Шестнадцатиричный код
LD A, I	ED 57
LD A, R	ED 5F
LD I, A	ED 47
LD I, R	ED 5F

Эти последние 4 команды влияют на флаг переполнения четности.

Подгруппа b. Команды копирования регистровая пара - регистровая пара.

Существуют только три команды этой подгруппы и они все копируют значения в указатель стека.

Мнемоника	Шестнадцатиричный код
LD SP, HL	F9
LD SP, IX	DD F9
LD SP, IY	FD F9

Эти команды не изменяют состояния флагов.

Заметьте, что если содержимое регистровой пары надо копировать в другую регистровую пару и эти команды не подходят, то необходимо выполнить две команды копирования регистр-регистр. Например, нет команды LD HL, DE и ее заменяют используя LD H,D и LD D,E. Или содержимое первой регистровой пары может быть сохранено в стеке и впоследствии переписано во вторую регистровую пару (см. команды стека на стр.103).

Подгруппа c. Команды EX DE, HL.

В системе команд Z80 существует только одна команда, которая позволяет обменивать содержимое регистров в пределах основного набора регистров.

Мнемоника	Шестнадцатиричный код
EX DE,HL	EB

Это очень полезная команда, позволяющая программисту обменивать значение в паре регистров DE с содержимым пары HL. Значения флагов не изменяется. Команда обычно используется, когда необходимо, чтобы адрес или двухбайтовая числовая константа из пары DE была записана в пару HL, но содержимое последней не было утеряно.

Подгруппа d.Команды альтернативного набора регистров

В этой группе две команды

Мнемоника	Шестнадцатиричный код
EXX	D9
EX AF, A'F'	08

Команда EXX вызывает переключение регистров H, L, B, C, D и E на регистры H', L', B', C', D', и E'.

Команда EX AF, A'F', как следует из мнемоники, переключает регистры A и F на A' и F'.

Альтернативный набор регистров часто используется для хранения адресов и данных. Помещенные в альтернативные регистры, эти значения сохраняются без искажения и могут быть легко и быстро восстановлены.

Группа 4. Команды загрузки регистров из памяти.

В систему команд Z80 входит много команд, позволяющих находить данные в памяти и затем загружать их в основной регистр. Все эти команды требуют, чтобы программист указал адрес или пару адресов байтов, откуда данные должны быть скопированы, и регистр (или пару регистров) -- получателей данных.

Команды этой группы лучше рассматривать как команды трех подгрупп в соответствии с методом адресации выбранным программистом.

Возможны три вида адресации:

a) непосредственная адресация -- действительный адрес указан в двух байтах вслед за кодом операции;

b) косвенная адресация -- двухбайтовый адрес уже размещен в адресной регистровой паре;

c) индексная адресация -- адрес данных должен быть вычислен путем сложения значения смещения d с основным адресом, уже содержащимся в регистровой паре IX или IY.

Подгруппа a. Команды использующие непосредственную адресацию.

Мнемоника	Шестнадцатиричный код
LD A, (addr)	3A addr
LD HL, (addr)	2A addr
LD BC, (addr)	ED 6B addr
LD DE, (addr)	ED 4B addr
LD IX, (addr)	DD 2A addr
LD IY, (addr)	FD 2A addr
LD SP, (addr)	ED 7B addr

Команда LD A, (addr) -- единственная в Z80, которая позволяет содержимое ячейки, заданное непосредственной адресацией загрузить в одинарный регистр.

Важно заметить, что шесть команд -- действительно двойные, т. е. команда LD BC, (addr) могла бы состоять из LD C, (add r), следующей за LD B, (addr+1).

В каждом случае содержимое адреса копируется в младший регистр, а содержимое следующего адреса -- копируется в старший.

Подгруппа b. Команды, использующие косвенную адресацию

Мнемоника	Шестнадцатиричный код
LD A, (HL)	7E
LD A, (BC)	0A
LD A, (DE)	1A
LD H, (HL)	66

LD L, (HL)	6E
LD B, (HL)	46
LD C, (HL)	4E
LD D, (HL)	56
LD E, (HL)	5E

В каждом случае адрес байта данных, который должен быть скопирован, уже находится в паре регистров HL, DC или BC.

Если программист хочет выполнить команду, которой нет в этом списке, необходимо использовать взамен инструкцию, например, команда LD D, (BC) не разрешена, возможна команда LD A, (BC) и LD D,A, которая меняет содержимое регистра A. Или: LD H,B; LD L,C и LD D,(HL) которая меняет содержимое пары HL.

Подгруппа c. Команды, использующие индексную адресацию.

Команды этой подгруппы позволяют программисту загружать динарные регистры байтами данных, содержащихся в таблице, списке или наборе данных. Основной адрес содержится в соответствующей индексной паре.

Команды подгруппы:

Мнемоника	Шестнадцатиричный код
LD A, (IX+d)	DD 7E d
LD H, (IX+d)	DD 66 d
LD L, (IX+d)	DD 6E d
LD B, (IX+d)	DD 46 d
LD C, (IX+d)	DD 4E d
LD D, (IX+d)	DD 56 d
LD E, (IX+d)	DD 5E d

Для команд, использующих пару IY, надо заменить IX на IY и DD на FD. Интересно рассмотреть время, требуемое Z80 на выполнение команд этой группы. Самые быстрые команды - это те, которые указаны в подгруппе b. Эти команды требуют Z80 выбрать один байт кода операции и затем фактически байт данных. Команда LD A,(HL), например, требует только 7 тактовых циклов.

Команды подгруппы a гораздо более сложные и требуют, в зависимости от вида 16-20 тактовых циклов. Команды подгруппы c также требуют много времени - 19 тактовых циклов - индексная адресация требует вычислений.

Ни одна из команд этой группы не меняет флагов.

Группа 5. Команды записи в память содержимого регистров или констант.

Обычно команды этой группы выполняют операции, противоположные тем, которые выполняют команды группы IY.

Команды позволяют содержимое регистров переписывать в память или записывать туда константы.

Эти команды также удобнее рассматривать по трем подгруппам.

Подгруппа d. Команды непосредственной адресации.

Мнемоника	Шестнадцатиричный код
LD (addr), A	32 addr
LD (addr), HL	22 addr (обычная форма)
	ED 63 addr (необычная)
LD (addr), BC	ED 43 addr
LD (addr), DE	ED 53 addr
LD (addr), IX	ED 22 addr
LD (addr), IY	FD 22 addr
LD (addr), SP	ED 73 addr

Приведенные команды выполняют только непосредственную адресацию, и важно заметить, что нет команд для записи в память констант. Если это необходимо, константа должна быть загружена в регистр A, или каким либо другим путем. Затем, команды типа LD (addr), HL - фактически двойные команды: LD (addr), L и LD (addr+1),H.

Команды подгруппы часто используются для сохранения адреса и значений в памяти, когда те значения рассматриваются как переменные. Например, часто используется LD (RAMTOP),HL, где RAMTOP - метка пары последних адресов в памяти. Выборка текущего значения может быть позднее выполнена командой 4 группы, например LD HL, (RAMTOP).

Подгруппа b. Команды косвенной адресации.

Команды этой подгруппы позволяют копировать содержимое регистров в память, адрес которой содержится в регистровой паре HL, BC или DE. Существует также команда записи однобайтовой константы по адресу, указанному в паре HL.

Мнемоника	Шестнадцатиричный код
LD (HL), A	77
LD (BC), A	02
LD (DE), A	12
LD (HL), H	74
LD (HL), L	75

LD (HL), B	70
LD (HL), C	71
LD (HL), D	72
LD (HL), E	73
LD (HL), +dd	36 dd

Подгруппа C. Команды индексной адресации.

Мнемоника	Шестнадцатиричный код
LD (IX+d), A	DD 77 d
LD (IX+d), H	DD 74 d
LD (IX+d), L	DD 75 d
LD (IX+d), B	DD 70 d
LD (IX+d), C	DD 71 d
LD (IX+d), D	DD 72 d
LD (IX+d), E	DD 73 d
LD (IX+d), +dd	DD 36 dd

Для команд, использующих IY - регистровую пару надо изменить IX на IY и DD на FD.

Группа 6. Команды сложения.

Эта группа - первая из четырех групп командz80, выполняющих арифметические и логические операции.

Команды сложения позволяют программисту прибавить (в абсолютной двоичной арифметике) заданное число к регистру, паре регистров или индексному адресу памяти.

Команды этой группы могут быть разделены на три группы, каждая из которых содержит собственное мнемоническое имя:

a) команды ADD

b) команды INK. Специальный случай сложения, когда к числу прибавляется единица.

c) команда ADC. Значение флага переноса прибавляется к результату. Флаг переноса - это один из битов регистра флагов, и используется для сигнализации о том, было ли при выполнении последней арифметической операции двоичное переполнение регистра или байта памяти. Команды ADD и ADC меняют флаг переноса, а INK - команды не меняют - факт, имеющий иногда определенные преимущества.

Мнемоника	16-ричный код	Мнемоника	16-ричный код
ADD A,+dd	C6 dd	ADD HL,HL	29
ADD A,A	87	ADD HL,BC	09
ADD A,H	84	ADD HL,DE	19
ADD A,L	85	ADD HL,SP	39
ADD A,B	80	ADD IX,IY	DD 29

ADD A,C	81	ADD IX,BC	DD 09
ADD A,D	82	ADD IX,DE	DD 19
ADD A,E	83	ADD IX,SP	DD 39
ADD A,(HL)	86		
ADD A,(IX+d)	DD 86 d		

Для команд, использующих IY регистр надо поменять IX на IY и DD на FD. Команды ADD, приведенные выше, все очень просты. Команда устанавливает или сбрасывает флаг переноса в зависимости от того, было или нет двоичное переполнение влево регистра или пары регистров.

Это иллюстрируют следующие примеры:

1. регистр A=60h
   B=90h
   выполняется ADD A,B
   A=F0h
   B=90h флаг переноса сброшен;
2. регистр A=A3h
   B=7Eh
   выполняется ADD A,B
   A=26h
   B=7Eh флаг переноса установлен.

Подгруппа b. Команда ADC.

Мнемоника	16-ричный код	Мнемоника	16-ричный код
ADC A,+dd	CE dd	ADC HL,HL	ED 6A
ADC A,A	3F	ADC HL,BC	ED 4A
ADC A,H	8C	ADC HL,DE	ED 5A
ADC A,L	8D	ADC HL,SP	ED 7A
ADC A,B	88		
ADC A,C	89		
ADC A,D	8A		
ADC A,E	8B		
ADC A,(HL)	8E		
ADC A,(IX+d)	DD 8E d		
ADC A,(IY+d)	FD 8E d		

Команды этой подгруппы позволяют программисту сложить два числа вместе с текущим значением флага переноса.

Все команды этой подгруппы меняют флаг переноса. Он сбрасывается, если операция ADC не дает десятичного переполнения и устанавливается, если дает.

Группа 7. Команды вычитания.

Команды вычитания позволяют программисту вычесть (в абсолютной арифметике) заданное число из ординарного регистра, пары регистров или индексного регистра памяти.

Снова команды этой группы могут быть разделены на 3 подгруппы, каждая из которых имеет свою мнемонику.

Подгруппы:

a) команды SUB:
b) команды DEC. Специальный случай вычитания, когда из числа вычитается единица;
c) команды SBC. Значение флага переноса вычитается из результата.

Все команды SUB и SBC изменяют флаг переноса в зависимости от того, требовался ли двоичный заем. Команда DEC оставляет флаг переноса без изменений.

Подгруппа a. Команды SUB.

Мнемоника	Шестнадцатиричный код
SUB +dd	D6 dd
SUB A	97
SUB H	94
SUB L	95
SUB B	90
SUB C	91
SUB D	92
SUB E	93
SUB (HL)	06
SUB (IX+d)	DD 96 d
SUB (IY+d)	FD 96 d

Замечание: мнемоника команды, хотя и пишется так, как указано выше, но SUB L подразумевает SUB A,L и т. д., поскольку все команды используют регистр A.

В Z80 команда SUB выполняет "истинное" абсолютное двоичное вычитание, как показано на примерах. Флаг переноса сбрасывается, если первоначальное значение регистра A больше или равно вычитаемому, и устанавливается, если меньше.

```
регистр A=DC          регистр A=32
            SUB B                   C флаг =0
регистр B=AA          регистр B=AA
регистр A=AA          регистр A=CE
            SUB B                   C флаг =1
регистр B=DC          регистр B=DC
```

Подгруппа b. Команды DEC.

Команды этой подгруппы позволяют вычесть 1 из содержимого восьмибитового регистра, ячейки памяти или 16-битовой пары регистров.

Во всех случаях флаг переноса не изменяется.

Мнемоника	16-ричный код	Мнемоника	16-ричный код
DEC A	3D	DEC HL	2B
DEC H	25	DEC BC	0B
DEC L	2D	DEC DE	1B
DEC B		DEC SP	3B
DEC C	0D	DEC IX	DD 2B
DEC D	15	DEC IY	FD 2B
DEC (HL)	35		
DEC (IX+d)	DD 35 d		
DEC (IY+d)	FD 35 d		

Подгруппа c. Команды SBC.

Мнемоника	16-ричный код	Мнемоника	16-ричный код
SBC A,+dd	DE dd	SBC HL,HL	ED 62
SBC A,A	9F	SBC HL,BC	ED 42
SBC A,H	9C	SBC HL,DE	ED 52
SBC A,L	9D	SBC HL,SP	ED 72
SBC A,B	98		
SBC A,C	99		
SBC A,D	9A		
SBC A,E	9B		
SBC A,(HL)	9E		
SBC A,(IX+d)	DD 9E d		
SBC A,(IY+d)	FD 9E d		

Команда SBC выполняет "истинное" вычитание, если флаг переноса сброшен и вычитание с заемом, если флаг переноса установлен. Это может быть очень полезно, когда используются значения повышенной точности, так как заем будет участвовать в операциях. Например, 4-байтовое число, содержащееся в регистрах H', L', H, L может быть вычтено из другого числа, хранящегося в регистрах D', E', D, E следующим образом.

```
AND A       сброс флага переноса
SBC HL,DE   вычитание младшей части
EXX         переключение набора
SBC HL,DE   вычитание старшей части
EXX         переключение набора
```

и соответствующий заем между младшей и старшей частями учитывается.

Группа 8. Команды сравнения.

Команды этой группы используются очень часто во всех программах. Они позволяют программисту сравнивать значение, находящееся в регистре A с константой, значением в регистре или области памяти.

Команда сравнения выполняет операцию вычитания без переноса, без запоминания результата вычитания и только устанавливаются флаги в регистре F. Первоначальное значение в регистре A не изменяется.

Флаг переноса устанавливается так же, как и при операции вычитания. Сравнение, которое дает равенство, сбрасывает флаг переноса, а неравенство - устанавливает.

Команды этой группы - команды "одинарного сравнения", а "блочное сравнение" рассматривается на странице 86.

Команды этой группы:

Мнемоника	Шестнадцатиричный код
CP +dd	EE dd
CP A	BF
CP H	BC
CP L	BD
CP B	B8
CP C	B9
CP D	BA
CP E	BB
CP (HL)	BE
CP (IX+d)	DD BE d
CP (IY+d)	FE BE d

Следующие примеры показывают использование команды CP B.

1. Регистр A = 31h CP B оставляет регистры без изменения
   B = 30h и сбрасывает флаг переноса (31 > 30).

   Регистр A = 30h CP B оставляет регистры без изменения
   B = 30h и сбрасывает флаг переноса (30 = 30).

   Регистр A = 01h CP B оставляет регистры без изменения
   B = 30h и устанавливает флаг переноса (1 > 30).

Группа 9. Логические команды.

В системе команд Z80 существуют команды AND, OR и XOR, которые применяются к регистру A и другому заданному операнду. Операции выполняются побитно и 8-битовый результат возвращается в регистр A.

Три типа логических команд будут теперь обсуждены по очереди.

Подгруппа a. Команды AND

Логическая операция "И" выполняется над двумя двоичными цифрами и результат равен 1 только, если оба тестируемых бита установлены. В противном случае результирующий бит нулевой.

Следующий пример показывает, как команда "И" выполняется над восемью отдельными битами.

```
    двоичн. 10101010          шестнадц. AA
              AND                     AND
            11000000                  C0
  результат ----------------------------------
            10000000                  80
```

Команды в подгруппе следующие:

Мнемоника	Шестнадцатиричный код
AND +dd	E6 dd
AND A	A7
AND H	A4
AND L	A5
AND B	A0
AND C	A1
AND D	A2
AND E	A3
AND (HL)	A6
AND (IX+d)	DD A6 d
AND (IY+d)	FD A6 d

Командой AND возможно обрабатывать биты 0-3 регистра A. Этот процесс называется демаркированием и позволяет программисту проверять определенные биты данных. Следующий пример показывает, как это делается.

В системе Спектрум биты 0-2 системной переменной ATTR-P содержат информацию о цвете чернил. Когда цвет чернил изменяется, старое значение удаляется использованием команды AND и новый цвет вводится по команде ADD, т.е.

```
LD A,(ATTR-P)          находится системная переменная
AND +F1                демаркируется старший цвет
AND +NEW COLOUR        добавляется новый цвет
LD (ATTR-P),A          сохраняется системная переменная
```

Подгруппа b. Команды OR

Логическая операция "ИЛИ" выполняется над двумя двоичными цифрами и результат равен 1, если одна или обе двоичные цифры установлены. В противном случае бит результата сброшен.

Следующий пример показывает, как команда OR выполняет операцию над восемью отдельными битами.

```
   двоичн. 10101010          шестнадц. AA
              OR                        OR
           11000000                     C0
результат  ------------------------------------
           11101010                     EA
```

Команды в подгруппе следующие:

Мнемоника	Шестнадцатиричный код
OR +dd	F6 dd
OR A	B7
OR H	B4
OR L	B5
OR B	B0
OR C	B1
OR D	B2
OR E	B3
OR (HL)	B6
OR (IX+d)	DD B6 d
OR (IY+d)	FD B6 d

Подгруппа c. Команды XOR

Логическая операция "XOR" ("ИСКЛЮЧАЮЩЕЕ ИЛИ") выполняется над двумя двоичными цифрами и результат равен 1, только если одна двоичная цифра установлена. В противном случае бит результата сброшен.

Следующий пример показывает, как команда XOR выполняет операцию над восемью отдельными битами.

```
    двоичное 10101010      шестн. AA
               XOR              XOR
             11000000           C0
результат: --------------------------
             01101010           6A
```

Команды этой подгруппы:

```
      XOR +dd      EE dd
      XOR A        AF
      XOR H        AC
      XOR L        AD
      XOR B        A8
      XOR C        A9
      XOR D        AA
      XOR E        AB
      XOR (HL)     AE
      XOR (IX+d)   DD AE d
      XOR (IY+d)   FD AE d
```

При работе команды XOR будут устанавливаться или сбрасываться до 8 битов регистра A. Это, вероятно, вначале трудно понять, но рассмотрите пример, приведенный выше еще раз; а именно:

```
AA XOR C0      дает 6A
```

В этом примере вторым операндом является байт "C0", который является байтом только из битов 6 и 7.

Поэтому эффект работы операции "XOR" заключается в переключении битов 6 и 7 первого операнда и изменении AA на 6A.

Использование команд XOR в программах на машинном коде часто является сложным, но команда XOR A, однако, часто используется как альтернатива для LD A,+00. Обе из этих команд "очищают" регистр A, но XOR A использует только один адрес, в то время, как LD A,+00 - использует два адреса.

Все команды AND, OR и XOR сбрасывают флаг переноса.

Группа 10. Команды перехода

В Z80 имеются 17 команд, которые позволяют программисту выполнять "переходы" внутри программы. Переход машинного кода может приравниваться к команде Бейсика "GO TO", но сравнение не должно заходить слишком далеко.

Команды этой группы лучше всего рассматривать в 8 подгруппах.

Четыре из них содержат команды, которые условно зависят от состояния одного из основных флагов и, если требуется,

флаги будут подробно рассмотрены.

Подгруппа a: команда безусловного перехода

JP addr C3 addr

Это классическая команда перехода. При выполнении команды значение addr загружается в счетчик команд и выполнение программы в машинных кодах продолжается с этого адреса.

Подгруппа b: команды перехода, использующие косвенную адресацию

Командами этой группы являются:

Мнемоника	Шестнадцатиричный код
JP (HL)	E9
JP (IX)	DD E9
JP (IY)	FD E9

Подгруппа c: команда относительного безусловного перехода

Это команда:

Мнемоника	Шестнадцатиричный код
JR e	I8 e

Подгруппа d: команды перехода по флагу переноса

Кратко можно описать следующее правило:

1. Все команды ADD и ADC влияют на флаг переноса. Если нет "переполнения", флаг сбрасывается, если есть - флаг устанавливается.

2. Все команды SUB,SBC и CP влияют на флаг переноса. Если был двоичный заем, тогда флаг устанавливается; в противном случае - сбрасывается.

3. Все команды AND, OR и XOR сбрасывают флаг переноса.

4. Команды сдвига влияют на флаг переноса (см. стр.).

Командами перехода по флагу переноса являются:

Мнемоника	Шестнадцатиричный код	
JR NC, addr	D2 addr	переход, если флаг С сброшен
JR NC, e	30 e	переход, если флаг С сброшен
JR NC, addr	DA addr	переход, если флаг С установлен
JR C, e	38 e	переход, если флаг С установлен

В качестве примера этих команд рассмотрите следующую программу, которая "распознает" цифры в коде ASCII.

```
                          введите регистр A
                          хранящий код
    CP +30                код цифры "0"
    JR C, error           переход, если он находится
                          за пределами диапазона
    CP +3A                код для ":"
    JR NC, error          переход, если он за пределами
                          диапазона
```

Подгруппа e: команды перехода по флагу нуля

Имеются четыре команды, которые позволяют выполнять "переход" только в том случае, если флаг нуля будет таким же, как требует команда.

Флаг нуля.

Этому флагу соответствует шестой бит регистра F и в большинстве случаев он устанавливается, если результат работы равен нулю, в противном случае он сбрасывается.

Например:

```
  6C ADD 5A     дает C6 и флаг нуля сбрасывается; но
  6C ADD 94     дает 00 и флаг нуля устанавливается.
```

Можно привести кратко следующие правила:

1. Все команды ADD, INC, ADC, SUB, DEC, SBC, CP, AND, OR и XOR, использующие одиночные регистры, и команды ADC и SBC, использующие пару регистров, установят флаг нуля, если результат равен нулю.

2. Команды сдвига, команды проверки битов и команды поиска блока влияют на флаг нуля.

3. Команды LD, за исключением LD A,I и LD A,R не влияют на флаг нуля.

Командами перехода по состоянию флага нуля являются:

Мнемоника	Шестнадцатиричный код	
JR NZ, addr	C2 addr	переход, если флаг
JR NZ, e	20 e	нуля сброшен
JR Z, addr	CA	переход, если флаг
JR Z, e	28 e	нуля установлен

Команды этой подгруппы широко используются, и следующий пример показывает, как распознать символы ASCII.

```
                        введите в регистр A код символа.
CP +3B                  Это символ ":"?
JR Z, S-colon           переход, если это не так.
CP +2C                  Это ","?
JR NZ, else             переход, если нет и
                        продолжение при ","
```

Подгруппа f: команды перехода по флагу знака

Имеются две команды, которые позволяют выполнить переход только в том случае, если флаг знака будет соответствовать условию команды.

Далее мы обсудим флаг знака.

Этому флагу соответствует седьмой бит регистра F и в большинстве случаев он представляет собой копии старшего (левого) бита результата.

Когда восьмибитное или шестнадцатибитное двоичное число представлено в виде арифметического дополнения до 2, тогда левый бит (бит 7 или 15) принимается как бит знака. Он сбрасывается для положительных чисел, и устанавливается для отрицательных.

Поэтому флаг знака может сбрасываться с помощью положительных результатов и устанавливается с помощью отрицательных.

Кратко опишем следующее правило:

1. Все команды ADD, INC, ADC, SUB, DEC, SBC, CP, AND, OR и XOR, использующие один регистр, а команды ADC и SBC, использующие два регистра, будут влиять на флаг знака, как указано выше.

2. Команды поиска блока и большинство команд сдвига также влияют на флаг знака.

3. Команды LD, за исключением LD A,I и LD A,R не изменяют флаг знака.

Командами перехода по состоянию флага знака являются:

```
JR P, addr  F2 addr  переход, если результат положительный
JR M, addr  FA addr  переход, если результат отрицательный
```

Команды в этой подгруппе обычно не используются. Частично потому, что требуют абсолютного адреса, с частично потому, что бит знака может считываться несколькими другими способами.

Подгруппа g: команды перехода по флагу переполнения/четности

Имеются две команды, позволяющие выполнить переход, если флаг переполнения / четности будет таким, как того требует команда.

Флаг переполнения / четности

Этому флагу соответствует второй бит регистра F и он является флагом двойного назначения. Некоторые команды используют флаг, чтобы указать на "переполнение", в то время, как другие команды используют его, чтобы хранить результат проверки на четность.

Понятие "переполнение" не относится к двоичному переполнению, а к арифметическому дополнению до 2, как показано следующими примерами.

Рассмотрим: 16-ричное 0A ADD 5C дает 66,
десятичное 10 ADD 92 дает 10, что правильно - нет переполнения.
16-ричное 6A ADD 32 дает 9C,
десятичное 106 ADD 50 дает 100, что неправильно - имеется переполнение.

Переполнение может также возникнуть при вычитании, а именно:

16-ричное 83 SUB 14 дает 6F,
десятичное 125 SUB 20 дает 111, что неправильно, т.е. имеется переполнение.

Флаг переполнение / четность устанавливается, когда имеет место переполнение.

Понятие "четности" касается числа единичных битов в заданном байте.

Это показано следующим примером.

Байт 01010101 имеет четное равенство и флаг устанавливается, но байт 00000001 имеет нечетное равенство и флаг сбрасывается.

Кратко опишем следующее правило:

1. Все команды ADD, ADC, SBC и CP, использующие одиночные регистры, а команды ADC и SBC, использующие два регистра, дают результат, который может быть проверен на переполнение.

2. Все команды AND, OR и XOR и большинство команд сдвига дают результат, который может быть проверен на четность.

3. Команда INC установит флаг, если результат будет равен 80h и команда DEC, результат равен 7Fh.

4. Различные другие команды, а именно: LD A,I и LD A,R и многие из команд обработки блоков также влияют на состояние флага переполнения / четности.

Команды перехода по флагу переполнения / четности представлены ниже.

```
JP PO, addr     E2 addr   переход, если равенство
                          нечетное или  нет переполнения
JP PE, addr     EA adda   переход, если равенство
                          четное или есть переполнение
```

Команды этой подгруппы используются только в тех случаях, когда невозможно использование других команд.

Группа 11. Команды DJNZ e

Одиночная команда в этой группе является одной из самых полезных и самой широко используемой в ZX80.

Это команда:

мнемоника	16-ричный код
DJNZ e	10 e

Мнемоника означает: уменьшение регистра B и переход, если флаг нуля сброшен.

Работа этой команды может быть похожа на цикл FOR-NEXT программы на Бейсике следующего вида:

```
FOR B=x TO 0 STEP -1: NEXT B
```

В этом цикле в переменную B заносится величина x. Затем, с каждым проходом цикла B уменьшается до тех пор, пока не достигнет нуля.

Команда DJNZ, e используется подобным образом. Прежде всего, программист должен определить значение переменной цикла и занести ее в регистр B. Затем идет цикл. Наконец, используется, что величина смещения подходит.

Следующий пример показывает, как эта команда должна использоваться для печати алфавита.

```
LD B,  +1A      26 букв в алфавите (английском).
LD A,  +5B      "A" - первая, т.е.
SUB B           3B - 1A = 41 (шестнадцатир.)
RST 0010        печать символа
DJNZ, L00P      "сдвиньте" для "B", "C" и т.д. аналогично
```

Шестнадцатиричным кодом для этого примера является
06, 1A, 3E, 5B, 90, D7, 10, FA
где байт "3E" является "меткой" цикла.

Автор обнаружил, что лучшие методики для вычисления правильной величины "e" в примерах коротких машинных кодов, какой приведен - это представить байт "e" величиной "FF" и затем считать в обратном направлении в 16-ричной форме до тех пор, пока не будет достигнут байт цикла. В приведенном выше

примере имеется пять шагов назад и соответствующая величина для "e" равна 16-ричному FA.

Группа 12. Команды стека

В большинстве программ в машинных кодах осуществляется активное использование машинного стека как программистом - в качестве места, где можно сохранить данные, так и микропроцессором - для сохранения адресов возврата.

Команды, которые образуют эту группу, можно разделить на две подгруппы для пользователя и три подгруппы для микропроцессора.

Подгруппа a: команды PUSH и POP.

Эти команды позволяют программисту по PUSH сохранять два байта данных в машинном стеке, а по PP - копировать два байта из машинного стека.

Два байта данных должны копироваться в строго установленную пару регистров, но важно знать, что никакой записи не производится, которая указывала бы, к какому из двух регистров какие байты должны принадлежать.

Командами этой подгруппы являются:

Мнемоника	16-ричный код	Мнемоника	16-ричный код
PUSH AP	F5	POP AF	F1
PUSH HL	E5	POP HL	E1
PUSH BC	C5	POP BC	C1
PUSH DE	D5	POP DE	D1
PUSH IX	DD E5	POP IX	DD E1
PUSH IY	FD E5	POP IY	FD E1

Подгруппа b: команды обмена

Эти команды позволяют обменять содержимое регистра и двух байт с вершины стека.

Командами этой группы являются:

Мнемоника	16-ричный код
EX (SP),HL	E3
EX (SP),IX	DD E3
EX (SP),IY	FD E3

Подгруппа c: команды CALL

Эти команды позволяют программисту обращаться к подпрограммам.

Командами этой подгруппы являются:

Мнемоника	16-ричный код
CALL adar	CD adar
CALL NZ,adar	C4 adar
CALL Z,adar	CC adar

```
CALL NC,addr      D4 addr
CALL C,addr       DC addr
CALL PO,addr      E4 addr
CALL PE,addr      EC addr
CALL P,addr       F4 addr
CALL M,addr       FC addr
```

Подгруппа d: команды RET

Командами этой подгруппы являются:

Мнемоника	16-ричный код
RETN	CB 45
RETI	CB 4D
RET	C9
RET C	D8
RET NC	D0
RET Z	C8
RET NZ	C0
RET M	F8
RET P	F0
RET PE	E8
RET PO	E0

По команде RET адрес возврата из машинного стека побайтно восстанавливается в счетчике команд. При этом указатель стека дважды модифицируется в сторону увеличения.

Важно понимать, что восстановленный адрес возврата из машинного цикла не обязательно тот, что был вначале помещен туда командой CALL.

Не совсем обычно при программировании на Бейсике модифицировать стек GO SUB, но при программировании в машинных кодах это имеет место.

Следующий пример показывает, как это делается. Адрес программы выбирается из таблицы адресов и на эту программу осуществляется переход.

```
.....              регистр A содержит входное значение
LD D +00           очистка регистра
ADD A              дублирование входного значения
LD E, A            и пересылка в регистр
LD HL, +Table-base HL указывает на таблицу
ADD HL, DE         указательна точку входа
LD D (HL)          получение старшего байта адреса
INC HL
LD E (HL)          получение младшего байта адреса
PUSH DE            сохранение адреса в стеке
RET                возврат
```

Подгруппа e: команды RST

Последняя подгруппа команд в этой группе содержит специальную команду RST или команду "повторный запуск".

Эти команды являются аналогичными командам CALL, но не требуют, чтобы "адрес" определялся.

Командами этой подгруппы являются:

RST 0000	C7	CALL 0000
RST 0000	CF	CALL 0008
RST 0010	D7	CALL 0010
RST 0018	DF	CALL 0018
RST 0020	E7	CALL 0020
RST 0028	EF	CALL 0028
RST 0030	F7	CALL 0030
RST 0038	FF	CALL 0038

В системе Спектрум восемь адресов "перезапуска" являются точками входа подпрограмм монитора и они будут рассмотрены в главе 7.

Группа 13. Команды сдвига

Набор команд Z80 имеет большее число команд для сдвига битов определенного байта. Эти команды часто очень полезны. Тем более, что все они сдвигают биты через флаг переноса, который затем может проверяться.

Сдвиг байта влево удваивает величину, записанную в этом байте, поскольку самые старшие биты не теряются. Во время сдвига байта вправо величина делится пополам.

Рисунок 5.1 показывает ряд сдвигов, которые возможны.

Следующая таблица демонстрирует команды этой группы

	RLC	RL	SLA	RRC	RR	SRA	SRL
A	CB07	CB17	CB27	CB0F	CB1F	CB2F	CB3F
H	CB04	CB14	CB24	CB0C	CB1C	CB2C	CB3C
L	CB05	CB15	CB25	CB0B	CB1D	CB2D	CB3D
B	CB00	CB10	CB20	CB08	CB18	CB28	CB38
C	CB01	CB11	CB21	CB09	CB19	CB29	CB39
D	CB02	CB12	CB22	CB0A	CB1A	CB2A	CB3A
E	CB03	CB19	CB23	CB0B	CB1B	CB2B	CB3B
(HL)	CB06	CB16	CB26	CB0E	CB1E	CB2E	CB3E
(IX+d)	DDCB	DDCB	DDCB	DDCB	DDCB	DDCB	DDCB
	d 06	d 16	d 26	d 0E	d 1E	d 2E	d 3E
(IY+d)	FDCB	FDCB	FDCB	FDCB	FDCB	FDCB	FDCB
	d 06	d 16	d 26	d 0E	d 1E	d 2E	d 3E

Имеется также четыре однобайтовые команды для сдвига регистра A и две специальные команды, работающие с полубайтами.

```
RLCA      07
RLA       17
RRCA      0F
RRA       1F
RRD       ED 67
RLD       ED 6F
```

Суммируя влияние команд из этой группы на основные флаги, можно сделать следующие заклинания:

1. Все команды, за исключением RLD и RRD, влияют на флаг переноса
2. Все команды, за исключением четырех команд с одиночными байтами влияют на флаги знака и переполнения / четности.

RLC & RLCA

RL & RLA

SLA

RRC & RRCA

RR & RRA

SRA

SRL

RLD (HL)

RRD

Две специальные команды

Рис. 5.1. Различные типы сдвигов

Группа 14. Команды обработки битов

Набор команд Z80 имеет команды, которые позволяют программисту проверять, устанавливать и сбрасывать определенные биты в пределах байта, помещенного в регистр или находящегося в памяти.

Три типа команд будут рассмотрены поочередно.

Подгруппа a: команды BIT

Команды BIT позволяют программисту определять состояние определенного бита. За использованием одной из этих команд обычным является процесс проверки, например, путем использования команды JP z.

Команда BIT устанавливает флаг нуля, если исследуемый бит сброшен, и наоборот.

Подгруппа b: команды SET

Команды SET позволяют программисту устанавливать определенный бит. Ни один из флагов не изменяется.

На практике команды используются для установки битов. Но во многих случаях их использование лучше описывать как подтверждение, что определенный бит установлен. Команда SET, работающая на бите, который уже установлен, не будет иметь демонстрационного эффекта.

Подгруппа c: команды RES

Команды RES позволяют программисту сбрасывать определенный бит. Ни один из флагов не изменяется.

Команды из этих трех подгрупп приведены в следующей таблице.

	BIT	47	4F	57	5F	67	6F	77	7F
	RES	87	BF	97	9F	A7	AF	B7	BF
B**	SET	C7	CF	D7	DF	E7	EF	E7	FF
	BIT	44	4C	54	5C	64	6C	74	7C
	RES	84	8C	94	9C	A4	AC	B4	BC
B**	SET	C4	CC	D4	DC	E4	EC	E4	FC
	BIT	45	4D	55	5D	65	6D	55	5D
	RES	85	8D	95	9D	A5	AD	B5	BD
B**	SET	C5	CD	D5	DD	E5	ED	F5	FD
	BIT	40	48	50	58	60	68	00	08

	RES	80	88	90	98	A0	A8	B0	B8
CB**	SET	C0	C8	D0	D8	E0	E8	F0	F8
C	BIT	41	49	51	59	61	69	11	19
	RES	81	89	91	99	A1	A9	B1	B9
CB**	SET	C1	C9	D1	D9	E1	E9	F1	F9
D	BIT	42	4A	52	5A	62	6A	22	2A
	RES	82	8A	92	9A	A2	AA	B2	BA
CB**	SET	C2	CA	D2	DA	E2	EA	F2	FA
F	BIT	43	4B	53	5B	63	6B	33	3B
	RES	83	8B	93	9B	A3	AB	B3	BB
CB**	SET	C3	CB	D3	DB	E3	EB	F3	FB
(HJ)	BIT	46	4E	56	5E	66	6E	66	6E
	RES	86	8E	96	9E	A6	AE	86	BE
CB**	SET	C6	CE	D6	DE	E6	EE	F6	FE

Замечание: для команд, использующих индексные регистры, см. приложение.

Группа 15. Команды обработки блока

Имеется восемь команд блока обработки в Z80. Эти команды очень интересны и полезны. Они позволяют программисту перемещать блок данных из одной области памяти в другую, или искать область памяти.

Чтобы перемещать блок данных, базовый адрес должен помещаться в пару регистров HL, и число байтов в блоке - в пару регистров BC.

Для того, чтобы искать область первого присутствия в памяти определенной величины, базовый адрес может опять быть в паре регистров HL, число байтов области поиска - в паре регистров BC, а регистр A должен содержать искомую величину.

Команды подгруппы:

AUTOMATIC	
LDIR	ED B0
LDDR	ED B8
CPIR	ED B1
CPDR	ED B9
NON - AUTOMATIC	
LDI	ED A0
LDD	ED A8
CPI	ED A1

CPD ED A9

Как можно увидеть из вышеприведенной таблицы, имеется как автоматические, так и неавтоматические команды. Автоматические команды более простые и более полезные.

Автоматическая команда будет выполнять свою задачу без другой команды, составленной программистом. Когда команда LDIR выполняется микропроцессором, тогда блок данных перемещается. Отсюда следует, что выполнение автоматической команды может длиться разное время. Это время зависит от количества байтов, которые будут перемещаться.

Однако, неавтоматическая команда передает только по одному байту и требует, чтобы программист использовал команду поочередно, если блок данных перемещается или находится в стадии поиска. Время неавтоматической команды поэтому определенное.

Неавтоматическая команда представляет собой команду "обработки блока", при выполнении которой адреса источника, получателя и счетчика приращения уменьшаются или увеличиваются микропроцессором.

Каждая команда этой группы будет обсуждаться по очереди.

LDIR

Это самая распространенная команда из 8 команд этой группы.

Команда LDIR будет перемещать данные, адрес источника которых содержится в паре регистров H, в область памяти с адреса, находящегося в паре регистров DE с числом байтов, определенных в паре регистров BC.

При работе одиночный байт перемещается из (HL) в (DE).

Величина в паре регистров BC затем уменьшается, а величины в парах HL и DE увеличиваются.

Если "счетчик" в регистре BC еще не достиг нуля, будет пересылаться следующий байт данных. Этот процесс продолжается пока "счетчик" не достигнет нуля, но заметьте, что в этот момент регистры HL и DE достигают значения адресов после блока.

Флаг переполнения / четности сбрасывается этой командой.

LDDR

Эта команда является такой же, как команда LDIR, за исключением того, что после пересылки каждого байта, значения в парах регистров HL и DE уменьшаются. Поэтому команда требует, чтобы "базовый адрес" блока соответствовал последнему адресу блока. Место назначения информации также должно относиться к последнему адресу области.

CPIR

Эта команда просматривает определенную область памяти для поиска первого появления образца. Регистр HL должен содержать базовый адрес, а BC - число байтов для исследования, регистр A - образец.

При работе байт из (HL) сравнивается с байтом (A). Если они не совпали, BC уменьшается, а HL увеличивается и сравниваются следующие. Так до тех пор, пока не совпадут байты, либо пока BC не обнулится. Если байты совпали, флаг нуля устанавливается, флаг знака сбрасывается, а регистр HL указывает адрес выбранного байта. Величина в BC указывает, как далеко от конца блока определено совпадение.

Если байт не был найден, регистр BC=0, флаги нуля, знака и переполнения / четности сбрасываются.

CPDR

Работа этой команды похожа на CPIR, но блок данных просматривается снизу вверх.

Далее рассматриваются неавтоматический команды.

LDI

Выполнение этой команды перемещает одиночный байт данных их адреса в регистровой паре HL по адресу, указанному в регистровой паре DE. Величина в BC уменьшается. Флаг переполнения / четности устанавливается до тех пор, пока регистровая пара BC не станет равна 0. Значения в регистровых парах DT и HL уменьшаются.

Это сделано для программиста, который решил либо перемещать, либо не перемещать байты данных. Программист, вероятно, проверит байт, который надо перемещать и затем уже осуществит его перемещение.

LDD

Как LDI, за исключением того, что величины в HL и DE уменьшаются.

CPI

Выполнение этой команды приведет к тому, что байт, определенный по паре HL, копируется в микропроцессор и сохраняется там до тех пор, пока величина в HL увеличивается, а в BC - уменьшается. Сохраненный байт затем сравнивается с байтом в регистре A.

Если они совпали, флаг нуля устанавливается, иначе - сбрасывается.

Флаги знака и переполнения / четности сбрасываются до тех пор, пока BC не станет равным нулю, при BC=0 он останавливается.

CPD

Эта команда подобна CPI за исключением того, что регистровая пара HL уменьшается.

Группа 16. Команды ввода/вывода

В Z80 имеется исчерпывающий набор команд, позволяющий программисту получать данные от внешнего устройства (IN) или выдавать данные на периферию (OUT).

Имеются простые, неавтоматические и автоматические команды в этой группе, хотя в Спектрум используется только простые команды. Во всех случаях данные, передающиеся через IN или OUT представляют форму 8-битовых байтов. По команде IN Z80 принимает байты данных с шины данных и копирует их в определенный регистр. Во время выполнения команды IN контрольная линия IORQ активна, также как и RD.

По команде OUT, МП передает данные с определенного регистра на шину данных, откуда они поступают на периферийное устройство. Во время работы команда OUT линии IORQ и WR будут активна.

В дополнение в состоянии RD, WR и IORQ периферийное устройство будет активизироваться с использование адреса, помещенного на адресной шине во время выполнения либо IN, либо OUT команды (по требованию). Этот адрес опознает адрес порта и в Z80 представляет собой 16-разрядный адрес. В небольшом микрокомпьютере, таком, как Спектрум, обычно используют всего несколько из 65536 возможных адресов портов, позволяя периферийному устройству идентифицироваться состоянием определенной линии адреса.

Принтер Спектрум выбирается линией A2.

Командами этой группы являются:

IN A, (+dd)	DB	dd	A	A	dd
IN A, (C)	ED	78	A	B	C
IN H, (C)	ED	60	H	B	C
IN L, (C)	ED	68	L	B	C
IN B, (C)	ED	40	R	B	C
IN C, (C)	ED	48	C	B	C
IN D, (C)	ED	50	B	B	C
IN E, (C)	ED	58	E	B	C
OUT (+dd).A	D3	dd	A	B	dd

```
OUT   (C),A    ED    79    A    B    C
OUT   (C),H    ED    61    H    B    C
OUT   (C),L    ED    69    L    B    C
OUT   (C),B    ED    41    B    B    C
OUT   (C),C    ED    49    C    B    C
OUT   (C),D    ED    51    D    B    C
OUT   (C),E    ED    59    E    B    C
```

Неавтоматическими и автоматическими командами являются:

```
INI     ED  A2   увеличение; неавтоматическая
INIR    ED  B2   увеличение; автоматическая
IND     ED  AA   уменьшение; неавтоматическая
INDR    ED  BA   уменьшение; автоматическая
OUTI    ED  A3   увеличение; неавтоматическая
OTIR    ED  B3   увеличение; автоматическая
OUTD    ED  AB   уменьшение; неавтоматическая
OTDR    ED  BB   уменьшение; автоматическая
```

Группа 17. Команды прерывания

Имеется 7 команд в Z80, которые позволяют программисту прервать систему Z80.

Командами этой группы являются:

```
EI     FB
DI     F3
IM0    ED 46
IM1    ED 56
IM2    ED 5E
RETI   ED 4D
RETN   ED 45
```

Каждая из этих команд будет рассмотрена поочередно.

EI

При включении Z80, система маскируемых прерываний не может прерывать Z80. Эта ситуация существует до тех пор, пока прерывания не будут разрешены по команде EI.

В Спектрум система маскируемых и немаскируемых прерываний используется для отсчета реального времени и опроса клавиатуры и эти прерывания синхронизируются частотой 50 гц.

DI

В любом месте программы программист может запретить пре-

рывания по команде DI, которая не позволяет микропроцессору принимать сигналы с линии INT.

В Спектрум маскируемые прерывания невозможны во время выполнения команд SAVE, VERITY, MERGE.

IM 0

Имеется три режима прерываний режим 0 выбирается автоматически микропроцессором при выключении или по команде IM 0. Этот режим позволяет периферийным устройствам обмениваться информацией с микропроцессором, программа перезапуска которого должна следить за приемом маскируемого прерывания на линии INT. Этот режим в Спектрум не используется.

IM 1

Режим прерывания 1 выполняется по команде IM1, и он используется в Спектрум. При использовании 16К монитора необходимо включить IM1 в начало программы.

В этом режиме перезапуск с адреса 0038h будет всегда выбираться при приеме сигнала по линии INT, длительность которого соответствует возможности обработки системы маскируемых прерываний. В Спектрум программа в машинных кодах с 0038 корректирует данные, полученные в реальном времени и опрашивает клавиатуру.

IM 2

Режим прерывания 2 не используется в Спектрум, но является самым мощным из трех режимов. В нем периферийное устройство может указывать микропроцессору, какая из 128 различных программ должна обрабатывать маскируемое прерывание. Содержимое регистра I и байта, который поступает на внешнее устройство, используются вместе, чтобы образовать 16-разрядный адрес, который затем используется для выбора "таблицы векторов", которая прежде размещается в памяти.

RETI

Это специальная команда возврата для использования с программой обработки маскируемых прерываний. По этой команде происходит возврат с сохранением маскируемого прерывания, принятого ранее.

RETN

Эта команда подобна RETI, но она применима в конце программы маскируемых прерываний.

Группа 18. Дополнительные команды

Имеется 6 команд, которые рассматриваются далее, а именно:

CPL	2F		CCF	3F
NEG	ED	14	HALT	76
SCF	37		DAA	27

Каждая из них будет рассмотрена по очереди.

CPL

Представляет собой простую команду ,которая инвертирует регистр A ,т.е. устанавливает сброшенный бит и сбрасывает установленный. Это называется дополнением до единицы. Основные флаги не изменяются.

NEG

Это команда дополнения до 2 содержимого регистра A, т.е. инвертирование с увеличением. Команда NEG влияет на основные флаги. Флаги знака и нуля зависят от результатов. Флаг переноса будет установлен ,если регистр A первоначально был нулевой, в противном случае он сбрасывается и флаг переполнения/четности устанавливается, если в начале в регистре A было 16-ричное 80, в противном случае сбрасывается.

SCF

Установка флага переноса.

CCF

Сброс флага переноса.

HALT

Это специальная команда, заставляющая микропроцессор останавливать выполнение команд, пока не возникает прерывание. В Спектрум единственными прерываниями могут быть маскируемые. Следовательно, при поступлении прерываний, команда HALT заканчивается следующим сигналом маскируемого прерывания. Команда PAUSE использует это, чтобы отсчитывать по 1/50 секунды.

DAA

Это команда десятичной переупаковки регистра A. В двоично-десятичной арифметике (BCD)-десятичные цифры представлены двоичными тетрадами 0000-1001, а тетрады 1010-1111 не используются. Поэтому: байт 0000 0000 представляет число 0 байт 0011 1001 представляет число 39.

Эта команда превращает байты в двоично-десятичную систему (BCD). Флаги нуля и знака зависят от результата, а флаг переполнения/четности устанавливается. Флаг переноса зависит от того, было ли переполнение или заем при операции BCD.

ГЛАВА 6. ДЕМОНСТРАЦИОННЫЕ ПРОГРАММЫ НА МАШИННОМ ЯЗЫКЕ.

6.1 Введение.

Цель данной главы - ознакомить читателя с некоторыми проблемами, возникающими на начальном этапе программирования на машинном языке.

Хотя в главе 5 перечислены все команды системы команд микропроцессора Z80, но при этом не указывалось, как эти команды можно использовать в конкретных программах, написанных на машинном языке микро-ЭВМ "СПЕКТР". В данной главе приведен набор демонстрационных программ , показывающих достаточно простое использование различных команд.

В главе 8 будут приведены дополнительные рекомендации по программированию на машинном языке, которые позволяют использовать преимущества цветного дисплея с высоким разрешением микро-ЭВМ "СПЕКТР".

Вначале рассмотрим следующие три вопроса.

1. Выбор области ЗУПВ.

Пользовательская программа, написанная на машинном языке

должна распределять нужное количество ячеек памяти. В микро-ЭВМ "СПЕКТР" имеется несколько доступных для использования областей памяти, но в демонстрационных программах данной главы используется область "свободной памяти" с произвольной выборкой, которая начинается в ячейке с десятичным адресом 32000. При необходимости можно сохранить (команда SAVE) или загрузить (команда LOAD) программу, написанную на машинном языке из этой области на магнитную ленту или с магнитной ленты в эту область данных.

2. Ввод байтов машинного кода.

Единственный способ ввода машинного кода в микро-ЭВМ "СПЕКТР" - использовать команду POKE. Операндами этой команды могут быть десятичные числа, двоичные числа или выражения.

Поэтому разрешена любая из следующих строк:

```
10 POKE 32000,201
```

или:

```
10 POKE 32000,BIN 11001001
```

или:

```
10 LET A=201:POKE 32000,A
```

полезность каждого способа зависит от конкретного условия.

Рекомендуется описывать команды машинного языка с помощью шестнадцатиричных кодов. Везде в данной главе используется приведенная ниже программа шестнадцатиричного загрузчика:

```
10 LET D=32000: REM Hex Loader (16-ричный загрузчик)
20 DEF FN A(A$,B)=CODE A$(B)-48-7*(CODE A$(B) > 57)
30 DEF FN C(A$)=16*FN A(A$,1)+FN A(A$,2)
40 READ A$: IF A$ <> " " THEN POKE D,FN C(A$):
   LET D=D+1: GO TO 40
```

Этот шестнадцатиричный загрузчик обеспечивает считывание списка DATA, в котором каждая пара шестнадцатиричных символов взята в кавычки и отделена запятыми от соседних пар.В качестве указателя конца используется пара пробелов " ".

Например, выполнение

```
50 DATA "00","01","02","  "
RUN
```

приведет к тому, что в ячейку 32000 будет помещен 0, в ячейку 32001 - 1, и в ячейку 32002 - 2.

3. Выполнение пользовательской программы, написанной на машинном языке.

С помощью команды языка Бейсик "USR номер" можно остановить выполнение управляющей программы микро-ЭВМ "СПЕКТР" и начать выполнение пользовательской программы, написанной на машинном языке. Вначале следует убедиться , что операнд команды USR адресует нужную ячейку , что пользовательская программа завершается командой RETURN (если пользователь действительно хочет вернуться к интерпретатору языка Бейсик) и что значение , возвращаемое командой USR , обработано соответствующим образом.

Обычно используются следующие формы команды:

PRINT USR n

выводит на печать десятичное содержимое двойного регистра BC;

RANDOMIZE USR n

использует генератор случайных чисел;

LET A=USR n

использует указанную переменную;

Полезность каждой из этих форм зависит от конкретных условий.

Демонстрационные программы приводятся в формате ассемблера; при обработке списка DATA используется шестнадцатиричный загрузчик , приведенный на стр. 92.

Порядок рассмотрения команд - как в главе 5.

6.2 Программы.

Группа 1. Команда NO OPERATION (НЕТ ОПЕРАЦИИ)

Это очень простая команда , и демонстрационная программа, написанная на машинном языке, сдержит всего одну или всего несколько строк NO OPERARIN с последующей командой RETURN.

На приведенной ниже распечатке ассемблера показана простейшая программа, написанная на машинном языке, в которой используется единственная команда NO OPERATION .

Адрес	Машинный код	Мнемокод	Комментарии
7D00	00	NOP	единственн.
7D01	C9	RET	команда возврат к Бейсику

Программа 1. NOP

Строки 10-40. Шестнадцатиричный загрузчик, приведенный на стр. 92.

```
50 DATA "00","09"," "
60 LET A=USR 32000
70 PRINT "The NOP mashine code program has been execude
   succesafully"
```

(выполнение написанной на машинном языке программы NOP успешно завершилось).

Советуем читателям хорошо изучить программу 1, прежде чем перейти к следующим разделам.

Группа 2. Команды для загрузки в регистры констант

По команде "USR номер" возвращается в виде 16-ти разрядного положительного числа-содержимое двоичного регистра BC, и по команде PRINT это число преобразуется к десятичному виду (из интервала 0-65535).

В приведенной ниже распечатке ассемблера с помощью команд этой группы константы загружаются в регистры B и C;рассматриваемой программе, написанной на машинном языке, соответствует приведенная дальше программа, написанная на языке Бейсик.

Адрес	Машинный код	Мнемокод	Комментарии
7D00	0600	LD B,+00	обнулить регистр B
7D02	0E00	LD C,+XX	пользователь вводит число
7D04	C9	RET	"Возврат к Бейсику"

В программе 2 используются обе команды-"LD B,+dd" и "LD C,+dd"; пользователь должен ввести нужное число как операнд последней команды.

Программа 2 LD B+dd, LD C,+dd

Строки 10-40. Шестнадцатиричный загрузчик, приведенный на стр.92.

```
 50 DATA "06","00","0E","00"
 51 DATA "09","  "
 60 INPUT "Enter a value for the C register (0-255 only)",N
      (введите число для регистра C (из интервала 0-255))
 70 CLS
 80 POKE 32003,N
 90 PRINT AT 10,0;"The C rehister now holds",
    CHR$ 32;USR 32000
                      (теперь в регистре C содержится)
100 GO TO 60
```

В следующей программе иллюстрируется использование команды "LD BC,+dddd".

Начнем с распечатки Ассемблера.

Адрес	Машинный код	Мнемокод	Комментарий
7D00	010000	LD BC,+xxxx	Пользователь вводит нужные значения
7D03	C9	RET	"Возврат к Бейсику"

В программе 3 используется команда "LD BC,+dddd"; пользователь должен ввести нужное число. Для его загрузки в соответствующую ячейку памяти по команде POKE следует выделить старшую и младшую части этого числа.

Программа 3. LD BC,+dddd

Строка 10-40. Шестнадцатиричный загрузчик, приведенный на стр. 92

```
 50 DATA "01","00","00"
 51 DATA "09","  "
 60 INPUT "Enter value far the BC register pair
    (0-65535 only)";CHR$ 32;N
 70 CLS
 80 POKE 32001,N-256* INT (N/256): POKE 32002,INT (N/256)
 90 PRINT AT 10,0 "BC register pair now holds";
    CNR$ 32;USR 32000
100 GO TO 60
```

Группа 3. Команды копирования и обмена содержимого

Действие команд копирования содержимого одного регистра в другой регистр можно показать, загрузив вначале константы

в регистры ,отличные от B и C, а затем скопировав эти константы в регистры B и C для возврата пользователю.

Ниже приводится программа, демонстрирующая применение команды "LD C,L".

Адрес	Машинный код	Мнемокод	Комментарии
7D00	06 00	LD B,+00	Обнулить регистр C
7D02	3E 00	LD L,+XX	Ввести числа
7D04	4D	LD C,L	Копировать "L в C"
7D05	C9	RET	"Возврат к Бейсику"

В программе 4 пользователь должен ввести в регистр L число, которое затем возвращается в регистр C.

Программа 4. LD C,L

Строки 10-40. Шестнадцатиричный загрузчик,приведенный на стр.92.

```
50 DATA "06","00","2E","00"
51 DATA "4D","C9","  "
60 INPUT "Enter a value for the L register
   (0-255 only)";CHR$ 32;N
70 CLS
80 POKE 32003,N
90 PRINT AT 10,0;"The C register nopw holds";
   CHR$32;USR 32000
100 GO TO 50
```

Советуем читателям попытаться использовать другие команды из этой группы. Например,допустимыми являются:

```
50    DATA    "06","00","0E","00"
51    DATA    "65","7C","57","5F"
52    DATA    "4B","C9"," "
```

Посмотрели ли Вы, что здесь делается? В программу 4 можно также включить команду "EX DE,HL"; например, это можно сделать следующим образом.

Адрес	Машинный код	Мнемокод	Комментарии
7D00	2600	LD H,100	Обнулить регистр H
7D02	2E00	LD L,+XX	Ввод числа
7D04	EB	EX DE,HL	Пересыл. числа в DE
7D05	42	LD B,D	Пересыл. сод.D в B
7D06	4B	LD C,E	и E в C
7D07	C9	RET	"Возврат к Бейсику"

Для этого можно определить следующий список.

```
50 DATA "26","00","2E","00"
51 DATA "EB","42","4B"
52 DATA "C9"," "
```

который следует использовать в программе 4.

Последние две команды этой группы -"EX AF,A'F'" и"EXX". В микро-ЭВМ "СПЕКТР" пользователь может без каких-либо ограничений использовать альтернативный набор регистров, кроме регистров H' и L', в которых сохраняется адрес возврата к Бейсику. Читатель может попытаться включить эти команды в программу 4.

Например:

```
50 DATA "06","00","2E","00"
51 DATA "08","D9","D9","08"
52 DATA "4D","C9"," "
```

или что-либо более сложное.
(В строке 51 задается переключение всех основных регистров, после чего без каких-либо изменений происходит их обратное переключение.)

Группа 4. Команды загрузки в регистры данных, скопированных из ячейки памяти

Команды этой группы позволяют пользователю загружать в регистры копии данных, содержащихся в ячейках с указанными адресами. Адресация может быть прямой, косвенной или индексной.

В первой программе раздела показан случай прямой адресации. Ячейке 31999 присваивается название STORE; для извлечения содержимого ячейки STORE используется команда "LD A,(addr)". В первой строке распечатки ассемблера название STORE присваивается ячейке с шестнадцатиричным адресом 7CFF (десятичный эквивалент 31999).

STORE	равно	7CFF	
Адрес	Машинный код	Мнемокод	Комментарии
7D00	0600	LD B,+00	Обнулить регистр B
7D02	3AFF7C	LD A,(STORE)	Извлечь текущ.знач.
7D05	4F	LD C,A	Переслать сод.A в C
7D06	C9	RET	"Возврат к Бейсику"

В программе 5 пользователь должен ввести число для загрузки в ячейку STORE . Затем программа использует приведенный выше блок для возврата этого числа пользователю.

Программа 5. LD ,A(addr)

Строки 10-40. Шестнадцатиричный загрузчик, приведенный на стр. 92.

```
50 DATA "06","00"
51 DATA "3A","FF","7C"
52 DATA "4F","C9"," "
60 INPUT "Enter a value for location STORE (0-255 only)";
   CHR$ 32;N
70  CLS
80  POKE  31999,N
90 PRINT AT 10,0,"The location STORE now hjlds";
   CHR$ 32;USR 32000
100 GO TO 60
```

Во второй программе раздела показан случай косвенной адресации. В ней до использования команды "LD 0,(HL) адрес ячейки STORE загружается в двойной регистр HL.

STORE равно 7CFF

Адрес	Машинный код	Мнемокод	Комментарии
7D00	06 00	LD B,+00	Обнулить регистр B
7D02	21 FF 7F	LD HL,+STORE	Занести в двойной регистр HL адрес ячейки STORE
7D05	4E	LD C,(HL)	Извлечь текущее значение
7D06	C9	RET	Возврат к Бейсику

В программе 6 пользователь, как и ранее, должен ввести число для занесения в ячейку STORE.

Программа 6. LD C,(HL)

Строки 10-40. Шестнадцатиричный загрузчик, приведенный на стр. 92.

```
50 DATA "06","00"
51 DATA "21","FF","7C"
52 DATA "4E","C9","  "
```

Строки 60-100. Как в программе 5.

В третьей программе раздела показан случай индексной адресации. Ячейке с десятичным адресом 31936 (шестнадцатиричный эквивалент 7CC0) присваивается имя BASE, а ячейке STORE-десятичный адрес 31999 (шестнадцатиричный эквивалент 7CFF) рассматривается как ячейка BASE+3F.Затем для получения текущего содержимого STORE используется команда "LD C,(IX+d)".

```
        BASE     равно   7CC0
        STORE    равно   BASE+3F
Адрес Машинный код    Мнемокод          Комментарии
7D00    06 00           LD B,+00          Обнулить рег.B
7D02    DD 21 C0 7C     LD IX,+BASE       Установить сод.IX
7D06    DD 4E 3F        LD C,(BASE+3F)    Извлечь тек.знач.
7D09    C9              RET               Возврат к БЕЙСИКУ
```

В программе 7 пользователь , как и ранее, должен ввести число для загрузки в ячейку STORE .

Программа 7. LD C,(IX+d)

Строки 10-40. Шестнадцатиричный загрузчик, приведенный на стр.92.

```
50 DATA "06","00"
51 DATA "DD","21","C0","7C"
52 DATA "DD","4E","3F"
53 DATA "C9","  "
```

Строки 60-100 как в программе 5.

В программе 7 используется двойной регистр IX,однако легко ввести изменения для использования двойного регистра IY. Для этого в распечатке ассемблера следует вместо IX поставить IY и вместо D0-FD.Заметим, что пока в регистре IY содержится новое значение, маскируемое прерывание невозможно.

Группа 5. Команды загрузки в ячейки памяти данных, скопированных с регистров или констант.

Команды этой группы позволяют пользователю загружать в ячейки с заданными адресами данные, скопированные с регистров или константы. Как и ранее, возможны 3 случая - прямая адресация и индексная адресация.

В первой программе раздела показан случай прямой адресации. В команде "LD (addr),A" используется прямой адрес ячейки-31999 (шестнадцатиричный эквивалент 7CFF), которой присвоено название STORE.

```
        STORE равно 7CFF
Адрес  Машинный код  Мнемокод        Комментарии
7D00   3E 00         LD A,+xx        Введите число
7D02   32 FF 7C      LD (STORE),A    Введите текущее значение
                                     в ячейку STORE
7D05   C9            RET             "Возврат к Бейсику"
```

В программе 8 пользователь,как и ранее, должен ввести число для пересылки в ячейку STORE. При выполнении программы написанной на машинном языке , используются RANDOMIZE USR 32000, и содержимое STORE возвращается пользователем по команде PEEK 31999.

Программа 8. LD (addr),A

Строки 10-40. Шестнадцатиричный загрузчик, приведенный на стр.92.

```
50 DATA "3E", "00"
51 DATA "32","FF","7C"
52 DATA "C9"," "
60 INPUT "Enter a value for location STORE
         (0-255 only)";CHR$32;N
70 CLS
80 POKE 32031,N: RANDOMIZE USR 32000
90 PRINT AT 10,0; "The location STORE now holds";CHR$32;
   PEEK 3999
100 GO TO 60
```

Во второй программе раздела показан случай косвенной адресации. двойной регистр HL адресует ячейку STORE, и для пересылки содержимого этой ячейки используется команда "LD (HL),E"

STORE равно 7CFF

Адрес	Машинный код	Мнемокод	Комментарии
7D00	1F 00	LD A,+xx	Введите число
7D02	21 FF 7C	LD HL,STORE	Установить в двойном регистре HL адрес STORE
7D05	73	LD (HL),E	Произвести пересылку
7D05	C9	RET	Возврат к Бейсику

Программа 9.LD (HL),E

Строки 10-40.Шестнадцатиричный загрузчик, приведенный на стр.92.

```
50 DATA "1E","00"
51 DATA "21","FF","7C"
52 DATA "73","C9"," "
```

Строки 60-100 - как в программе 8.

В третьей программе раздела используется индексная адресация . Ячейке 32061 (шестнадцатиричный эквивалент 7D3D) присваивается имя BASE , в силу чего ячейка STORE рассматривается как "BASE-3E".

BASE равно 7D3D
STORE равно BASE-3E

Адрес	Машинный код	Мнемокод	Комментарии
7D00	3E 00	LD A,+xx	Введите число
7D02	DD 21 3D 7D	LD IX,+BASE	Установить в двойном регистре IX адрес BASE
7D06	DD 77 C2	LD (IX-3E),A	"IX-3E" берется как "IX+C2"
7D09	C9	RET	"Возврат к Бейсику"

В программе 10 пользователь, как и ранее, должен ввести число для пересылки в ячейку STORE и считывания обратно по команде PEEK 31999.

Программа 10. LD (IX+d),A

Строки 10-40. Шестнадцатиричный загрузчик, приведенный на стр. 92.

```
50 DATA "3E","00"
51 DATA "DD","21","3D","7D"
52 DATA "DD","77","C2"
53 DATA "C9","  "
```

В программах 5-10 используются однобайтовые номера, но читатель легко может изменить эти программы для работы с двухбайтовыми номерами и соответствующими командами для двойных регистров

Группа 6. Команды сложения.

Команды этой группы позволяют пользователю складывать числа (ADD), увеличивать имеющееся значение на единицу (INC) и складывать числа с переносом (ADC).

В первой программе раздела показан случай использования команды "ADD A,B".

Адрес	Машинный код	Мнемокод	Комментарии
7D00	00	NOP	Для последующего использования
7D01	3E 00	LD A,+xx	ввести в указанные регистры
7D03	06 00	LD B,+xx	числа

```
7D05    80          ADD A,B     Выполнить сложение
7D06    06 00       LD B,+00    Обнулить регистр B
7D08    4F          LD C,A      Пересылка результата
7D09    C9          RET         Возврат к Бейсику
```

В программе 11 пользователь должен ввести два числа.Эти числа пересылаются в регистры A и B и складываются по правилам абсолютной двоичной арифметики.Результат возвращается с помощью функции USR.

Программа 11. ADD A,B

Строки 10-40. Шестнадцатиричный загрузчик, приведенный на стр. 92.

```
50  DATA  "00","3E","00"
51  DATA  "06","00","80"
52  DATA  "06","00","4F"
53  DATA  "0C"," "
60  INPUT  "Enter a first value (0-255)";CHR$32;F
70  INPUT  "Enter a second value (0-255)";CHR$32;S
80  CLS
90  POKE 32002,F: POKE 32004,S
100  PRINT AT 10,5;F;CHR$32;"ADD";CHR$32;S;CHR$32;
     "=";CHR$32;USR 32000
110  GO TO 60
```

Во второй программе раздела используется команда "INC BC".

```
Адрес  Машинный  Мнемокод          Комментарии
       код
7D00   01 00 00  LD BC,+xxxx       Ввести в регистр BC
                                   нужное значение
7D03   03        INC BC            Увеличить его на 1
7D04   C9        RET               "Возврат к Бейсику"
```

В программе 12 пользователь должен ввести число из интервала 0-65535. Это число разбивается на старшую и младшую части и по команде POKE пересылается в ячейки с шестнадцатиричными адресами 7D01 и 7D02. Затем содержимое двойного регистра BC увеличивается на 1, и полученное значение возвращается с помощью функции USR.

Следует отметить своеобразие ввода числа 65535.

Программа 12.

Строки 10-40. Шестнадцатиричный загрузчик, приведенный на стр.92.

```
 50 DATA "01","00","00"
 51 DATA "03","C9","  "
 60 INPUT  "Enter a value (0-65535)";CHR$32;N
 70 POKE 32002,INT(N/256)
 80 POKE 32001,N-256*INT(N/256)
 90 CLS
 100 PRINT AT 10,0;N;"Increments to give";
     CHR$32;USR 32000
 110 GO TO 60
```

В третьей программе раздела показано использование команды "ADC A,B". (Это такая же программа , как и 11* единственное изменение- вместо "ADD A,B" включено "ADD A,B").

Адрес	Машинный код	мнемокод	Комментарии
7D00	37	SCF	Установить флаг ПЕРЕНОС
7D01	3E 00	LD A,+xx	Ввести заданное число
7D03	06 00	LD B,+xx	в регистры.
7D05	88	ADC A,B	Сложение с переносом
7D06	06 00	LD B,+00	Обнуление регистра B
7D08	4F	LD C,A	Переслать результат
7D09	C9	RET	"Возврат к Бейсику"

Этой программе соответствует написанная на Бейсике программа 13 , в которой пользователь должен вести два числа для сложения с переносом. Флажок ПЕРЕНОС при этом всегда установлен.

Посмотрите, что произойдет в результате замены команды SCF на AND A (шестнадцатиричный код A7), которая сбрасывает флажок ПЕРЕНОС.

Программа 13. ADS A,B

Строки 10-40. Шестнадцатиричный загрузчик, приведенный на стр.92.

```
 50 DATA "37","3E","00"
 51 DATA "06","00","88"
 52 DATA "06","00","4F"
 53 DATA "C9","  "
 60 INPUT "Enter a first value (0-255)";CHR$32;F
 70 INPUT "Enter a second value (0-255)";CHR$32;S
 80 CLS
 90 POKE 32002,F:POKE 32004,S
```

```
100 PRINT AT 10,0;"With carry set ";CHR$32;F;
    CHR$32;"ADC";CHR$32;S;CHR$32;"=";CHR$32;USR 32000
```

Группа 7.Команды вычитания

Команды этой группы позволяют пользователю производить обычное вычитание (SUR), уменьшение на 1 (DEC) и вычитание с переносом (SBC).

В первой программе раздела приведен пример использования команды "SUB B". Определяемое после вычитания значение флажка ПЕРЕНОС (0 или 1) сохраняется в ячейке STORE.

STORE равно 7CFF

Адрес	Машинный код	Мнемокод	Комментарии
7D00	00	NOP	Для последующего использования
7D01	3E C0	LD A,+xx	Ввести в регистры
7D03	06 00	LD B,+xx	два числа
7D05	90	SUB B	Вычитание
7D06	06 00	LD B,+00	Обнулить регистр B
7D08	4F	LD C,A	Пересылка результата
7D09	3E 00	LD A,+00	Обнуление регистра A
7D0B	CE 00	ADC A,+00	Сложение с переносом
7D0D	32 FF 7C	LD (STORE),A	Пересылка в STORE значения флажка ПЕРЕНОС
7D10	C9	RET	"Возврат к БЕЙСИКУ"

Этой программе соответствует написанная на Бейсике программа 14, в которой пользователь должен ввести значения для регистров A и B.Значение флажка ПЕРЕНОС возвращается по команде PEEK 31999.

Программа 14. SUB B

Строки 10-40.Шестнадцатиричный загрузчик, приведенный на стр.92.

```
50 DATA "00","3E","00"
51 DATA "06","00","90"
52 DATA "06","00","4F"
52 DATA "3E","00","CE","00"
54 DATA "32","FF","7C"
55 DATA "C9"," "
60 INPUT "Enter a first value (0-255)";CHR$32;F
70 INPUT "Enter a second value (0-255)";CHR$32;S
80 CLS
90 POKE 32002,F:POKE 32004,S
100 PRINT AT 10,0;F;CHR$32;"SUB";CHR$32;S;CHR$32;"=";
```

```
    CHR$32;USR 32000;CHR$32;"With carry";CHR$32;"set";
    AND PEEK 31999;"reset";AND NOT PEEK 31999
110 GO TO 60
```

Используйте эту программу с числами, которые "больше", "меньше" и "равны".

Во вторую подгруппу входят команды DEC.

Действие команды "DEC BC" можно показать, внеся соответствующие изменения в программу 12. Для этого нужно изменить строки 51 и 100 следующим образом.

```
51 DATA "CB","C9","  "
100 PRINT AT 10,0;N;CHR$32;"Decrement to give";
    CHR$32;USR 32000
```

В третью группу входят команды SBC.

Действие команды "SBC A,B" можно показать, внеся соответствующие изменения в программу 14. Для этого нужно изменить строки 50,51 и 100 следующим образом.

```
50 DATA "37","3E","00" - для установки флажка ПЕРЕНОС
   или
50 DATA "A7","3E","00" - для сброса флажка ПЕРЕНОС
51 DATA "06","00","98" - шестнадцатиричный код команды
                         "SBC A,B"-98.
```

В строке 100 следует изменить SUB на SBC.

Испробуйте эту программу с числами, которые "больше", "меньше" и "равны".

Заметим, что результаты выполнения команды SUB и команды SBC при сброшенном флажке ПЕРЕНОС не отличаются.

Группа 8. Команды сравнения.

Команды сравнения действуют так же, как команды SUB, за исключением того, что содержимое регистра A остается неизменным. При этом используется флажок ПЕРЕНОС.

Действие команды "CP B" можно показать, внеся соответствующие изменения в программу 12. Для этого нужно изменить строки 51 и 100 следующим образом.

```
51 DATA "06","00","B8"  Шестнадцатиричный код
                        команды "CP B" - B8.
100 PRINT AT 10,0;F;
    CHR$32;"CP";CHR$32;S;CHR$32;"gives carry";CHR$32
```

и прибавить строки 101 и 102 :

```
101 RANDOMIZE USR 32000
102 PRINT "re" AND NOT PEEK 31999;"set"
```

Команды этой группы позволяют пользователю производить логические операции AND (И), OR (ИЛИ) и XOR (ИСКЛЮЧАЮЩЕЕ ИЛИ) с двумя 8-разрядными числами.

В приведенной ниже программе показано использование логической команды AND с двумя числами, введенными пользователем.

Адрес	Машинный код	Мнемокод	Комментарии
7D00	3E 00	LD A,+xx	Ввести в указанные
7D02	06 00	LD B,+xx	регистры числа
7D04	A0	AND B	Логическое И
7D05	32 FF 7C	LD (STORE),A	Копирование результата в "STORE"
7D08	C9	RET	"Возврат к Бейсику"

Этой программе соответствует написанная на Бейсике программа 15. После введения каждого числа оно отображается в двоичном виде. Результат извлекается из STORE с помощью команды PEEK и также отображается в двоичном виде.

Программа 15. AND B

Строки 10-40. Шестнадцатиричный загрузчик, приведенный на стр.92

```
50 DATA "3E","00"
51 DATA "06","00"
52 DATA "A0"
53 DATA "32","FF","7C"
54 DATA "C9","  "
60 INPUT "Enter a first value (0-255)";CHR$32;F
70 CLS
80 POKE 32001,F
90 PRINT AT 8,4;:GO SUB 300
100 PRINT AT 10,8;"AND"
110 INPUT "Enter a second value (0-255)";CHR$32;S
120 POKE 32003,S
130 PRINT AT 12,4;:LET F=S: GO SUB 300
140 PRINT AT 14,7;"givas"
150 RANDOMISE USR 32000
160 PRINT AT 16,4;:LET F=PEEK 31999: GO SUB 300
170 GOTO 60
300 REM Binary of F
310 FOR N=7 TO 1 STEP -1
320 LET P=2**N
```

```
330 PRINT CHR$(48+INT(F/P));CHR$32;
340 LET F=F-INT(F/P)*P
350 NEXT N
360 PRINT INT F
370 RETURN
```

Программу 15 можно приспособить для демонстрации команд "OR B" и "XOR B". В частности, следует внести следующие изменения в строки 52 и 100 для демонстрации команды "OR B":

```
52 DATA "B0"
100 PRINT AT 10,8;"OR"
```

и для демонстрации команды "XOR B":

```
52 DATA "A8"
100 PRINT AT 10,8;"XOR B"
```

Советуем, читатель, хорошо изучить эти три логические команды.

Группа 10. Команды перехода.

Семнадцать команд этой группы позволяют пользователю совершать переходы между частями программы, написанной на машинном языке. Переходы могут быть относительными, т.е. на указанное число ячеек (от-128 до+127) от ячейки с текущим адресом, расположенным в счетчике команд, или абсолютными, т.е. в ячейку с указанным адресом. Также разрешены условные переходы по состоянию одного из основных флажков - но только с имеющимися командами безусловного перехода.

Для демонстрации команд этой группы в программе 16 можно использовать следующую программу , написанную на машинном языке.

NEXT равно 7D0F

Адрес	Машинный код	Мнемокод	Комментарии
7D00	3E 00	LD A,+xx	Ввести два числа и сравнить
7D02	FE 00	CP +xx	их для установки флажков
7D04	01 00 00	LD BC,+0000	Обнулить BC
7D07	18 06	JR NEXT	Переход вперед к NEXT
7D09	00	NOP	Для последующего исполь-
7D0A	C9	RET	зования
7D0B	00 00 00 00	-	4 неиспользуемые ячейки
7D0F	03	INC BC	Увеличить после перехода содержимое BC на 1
7D10	C9	RET	"Возврат к Бейсику"

В результате выполнения этой программы в регистре BC будет содержаться 0, если перехода не было, и 1 - если переход был.

В программе 16 показано, как эта программа, написанная на машинном языке, используется для демонстрации команды "JR e". Это - команда безусловного перехода, так что переход не будет зависеть от результата проверки, заданной пользователем.

Программа 16. JR e

Строки 10- 40. Шестнадцатиричный загрузчик, приведенный на стр. 92.

```
 50 DATA "3E","00"
 51 DATA "FE","00"
 52 DATA "01","00","00"
 53 DATA "18","06","00"
 54 DATA "C9"
 55 DATA "00","00","00","00"
 56 DATA "03","C9"
 60 PRINT AT 4,0;"Instruction -"
 70 PRINT "JR e"
 80 PRINT AT 8,0;"Test-"
 90 INPUT "Enter a first value (0-255) ";CHR$32;F
 100 PRINT F;CHR$32;"CP";CHR$32;
 110 INPUT "Enter a second value (0-255) ";CHR$32;S
 120 PRINT S
 130 PRINT at 12,0; "jump -"
 140 POKE  320001,F: POKE 32003,S
 150 LET R=USR 32003: PRINT "N" AND NOT R; "YES" AND R
 160 PRINT AT 21,0;"Any key to continue"
 170 PAUSE 50
 180 IF INKEY$="" THEN GO TO 180
 190 CLS: GO TO 60
```

В этой программе пользователь должен ввести два числа. Затем, после операции сложения, в случае необходимости осуществляется переход. При отсутствии перехода переменная R приравнивается 0; в случае перехода она равняется 1.

Программа написана таким образом, что пользователь может, изменив строки 53 и 70, демонстрировать четырнадцать из семнадцати команд перехода.

Для демонстрации команд относительного перехода следует внести следующие изменения.

'JR NZ,e'	53 DATA "20","06","00" 70 PRINT "JR NZ,e"
'JR Z,e'	53 DATA "28","06","00" 70 PRINT "JR Z,e"

```
'JR NC,e'          53 DATA "30","06","00"
                   70 PRINT "JR NC,e"

'JR C,e'           53 DATA "38","06","00"
                   70 PRINT "JR C,e"
```

Для демонстрации команд абсолютного перехода нужно изменить программу, написанную на машинном языке, таким образом, чтобы задавался переход в ячейку памяти NEXT с шестнадцатиричным адресом 7D0F.

Соответствующая строка команды на машинном языке будет выглядеть так:

Адрес	Машинный код	Мнемокод	Комментарии
7D07	03 0F 7D	JP NEXT	Переход вперед к NEXT

В программу 16 следует внести следующие изменения.

```
'JP addr'          53 DATA "C3","0F","7D"
                   70 PRINT "JP addr"

'JP NZ,addr'       53 DATA "C2","0F","7D"
                   70 PRINT "JP NZ,addr"

'JP Z,addr'        53 DATA "CA","0F","7D"
                   70 PRINT "JP Z,addr"

'JP NC,addr'       53 DATA "D2","0F","7D"
                   70 PRINT "JP NC,addr"

'JP C,addr'        53 DATA "DA","0F","7D"
                   70 PRINT "JP C,addr"

'JP PO,addr'       53 DATA "E2","0F","7D"
                   70 PRINT "JP PO,addr"

'JP PE,addr'       53 DATA "EA","0F","7D"
                   70 PRINT "JP PE,addr"

'JP P,addr'        53 DATA "F2","0F","7D"
                   70 PRINT "JP P,addr"

'JP M,addr'        53 DATA "FA","0F","7D"
                   70 PRINT "JP M,addr"
```

В остальных трех командах этой группы используется косвенная адресация; читателю предлагается самому внести изменения

в программу, позволяющие демонстрировать действие этих команд.

Группа 11. Команда "DJNZ,e"

Команда "DJNZ,e" очень часто оказывается довольно полезной, поскольку ее легко использовать для организации простых циклов в программе, написанной на машинном языке.

Для использования команды "DJNZ,e" программист должен вначале задать число необходимых циклов, затем скопировать это число в регистр B. После этого задается тело цикла, а потом - команда "DJNZ,e", действующая подобно "NEXT B".

Ниже приводится программа, состоящая из четырех строк, написанная на Бейсике, и предназначенная для вывода алфавита (заглавных букв). После заполнения экрана появляется сообщение "scroll?".

```
10 FOR A 65 TO 90
20 PRINT CHR$ A;
30 NEXT A
40 GO TO 10
```

Такую же операцию можно задать с помощью приведенной ниже программы на машинном языке. Заметим, что программист может использовать лишь "STEP - 1".

LOOP равно 7D02

Адрес	Машинный код	Мнемокод	Комментарии
7D00	06 1A	LD B,+1A	Задается 25 циклов -
			обратно от команды
7D02	3E 5B	LD A,+5B	"A" соответствует шестнад-
7D04	90	SUB B	цатиричное 5B - 1A
			Значение находится в рег. A
7D05	D7	RST 0010	Печать символа - см. дальше
7D06	10 FA	DJNZ,LOOP	Обратно для печати следующей
			буквы (если нужно), иначе -
7D08	C9	RET	"возврат к Бейсику"

Этой программе соответствует программа 17 предназначенная для вывода алфавита на экран телевизионного дисплея. Для неоднократной печати алфавита используются повторные вызовы "USR 32000".

Программа 17. DJNZ,e

Строки 10-40. Шестнадцатиричный загрузчик, приведенный на стр. 92.

```
50 DATA "06","1A"
51 DATA "3E","5B"
52 DATA "90"
53 DATA "D7"
54 DATA "10","FA"
55 DATA "C9"
60 PRINT
70 RANDOMIZE USR 32000
80 GO TO 70
```

Замечание: Важным является введение строки 60, задающей открытие канала дачи данных в "основную" область памяти телевизионного дисплея. Если эта команда (или что-либо подобное) отсутствует, вывод производится в область редактирования.

Группа 12. Команды стека

В этой группе команд имеется пять подгрупп. В первую подгруппу входят команды PUSH и POP, во вторую - команды обмена со стеком.

Для демонстрации команд первой группы в программе 18 используется следующий способ.

Адрес	Машинный код	Мнемокод	Комментарии
7D00	21 00 00	LD HL,+xxxx	Ввод числа
7D03	E5	PUSH HL	Занесение его в стек
7D04	C1	POP BC	Теперь оно в BC
7D05	C9	RET	"Возврат к Бейсику"

В программе 18 пользователь должен ввести число. Затем это число копируется в двойной регистр HL, и в дальнейшем пересылается в стек. Окончательно это число копируется из стека в двойной регистр BC.

Программа 18. PUSH HL; POP HL

Строки 10-40. Шестнадцатиричный загрузчик, приведенный на стр. 92.

```
50 DATA "21","00","00"
51 DATA "E5","C1"
55 DATA "C9","  "
60 INPUT  "Enter a value (0-255)";CHR$32;F
```

```
70 POKE 32001,F-INT(F/256)*256: ROKE 32002,INT(F/256)
80 CLS
90 PRINT AT 10,0;"Value taken off stack=";CHR$32;
   USR 32000
100 GO TO 60
```

Читатель может испробовать разные варианты этой программы. При этом следует соблюдать осторожность, чтобы число занесений в стек равнялось числу извлечений из стека.

В третью и четвертую подгруппы входят команды CALL и RET.

Для демонстрации различных команд CALL можно использовать следующую программу, написанную на машинном языке.

NEXT равно 7D0F

Адрес	Машинный код	Мнемокод	Комментарии
7D00	3E 00	LD A,+xx	Ввести два числа и сравнить их
7D02	FE 00	CP +xx	
7D04	01 00 00	LD BC,+0000	Обнулить BC
7D07	CD 0F 7D	CALL NEXT	Вызов подпрограммы
7D0A	C9	RET	"Возврат к Бейсику"
7D0B	00 00 00 00	-	4 неиспользуемые ячейки
7D0F	03	INC BC	Теперь в BC - 1
7D10	C9	RET	Возврат из подпрограммы

Для других команд в программе 16 можно использовать этот блок со следующими изменениями.

```
                130 PRINT AT 12,0; "CALL  ",

        'CALL addr'     53 DATA "CD","0F","7D"
                        70 PRINT "CALL addr"

        'CALL NZ,addr'  53 DATA "C4","0F","7D"
                        70 PRINT "CALL NZ,addr"

        'CALL Z,addr'   53 DATA "CC","0F","7D"
                        70 PRINT "CALL Z,addr"

        'CALL NC,addr'  53 DATA "D4","0F","7D"
                        70 PRINT "CALL NC,addr"

        'CALL C,addr '  53 DATA "DC","0F","7D"
                        70 PRINT "CALL C,addr"

        'CALL PO,addr'  53 DATA "E4","0F","7D"
                        70 PRINT "CALL PO,addr"
```

```
'CALL PE,addr'  53 DATA "EC","0F","7D"
                70 PRINT "CALL PE,addr"

'CALL P,addr'   53 DATA "F4","0F","7D"
                70 PRINT "CALL P,addr"

'CALL M,addr'   53 DATA "FC","0F","7D"
                70 PRINT "CALL M,addr"
```

Для демонстрации действия команды RET можно использовать следующую программу, написанную на машинном языке. Однако соответствующие изменения в программе 16 читателю предлагается продумать самому.

Адрес	Машинный код	Мнемокод	Комментарии
7D00	3E 00	LD A,+xx	Ввести два числа и срав-
7D02	FE 00	CP +xx	нить их
7D04	01 01 00	LD BC,+0001	В регистр BC заносится 1
7D07	D8	RET C	Возврат если флажок
7D08	0B	DEC BC	ПЕРЕНОС установлен
7D09	C9	RET	обычный возврат

Замечания: В этой программе демонстрируется команда "RET C". Логическая схема изменена так, что прогр.16 будет давать на выходе "YES" при условном переходе. Команда RST будут рассматриваться в главах 7 и 8.

Группа 13. Команды циклического сдвига.

В системе команд микропроцессора Z80 имеется много команд циклического сдвига.

Для демонстрации семи типов циклического сдвига содержимого регистра C используется следующая программа, написанная на машинном языке.

STORE равно 7CFF

Адрес	Машинный код	Мнемокод	Комментарии
7D00	AF	XOR A	Очистить регистр A
7D01	FE 00	CP +xx	Сравнение с числом 0-флажок ПЕРЕНОС сбрасывается; 1-устанавливается
7D03	06 00	LD B,+xx	Обнулить регистр B
7D05	0E 00	LD C,+xx	Ввод числа в C
7D07	CB 01	RLC C	Циклический сдвиг
7D09	3E 00	LD A,+00	Обнулить регистр A

```
7D0B     CE 00     ADC A,+00       Пересылка значения флажка
7D0D     32 FF 7C  LD (STORE),A    ПЕРЕНОС в STORE
7D10     C9        RET             "Возврат к Бейсику"
```

В программе 19 пользователь должен ввести 0 или 1 - для установки или сброса флажка ПЕРЕНОС. Затем следует ввести некоторое число. Оно копируется в регистр C и циклически сдвигается - если нужно. Результат печатается вместе с текущим значением флажка ПЕРЕНОС.

Программа 19. RLC C

Строки 10 - 40. Шестнадцатиричный загрузчик, приведенный на стр. 92.

```
50 DATA "AF","FE","00"
51 DATA "06","00","0E","00"
52 DATA "CB","01"
53 DATA "3E","00","CE","00"
54 DATA "32","FF","7C"
55 DATA "C9","  "
60 PRINT AT 2,0;"Instruction -"
70 PRINT "RLC C"
80 INPUT "Carry reset or set(0/1)?";CHR$32;C
90 POKE 32002,C
100 PRINT AT 6,0;"CARRY -",C
110 INPUT "Enter a value (0-255)";CHR$32;F
120 POKE 32006,F
130 PRINT AT 10,0; "Initial value";: GO SUB 300
140 LET F=USR 32000
150 PRINT AT 14,0;"Final value";: GO SUB 300
160 PRINT AT 18,0;"CARRY -",PEEK 31999
170 PRINT AT 21,0;"Any key to continue"
180 PAUSE 50
190 IF INKEY$="" THEN GO TO 190
200 CLS: GOTO 60
300 REM Binary of F
310 FOR N=7 TO 1 STEP -1
320 LET P=2**N
330 PRINT CHR$(48+INT(F/P));CHR$32;
340 LET F=F-INT(F/P)*P
350 NEXT N
360 PRINT INT F
370 RETURN
```

Программу 19 можно использовать для демонстрации семи команд, которые задействуют регистр C.

Для этого необходимо внести следующие изменения.

```
'RLC C'              52 DATA  "CB","01"
                     70 PRINT "'RLC C'"
'RRC C'              52 DATA  "CB","09"
                     70 PRINT "'RRC C'"
'RL C'               52 DATA  "CB","11"
                     70 PRINT "'RL C'"
'RR C'               52 DATA  "CB","19"
                     70 PRINT "'RR C'"
'SLA C'              52 DATA  "CB","21"
                     70 PRINT "'SLA C'"
'SRA C'              52 DATA  "CB","29"
                     70 PRINT "'SRA C'"
'SRL C'              52 DATA  "CB","39"
                     70 PRINT "'SRL C'"
```

По желанию используемую в примере 19 программу, написанную на машинном языке, можно изменить для демонстрации четырех однобайтных команд.

Группа 14. Команды обработки бит.

Команды этой группы можно разбить на три подгруппы.

Команды RES и SET используются не очень часто, и здесь программа для их демонстрации не приводится. Читатель может сам написать простые программы, использующие эти команды.

Несомненно, наиболее полезные команды этой группы - команды BIT, и ниже приводится программа, показывающая использование команды "BIT 7,H".

Программа 20 осуществляет вывод в двоичном представлении. Пользователь должен ввести число из интервала 0 - 65535. Затем для вывода на печать двоичного представления этого числа используется соответствующая программа, написанная на машинном языке.

Число, введенное пользователем, пересылается в двойной регистр HL; бит 7 регистра H считывается, и печатается соответственно 0 или 1. Затем содержимое регистров HL сдвигается влево, перенос из L захватывается в H. Опять считывается бит 7 регистра H, и полученное значение выводится на печать. Для получения полного двоичного представления эта операция (считывание и сдвиг) выполняется 16 раз.

Соответствующая программа имеет вид:

```
        LOOP  равно 7D05
        PRINT равно 7D0C
Адрес   Машинный   Мнемокод     Комментарии
        код
7D00    21 00 00   LD HL,+xxxx  Ввод числа
7D03    06 10      LD B,+10     Имеется 16 бит
```

```
7D05     CB 7C      BIT 7,H       Проверка самого левого бита
7D07     3E 30      LD A,'0'      Подготовка печати нуля
7D09     28 01      JR Z,PRINT    Переход, если необходим 0
7D0B     3C         INC A         Теперь - символ "1"
7D0C     D7         RST 0010      Печать символа - 0 или 1
7D0D     3E 20      LD A, 'sp.'   Подготовка печати "пробела"
7D0F     D7         RST 0010      Печать "пробела"
7D10     CB 15      RL L          Циклический сдвиг L
7D12     CB 14      RL H          Циклический сдвиг H с зах-
                                  ватом переноса
7D14     10 EF      DJNZ,LOOP     Обратно - до окончания
7D16     C9         RET           "Возврат к Бейсику"
```

В этой программе, написанной на машинном языке соответствует программа 20.

Программа 20. BIT 7,H

Строки 10-40. Шестнадцатиричный загрузчик, приведенный на стр.92.

```
50 DATA "21","00","00"
51 DATA "06","10","CB","7C"
52 DATA "3E","30","28","01"
53 DATA "3C","D7"
54 DATA "3E","20","D7"
55 DATA "CB","15","CB","14"
56 DATA "10","EF","C9"," "
60 INPUT "Enter a value (0-255)";CHR$32;F
70 POKE 32001,F-INT (F/256)*256
80 POKE 32002,INT (F/256)
90 CLS
100 PRINT AT 10,8;"Binary of";CHR$ 32;F
110 PRINT AT 12,14; "is"
120 PRINT AT 14,0;
130 RANDOMISE USR 32000
140 GO TO 60
```

Группа 15. Команды обработки блока

Команды этой группы позволяют пользователю перемещать или исследовать блоки данных.

Несомненно, чаще всего в этой группе используется команда LDIR.

Ниже приводится написанная на машинном языке программа, в которой эта команда используется для копирования верхней трети области дисплея на среднюю треть. Это означает, что при каждом вызове рассматриваемой программы все байты ячеек с шестнадцатиричными адресами 4000-47FF копируются в ячейки

с шестнадцатиричными адресами 4800-4FFF. При этом все символы из строк 0-7 дублируются в строки 8-15.

Эта программа имеет следующий вид:

Адрес	Машинный код	Мнемокод	Комментарии
7D00	21 00 40	LD HL,+4000	Верхний левый угол экрана
7D03	11 00 48	LD DE,+4800	Начало строки 8
7D06	01 00 08	LD BC,+0800	Имеется 2048 ячеек
7D09	ED B0	LDIR	Перемещение блока
7D0B	C9	RET	"Возврат к Бейсику"

Этой программе,написанной на машинном языке, соответствует программа 21.

Программа 21. LDIR

Строки 10-40. Шестнадцатиричный загрузчик,приведенный на стр.92

```
 50 DATA "21","00","40"
 51 DATA "11","00","48"
 52 DATA "01","00","08"
 53 DATA "ED","B0"
 54 DATA "C9","  "
 90 INPUT  "Enter your characters - 8 Lines";C$
100 PRINT C$
150 RANDOMISE USR 32000
```

Отметим, что для дублирования областей экрана можно использовать "LDIR" или "LDDR",но аппаратное окружение обычно использует эти команды для других видов копирования,при котором задействуются области символов целиком,но не частично.

Ниже приводится программа,демонстрирующая использование команды "CPIR" Программа 22, соответствующая этой программе на машинном языке,определяет адрес первой ячейки постоянной памяти, в которой содержится заданный байт. После выполнения программы можно убедиться,что в постоянной памяти находятся все числа из интервала 0-255,но десятичное число 154 впервые появляется в ячейке 11728 и сравнительно редко появляется в рассматриваемой конфигурации постоянной памяти.

Программа,написанная на машинном языке,имеет следующий вид.

Адрес	Машинный код	Мнемокод	Комментарии
7D00	3E 00	LD A,+xx	Сравниваемое число
7D02	01 FF 3F	LD BC,+3FFF	Верхняя ячейка ПЗУ

```
7D05   21 00 00    LD HL,+0000  Первая -  " "  -
7D08   ED B1       CPIR         Исследовать пост.память
7D0A   44          LD B,H       Переместить старший байт адр
7D0B   4D          LD C,L       Переместить младший байт адр
7D0C   0B          DEC BC       Сохранение адр.иск.ячейки
7D0D   C9          RET          "Возврат к Бейсику"
```

Программа 22, соответствующая этой программе, имеет вид:

Программа 22. CPIR

Строки 10-40 "Шестнадцатиричный загрузчик приведенный на стр. 92.

```
50 DATA "3E","00"
51 DATA "01","FF","3F"
52 DATA "21","00","00"
53 DATA "ED","B1","44","4D"
54 DATA "0B","C9"," "
60 FOR F=0 TO 255
70 POKE 32001,F
80 PRINT F;TAB  ; " occurs first at loc";
   CHR$32; USR 32000
90 NEXT F
```

Читатель может попытаться изменить эти программы для демонстрации неавтоматических команд.

Здесь не приводятся демонстрационные программы для команд ввода-вывода,команд прерывания и небольшой группы остальных команд, но, как и ранее ,пользователь может попытаться написать соответствующие программы самостоятельно.