<< И Н Ф О Р К О М >>

ПЕРСОНАЛЬНЫЙ КОМПЬЮТЕР

"ZX-SPECTRUM"

Программирование в машинных кодах

и на языке АССЕМБЛЕРА

Часть 1.
" Первые шаги в машинных кодах"

Часть 2.
"Практикум по программированию в машинных кодах"

Часть 3.
"Справочник по программированию в машинных кодах"

Москва 1993

+

Книга является наиболее доступным изданием по программированию в машинных кодах для широко распространенных персональных компьютеров "ZX-Spectrum" (система "Синклер"). В доступной для начинающих форме рассмотрена система команд процессора, описан встроенный калькулятор, разобраны многочисленные примеры. Книга содержит интересные рекомендации по просмотру машинного кода фирменных программ.

Предназначена для самостоятельного обучения.

ОТ АВТОРОВ

Книга, которую Вы держите в руках, уже получила широкую известность, как самый доступный самоучитель для тех, кто хочет преодолеть психологический барьер и перейти от программирования на БЕЙСИКе к программированию в машинном коде или просто хочет понимать машинный код фирменных программ.

Первое издание этой книги мы выпустили два года назад, в 1990 году, и сейчас в стране уже есть тысячи любителей бытовых персональных компьютеров типа ZX_Spectrum (система "Синклер"), самостоятельно освоивших тонкости машинного программирования по этой книге. Первое издание было выпущено в трех томах. Сейчас мы объединили все три тома (учебник, практикум и справочник) в рамках единой книги, несколько подкорректировали и значительно (примерно на 20%) дополнили содержание. Основным дополнением стали разделы, посвященные описанию системы прерываний компьютера, разбору концепции потоков и каналов, понятию о директивах АССЕМБЛЕРа и вопросам, связанным со стандартизацией русификации компьютеров. Как показал первый опыт, именно эти вопросы нуждались в усиленном освещении.

"ИНФОРКОМ" продолжает работу над книгами, посвященными компьютерам "Спектрум" и совместимым с ними. В настоящий момент начат выпуск многотомного издания, посвященного работе с графикой "Спектрума". Несмотря на то, что каждый из этих томов является самостоятельной учебной единицей и может быть использован независимо от прочих, тем не менее, их можно считать логическим продолжением данного издания уже хотя бы потому, что "Первые шаги в машинном коде Z-80" в силу естественных причин стали базовой книгой для наших последующих разработок.

"ИНФОРКОМ" благодарит всех читателей первого издания, приславших свои отзывы, пожелания и рекомендации и особо выражает персональную признательность своим корреспондентам Баянову К.Н. за подготовку разделов I.5.18, II.4.4.7, III.5 и Пашорину В.И. за подготовку раздела II.5.5.

"ИНФОРКОМ".

Москва, август 1992.

С О Д Е Р Ж А Н И Е

Часть. II

ПРАКТИКУМ ПО ПРОГРАММИРОВАНИЮ
В МАШИННЫХ КОДАХ

Часть. III

СПРАВОЧНИК ПО ПРОГРАММИРОВАНИЮ
В МАШИННЫХ КОДАХ

ВВЕДЕНИЕ

Проведенное в начале 1990 года анкетирование наших заказчиков показало их глубокую заинтересованность в освоении программирования в машинных кодах для Синклер-совместимых компьютеров, получивших в нашей стране наибольшее распространение среди ПЭВМ бытового класса.

Процессор Z-80, на базе которого собраны компьютеры этой системы, приобрел широкую популярность в мире благодаря своей универсальности, наличию обширной системы команд и технологичности производства, обеспечившей ему большие объемы выпуска при сравнительно невысокой цене.

Этот процессор применяется не только в многочисленных компьютерах, входящих в систему "Синклер-Спектрум", но и в компьютерах других систем. Среди них компьютеры семейства MSX ("ЯМАХА", "СПЕКТРОВИДЕО", "ТОШИБА", "ПАНАСОНИК" и др.), компьютеры систем "ЭНТЕРПРАЙЗ", "ШАРП", многие компьютеры фирмы "АМСТРАД" и пр. В принципе, материалы этой книги могут быть на 90% использованы и теми, кто работает с компьютерами этих систем.

В настоящее время процессор Z-80 очень хорошо документирован в мировой литературе. Наиболее фундаментальными трудами для программистов любого уровня являются монографии Лэнса Левенталя (LANCE A.LEWENTHAL) "Программирование на Ассемблере Z-80" ("Z-80 ASSEMBLY PROGRAMMING"), а также "Процедуры на Ассемблере для процессора Z-80" ("Z-80 ASSEMBLY LANGUAGE SUBROUTINES).

Эти очень хорошие книги, к сожалению, весьма объемны (по 500-600 стр.) и не переведены на русский язык, что делает сомнительной возможность их широкого распространения у нас в ближайшие годы.

Многочисленные прочие книги зарубежных авторов, посвященные программированию для Синклер-совместимых компьютеров в машинном коде, имеют в качестве недостатков либо недостаточную систематичность изложения, либо повышенную инструктивность подачи материала. В первом случае они оказываются неплохим подручным справочником, а во втором - неплохим учебником, но для тех, кто уже в принципе подготовлен. Учебником же для тех, кто самостоятельно начинает разбираться с самого начала, они могут

служить очень ограниченно.

В своей книге, предлагаемой Вашему вниманию, мы постарались, как сумели, сочетать популярность, систематичность и информативность изложения. Те, кто не нуждаются в элементарном освоении программирования в машинных кодах, могут сразу обратиться ко второй части "Практикум...". Чтобы не дублировать справочный материал, который необходим как тем, кто работает с первой частью, так и тем, кто работает со второй, мы вынесли его отдельно, в третью часть.

Мы очень рекомендуем сопровождать чтение этой книги самостоятельным просмотром кода фирменных программ с помощью какой-либо дисассемблирующей программы, например MONITOR 16/48, MONS 3, ULTIMON и т.п.

Мы должны также порекомендовать Вам наше периодическое издание "ZX-РЕВЮ". На его страницах Вы сможете найти почти все, что Вам может потребоваться из программного и информационного обеспечения компьютеров системы "Синклер" и кое-что, связанное с аппаратными вопросами.

У нас возможно приобретение полных годовых комплектов "ZX-РЕВЮ" за 1991 и за 1992 год, выполненных в виде отдельной книги. Выходят и выпуски 1993 года. Для справки укажем, что объем одного полного годового комплекта "ZX-РЕВЮ" примерно в 5 раз превосходит объем данной книги.

Свои пожелания, замечания и предложения Вы можете направить нам по адресу 121019, Москва, Г-19, а/я 16. Вы всегда сможете выбрать из списка выпущенной нами литературы то, что Вам будет по душе.

I. ПЕРВЫЕ ШАГИ В МАШИННЫХ КОДАХ

1. САМЫЙ - САМЫЙ ПЕРВЫЙ ШАГ...

Многие любители не испытывают серьезных трудностей в овладении БЕЙСИКом. Для этого достаточно сравнительно немного практики. Но рано или поздно они приходят к барьеру "машинного кода". Как это ни печально, но некоторые так перед ним и останавливаются. Это ни в коей мере не связано с отсутствием желания или способностей, просто многие не знают, с чего начать. Если в БЕЙСИКе можно начинать с чего угодно (при ошибке компьютер сам Вас поправит), то здесь Вы оказываетесь с процесором один на один, и такой метод проб и ошибок не срабатывает.

Одним словом, есть некий психологический барьер, который бывает трудно преодолеть в одиночку. Известно, что для того, чтобы научиться программировать, надо взять и начать программировать. "ИНФОРКОМ" предлагает Вам следующий компромиссный подход - сначала в рамках этой главы мы, беря "быка за рога", просто начнем программировать, а затем посвятим оставшуюся часть книги систематическому изложению материала.

Итак, давайте напишем первую программу в машинном коде. Прежде всего, выделим для нее область памяти. Если Вы читали нашу книгу "Большие возможности Вашего "Спектрума", то знаете, что для БЕЙСИКа в оперативной памяти компьютера отведена область памяти, начинающаяся с адреса, на который указывает системная переменная PROG и заканчивается адресом, на который указывает системная переменная RAMTOP. Предположим, что Вы хотите записать программу в машинных кодах, начиная с адреса 30000. Дайте команду CLEAR 29999. Эта команда установит RAMTOP в 29999 и Ваша программа будет защищена от возможной порчи из БЕЙСИКа. Даже если Вы дадите команду NEW, области памяти, находящиеся выше RAMTOP, не будут поражены.

Теперь дайте две прямые команды одну за другой:

```
POKE 30000,0
POKE 30001,201
```

Мы сейчас записали два числа в нужные нам адреса. Они образуют простейшую программу. Выполнить ее можно командой RANDOMIZE USR 30000. Попробуйте сами... Вам покажется, что ничего не произошло, но это не так. Сначала процессор обратился по адресу 30000 и нашел там число 0, которое обозначает машинный код операции NOP. Операция NOP (NO OPERATION - нет операции) дает команду процессору, что ничего делать не надо. В течение 0,0000014 сек. он действительно ничего не делает, а затем переходит к следующему адресу, где находит число 201.

Это команда RET (RETURN - возврат). Она дает процессору указание прекратить в этом месте программу в машинных кодах и вернуться туда, откуда она вызывалась, т.е. в нашем случае - в БЕЙСИК. Это самое процессор и сделал, о чем Вы получили сообщение БЕЙСИКа "О.К.".

Если все, что Вы здесь прочитали, Вам понятно, то Вы уже поняли, как составляются программы в машинных кодах. Можно, конечно, возразить, что пользы от такой программы не очень много, но сейчас не в этом суть. Важно, чтобы Вы поняли, что некая последовательность чисел может быть последовательностью команд для процессора Z-80.

К сожалению, для нас мало что говорит простая последовательность чисел вроде таких, как 0 и 201. Держать в памяти коды всех команд процессора (а их около семисот) непросто, но дело упрощается тем, что есть промежуточный язык между процессором и человеком - язык АССЕМБЛЕРа. У каждого кода есть своя мнемоника АССЕМБЛЕРа. Мнемоника - это набор букв, являющихся сокращением английских слов. Для нашего примера программа на АССЕМБЛЕРе выглядит так:

```
NOP
RET
```

Перевод этих мнемоник в машинные коды тоже можно поручить компьютеру. Для этого существуют специальные программы, которые тоже называют АССЕМБЛЕРами. Есть и противоположные программы - ДИСАССЕМБЛЕРы. Они наоборот переводят машинные коды в мнемоники АССЕМБЛЕРа.

И тех программ и других достаточно много. Часто они объединяются в пакеты. Широко распространены пакеты GENS3/MONS3 фирмы HISOFT и EDITAS/MONITOR 16/48 фирмы PICTURESQUE. Здесь

GENS3 и EDITAS - АССЕМБЛЕРы, а MONS3, MONITOR 16 и MONITOR 48 - ДИСАССЕМБЛЕРы.

Теперь давайте вернемся к нашей первой программе и попробуем ее несколько развить, чтобы она все же что-то делала. Процессор Z-80 имеет несколько регистров, у которых есть имена - "A", "B", "C" и т.д. Каждый из них может содержать одно какое-либо целое число от 0 до 255 (т.е. один байт).

Существуют десятки команд процессора, которые позволяют копировать содержимое регистров из одного в другой, а также выполнять связь с внешним миром, в т.ч. и с оперативной памятью.

Так, например, команда АССЕМБЛЕРа LD B,A (машинный код - 71) означает "загрузить содержимое регистра A в регистр B". LD - это сокращение от LOAD (LOAD - ЗАГРУЗКА).

Точно так же LD C,B (машинный код 72) означает "загрузить в регистр C содержимое регистра B". Можно загружать в регистры и целые числа. Например, LD A,N - означает "загрузить в регистр A целое число N, где N может быть числом от 0 до 255". До этого все команды были однобайтными. Эта же команда - двухбайтная. Сначала идет машинный код - 62, а за ним само число - N. Так, например, LD A,77 (загрузить в регистр A число 77) будет выглядеть так: 62,77. Здесь 62 - код операции, - он сообщает процессору, что надо сделать, а 77 - это операнд. Заметим здесь же, что бывают операции и трехбайтные и четырехбайтные. Первый байт, как правило, - код операции, а следующие за ним - операнды. Мы говорим "как правило" потому, что есть некоторые операции, код которых записывается двумя байтами /1.

Итак, мы уже готовы к тому, чтобы написать программу, которая будет перебрасывать какое-либо число из одного регистра процессора в другой.

/1 ИСТОРИЧЕСКАЯ СПРАВКА: Процессор Z-80 разрабатывался на основе своего предшественника, процессора 8080. Большинство команд у них совпадают, но для Z-80 было добавлено несколько десятков новых команд. Чтобы их отличить от команд 8080, они начинаются с префикса 203, 221, 237 или 253 (в шестнадцатиричном коде, соответственно: CB, DD, ED или FD). Поэтому появились операции, код которых состоит из двух байтов, за которыми следуют операнды.

АССЕМБЛЕР	МАШИННЫЙ КОД	
NOP	30000	0
LD A,77	30001	62
	30002	77
LD B,A	30003	71
LD C,B	30004	72
RET	30005	201

Гораздо интереснее было бы организовать обмен между процессором и оперативной памятью компьютера. Команд, которые позволяют это сделать, тоже очень много. Например, это команда LD (NN),A. Ее машинный код - 50. Она означает следующее: "Загрузить в ячейку памяти, адрес которой задан двухбайтным числом NN, содержимое регистра A". Эта команда - трехбайтная. Первым идет код операции (50), а за ним - двухбайтный операнд NN, который задает нужный Вам адрес. Например, если Вам нужен адрес 30008, то тогда операнд NN будет иметь вид: 56, 117 потому, что 117*256 + 56 = 30008.

Этот двухбайтный операнд имеет старшую часть и младшую. Запомните, что двухбайтные числа (как правило, это адреса) хранятся в памяти всегда в обратном порядке, т.е. сначала младший байт, а потом старший /1.

Таким образом, команда АССЕМБЛЕРа LD (30008),A в машинных кодах запишется так: 50, 56, 117.

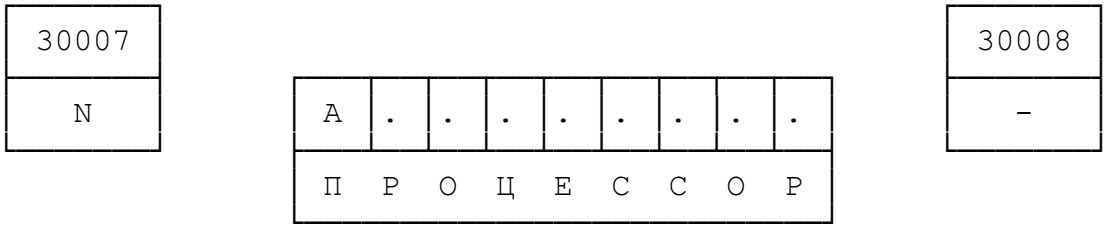
Существует и противоположная команда LD A,(NN) - "Загрузить в регистр A содержимое ячейки памяти, заданной адресом NN". Ее код - 58, за которым следуют два байта, указывающие на адрес: сначала младший, а потом - старший. Тогда LD A,(30007) запишется так: 58, 55, 117.

Теперь мы уже можем перемещать содержимое ячеек памяти из одной в другую, копируя его через регистр A процессора.

/1 ИСТОРИЧЕСКАЯ СПРАВКА: Наш житейский опыт учит, что сначала идут старшие разряды, а потом младшие. Например, в числе 567 сначала идут сотни (5), потом - десятки (6) и только затем единицы (7). В компьютере - наоборот. Это связано с тем, что первые компьютеры имели такую маленькую память, что о старших разрядах речь тогда и не шла. Они явились более поздним дополнением и им пришлось занять не первое место.

Теперь мы уже можем перемещать содержимое ячеек памяти из одной в другую, копируя его через регистр А процессора.

До операции



После операции

АССЕМБЛЕР	МАШИННЫЙ КОД	
LD A,(30007)	30000	58
	30001	55
	30002	117
LD (30008),A	30003	50
	30004	56
	30005	117
RET	30006	201

Можно еще несколько усложнить задачу и попытаться выполнить сложение содержимого двух каких-либо ячеек памяти и отправить результат на хранение в какую-либо ячейку. Для выполнения арифметических действий, например сложения, процессор Z-80 также имеет несколько команд. Рассмотрим команду ADD A,B (ее машинный код - 128). ADD = ADDITION = СЛОЖЕНИЕ.

ADD A,B означает следующее: "Прибавить содержимое регистра В процессора к содержимому регистра А и результат оставить в регистре А".

Как видите, эта команда однобайтная, т.к. все указания на

то, что откуда взять, что с ним сделать и куда отправить результат, в ней есть уже сами собой.

Предположим, что Вы хотите сложить содержимое ячеек 30013 и 30014, а результат поместить а ячейку 30015. Тогда программа на АССЕМБЛЕРе и в машинных кодах будет иметь вид:

АССЕМБЛЕР	МАШИННЫЙ КОД		КОММЕНТАРИЙ
LD A,(30013)	30000	58	Загрузить в регистр А
	30001	61	число, содержащееся в
	30002	117	адресе 30013.
LD B,A	30003	71	Загрузить в регистр В
			содержимое регистра А
LD A,(30014)	30004	58	Загрузить в решистр А
	30005	62	число, содержащееся в
	30006	117	адресе 30014.
ADD A,B	30007	128	Прибавить к содержимому А
			содержимое регистра В.
LD (30015),A	30008	50	Выгрузить содержимое
	30009	63	регистра А в адрес 30015.
	30010	117	
NOP	30011	0	Нет операции. Пауза.
RET	30012	201	Возврат туда, откуда эта
			программа вызывалась.

Попробуйте эту программу в работе. Пусть Вы хотите сложить два числа, скажем 50 и 70. Сначала выделим память для этой программы в машинных кодах.

```
10 CLEAR 29999
```

Теперь введем эту программу в память, начиная с адреса 30000.

```
20 FOR i=30000 TO 30012: READ q: POKE i,q: NEXT i
30 DATA 58, 61, 117, 71, 58, 62, 117, 128, 50, 63, 117, 0, 201
```

Запишем в ячейки 30013 и 30014 те два числа, которые мы желаем сложить:

```
40 POKE 30013, 50: POKE 30014, 70
```

Введем команду на исполнение нашей программы в машинных кодах.

```
50 RANDOMIZE USR 30000
```

И, наконец, стартуем нашу БЕЙСИК-программу - RUN.

Через долю секунды она отработает и на экране появится сообщение О.К. Вроде бы ничего не изменилось, но если Вы проверите, что находится в ячейке 30015, то убедитесь, что там находится результат сложения, т.е. число 120.

PRINT PEEK 30015 даст Вам: 120 О.К.

Итак, мы уже познакомились с несколькими командами процессора Z-80. Теперь нам придется прервать такое сверхпопулярное изложение, поскольку система команд этого процессора довольно обширна и при таком подходе пришлось бы отвести ей около тысячи страниц. Мы надеемся, что психологический барьер Вы уже преодолели и далее мы переходим к систематическому и постепенному освоению системы команд процессора.

Вы, по-видимому, поняли, что всякой команде в машинных кодах соответствует своя мнемоника АССЕМБЛЕРа, в которой в нескольких буквах зашифрована суть операции. К сожалению, обычно литературу пишут профессионалы для профессионалов и они не утруждают себя переводом мнемоник на русский язык, тем более, что по Z-80 книг на русском языке почти нет (есть несколько ведомственных переводов). Поэтому "Инфорком" в своем "Справочнике.." (часть 3) уделил место словарю для перевода мнемоник АССЕМБЛЕРа в нормальный английский, а через него и в русский язык.

В заключение этой главы несколько общих замечаний. Различные процессоры имеют различные системы команд. Знание системы команд процессора Z-80 еще не означает, что Вы сможете разбираться в машинном коде других компьютеров, собранных на других процессорах, но совершенно точно, что их освоение пройдет у Вас в десятки раз быстрее.

Даже для одного только процессора Z-80 мнемоники до сих

пор не стандартизированы. Многие ассемблирующие программы, например, EDITAS, ZEUS, GENS1...GENS3 и др. имеют отклонения в форме записи мнемоник, но они незначительны и всегда оговариваются в сопроводительной инструкции к программе.

2. ЗАЧЕМ ИЗУЧАТЬ ПРОГРАММИРОВАНИЕ В МАШИННЫХ КОДАХ

Законный вопрос - зачем изучать программирование в машинных кодах, когда существует столько разнообразных языков программирования, рассчитанных на любые вкусы и интересы? Разве это не возврат на тридцать пять лет назад, когда программировали только в кодах и языков программирования еще не существовало? Разве писать программу в машинном коде это не то же самое, что высекать дом в скале с помощью напильника? Чтобы ответить на эти вопросы, надо рассмотреть, что же такое языки программирования и как они обеспечивают общение человека с компьютером.

Процессор - это микросхема, которая не понимает никаких языков программирования, воспринимает только машинные коды, поскольку они представлены электрическими импульсами. Поэтому, когда мы запускаем программу в машинных кодах, то работаем с процессором напрямую, без каких-либо посредников, какими являются языки программирования. Для компьютера язык программирования сам по себе ничего не говорит. Ему нужна системная программа, которая прочитает операторы Вашей программы и переведет их (транслирует) в машинный код. Такие программы существуют, их называют трансляторами.

Трансляторы бывают двух типов - ИНТЕРПРЕТАТОРЫ и КОМПИЛЯТОРЫ.

Интерпретатор переводит Вашу программу с языка высокого уровня (например, БЕЙСИКа) в машинный код последовательно строку за строкой. Он работает примерно так: прочитал строку, проверил нет ли в ней ошибок, перевел ее в машинный код, выполнил команды машинного кода, запомнил где нужно результат и перешел к следующей строке. Чтобы сделать, например, операцию

```
10 PRINT  2 + 2,
```

интерпретатор обращается к процессору несколько сот раз. Вам этого не видно, все равно результат появится на экране через

доли секунды, но это так.

Если же Вам позже придется вернуться к этой строке (например, с помощью GO TO 10), то все эти действия будут повторены. А ведь многие операции выполняются в циклах.

Таким образом, интерпретатор работает крайне медленно. Зато имеется возможность работы в диалоговом режиме. Так, на Бейсике, когда Вы набираете программу, каждая строка сразу же и проверяется на правильность синтаксиса и, если Вы сделаете ошибку, то строка не будет введена в программу нажатием клавиши ENTER до тех пор, пока Вы эту ошибку не устраните. Вы всегда можете прервать работу программы, внести изменения и стартовать опять, причем с той строки, с какой хотите. Работать с интерпретатором БЕЙСИКа настолько удобно для начинающих, что на многих моделях персональных ЭВМ, в том числе и на "Спектруме", он уже "зашит" в постоянное запоминающее устройство (ПЗУ) и служит не только языком программирования, но и выполняет функции операционной системы компьютера. При включении компьютера в сеть он сразу готов к выполнению команд БЕЙСИКа.

В отличие от интерпретатора, компилятор переводит Вашу программу с языка высокого уровня (например ПАСКАЛя или ФОРТРАНа) в машинные коды всю целиком. После того, как программа написана, она компилируется в машинный код. Программа, написанная Вами на языке, называется исходным текстом (исходным модулем, исходным блоком, исходным файлом). То, что Вы получаете в результате компиляции, называется объектным кодом (модулем). Вы можете выгрузить объектный код на ленту, а потом снова загрузить. Можете запустить его на исполнение, но здесь у Вас уже нет возможности во время работы программы ее остановить, внести изменения и снова запустить с произвольно взятого места. Если такая необходимость возникает, надо заново загрузить исходный текст программы, внести изменения, а потом опять откомпилировать его в машинный код.

Поскольку здесь компиляция выполняется для каждой строки только один раз, а потом полученный машинный код можно использовать хоть всю жизнь, то здесь скорость работы программы гораздо выше и лишь немного уступает скорости программ, сразу написанных на АССЕМБЛЕРе. Те все же быстрее, т.к. как бы хорош компилятор ни был, он все же не в состоянии сделать объектный

код оптимальным по быстродействию и по объему занимаемой памяти.

К недостаткам компиляторов относится и то, что здесь процесс составления и отладки программ более трудоемкий и менее очевидный, т.к. между внесением изменений в программу и результатом запуска есть еще промежуточный процесс - компиляция. Кроме того, компиляторы часто накладывают ограничения на применение тех или иных операторов языка программирования.

Итак, программирование в машинном коде (или на АССЕМБЛЕРе, что то же самое) позволяет повысить скорость работы программы по сравнению с работой через интерпретатор в 50...200 раз и в 1,5...3 раза по сравнению с кодом, прошедшим компиляцию. Это бывает чрезвычайно важно, если в программе есть многочисленные вложеннные друг в друга циклы, если многократно выполняются поиск и выбор данных из обширных областей памяти. Много времени занимают операции, связанные с обработкой графических изображений на экране. Эффект плавного и быстрого перемещения (и изменения формы) объектов в компьютерных видеоиграх практически всегда создается программированием в машинном коде.

Второй важный момент состоит в сокращении объема занимаемой памяти. Казалось бы, что 48 килобайтов в "Спектруме" - это немало. Тем не менее, если учесть, что почти 7К занимает экранная область памяти, то при создании красочных программ, содержащих несколько десятков экранов, на их хранение даже и в компрессированном виде уходит несколько десятков К, а надо еще выделить место для хранения таблиц данных, переменных, текстов сообщений и т.п. Поэтому в лучших программах на сам машинный код, по которому они работают, остается всего 4...8К. Здесь и проявляется мастерство программиста, сумевшего обеспечить в столь малом объеме богатое многообразие вариаций игры, палитру цветов и гамму звуков.

Сравним расход памяти при работе на БЕЙСИКе и в машинных кодах. Программа на БЕЙСИКе размером в 30 строк занимает примерно 1К памяти.

Аналогичная ей, выполняющая те же задачи, программа в машинных кодах будет иметь примерно 150 строк (команд), но занимают они всего 200...250 байтов оперативной памяти.

В нашей стране есть еще две объективные причины, вызывающие повышенный интерес к программированию в машинных кодах.

Дело в том, что наибольшее число пользователей этого класса компьютеров у нас составляют радиолюбители и специалисты по электронике, самостоятельно собравшие компьютер. Обычно они не останавливаются на достигнутом, а развивают свое хобби дальше, ищут пути усовершенствования машины, пути подключения дополнительных устройств: интерфейсов принтера, дисковода, джойстика, светового пера, программатора, контроллеров бытовой аппаратуры, бытовых систем, систем управления различными моделями и т.д. вплоть до систем управления технологическими процессами промышленных предприятий. В конце 80-х годов в Душанбе на базе этого компьютера была сделана система голосования республиканского парламента. Работают под управлением "Спектрума" и очень интересные системы управления сельскохозяйственными предприятиями (фермами и птицефермами). Интересны автоматизированные системы диагностирования автомобиля, системы контроля состояния спортсменов и многое многое другое. Поскольку процедуры, управляющие работой всех этих устройств (их называют драйверами), обычно пишутся в машинных кодах, то их надо знать и уметь с ними работать.

Другая особенность состоит в том, что основная масса программ для Синклер-компьютеров написана в Англии на английском языке. Желание адаптировать эти программы на русский язык во многих случаях упирается в необходимость понимания структуры программы, а большинство лучших программ написано именно в машинных кодах.

"ИНФОРКОМ" получает множество писем с вопросами по поводу переделки системы загрузки блоков фирменных программ. Мы так понимаем, что многие уже обзавелись дисководом с Бета-диск интерфейсом, и теперь перед ними стоит задача переписывания программ на диск. При этом пользоваться "магической кнопкой" они не хотят, т.к. при этом любая, даже самая короткая программа будет занимать на диске 48К, а хотят ее переписать на диск блок за блоком и сопроводить загрузчиком с диска. На все эти вопросы ответ может быть только один. Поскольку разные фирмы в своих программах применяют разные системы загрузки, универсального решения здесь не существует. К каждой программе нужен индивиду-

альный подход. Надо прочитать загрузчик программы, понять из него куда какой блок загружается и в каком порядке они стартуют, а затем, если надо, внести в него свои изменения. Поскольку лучшие программы имеют при себе загрузчик в машинных кодах (обычно он следует после БЕЙСИК-загрузчика или организован внутри него в строке после оператора REM), то умение работать с машинным кодом Вам пригодится и здесь.

Вот в основном те причины, которые могут побудить Вас к освоению программирования в машиных кодах или хотя бы их пониманию (что достигается гораздо быстрее, чем способность активного программирования, но имеет не меньше значения), хотя хотелось бы отметить еще два важных, на наш взгляд, обстоятельства.

Во-первых, "Спектрум" имеет ПЗУ объемом 16К. Эта память буквально насыщена множеством очень полезных системных процедур. Все они записаны в машинных кодах. Их можно смело применять в собственных программах, обращаясь к ним по мере необходимости. Это дает колоссальный выигрыш в расходе памяти и вообще очень упрощает программирование. Поскольку все содержимое ПЗУ записано в машинном коде, умение разбираться в нем является необходимым. Для использования системных программ, содержащихся в ПЗУ, Вам необходимо ознакомиться с основами программирования в машинных кодах.

Во-вторых, изучение программирования в машинном коде процессора Z-80 хоть и трудоемкий, но очень полезный шаг для Вашего будущего. Компьютерная техника непрерывно прогрессирует. Широко внедряются IBM-совместимые машины с процессорами 8088, 8086, 80286, 80386, 80486. Процессор Z-80 - "двоюродный брат" процессора 8088 и имеет определенные черты сходства со всей этой серией. Те из Вас, кто свяжут свою судьбу с профессиональной вычислительной техникой, еще не раз вспомнят добром свои первые шаги в освоении машинного кода Z-80.

Что же касается особой трудоемкости работ по программированию в машинных кодах, то и здесь есть ряд возражений.

1. Нет необходимости сразу программировать. На первом этапе Вы уже сможете многого достичь, если будете просто разбираться в программах, а дальше все придет с набором опыта.

2. Существуют ассемблирующие программы, которые имеют достаточный набор средств, чтобы освободить Вас от самой рутинной

работы и снизить трудоемкость программирования.

3. Как правило, нет никакой необходимости всю программу писать в машинных кодах. Всегда в ней можно выделить блок, который решающим образом влияет на быстродействиве. Он может быть очень маленьким по размеру. Вот его-то и надо записать в маш. кодах, а остальную часть программы оставить, например, на БЕЙСИКе. Если Вы создаете программу "русско-английский словарь", то она вполне может быть написана на БЕЙСИКе и только процедура поиска перевода слова, занимающая много времени, должна быть записана в машинных кодах. Если же Вы создаете русско-китайский словарь, то еще одним узким местом станет рисование на экране иероглифов. Вам придется записать несколько процедур, которые смогут делать это быстро.

Диалог же с пользователем программа может вести и из БЕЙСИКа.

4. И, наконец, последнее. Ни один программист, работающий в машинных кодах, не пишет большую программу от начала и до конца с чистого листа. Программа представляет хитроумное сплетение больших и малых подпрограмм (процедур), из которых может быть до 80% стали для этого программиста стандартными, т.е. он применяет их регулярно во всех своих программах без особых перемен, а Вы никогда об этом и не догадаетесь. Это могут быть процедуры поиска, сортировки, арифметических и логических вычислений, обработки изображений, опроса внешних устройств (например, джойстика), вывода текста на экран, звуковых эффектов и т.д. и т.п.

Конечно, если Вы делаете только первые шаги в машинных кодах, то у Вас нет еще такой библиотеки, но прочитав эту книгу, Вы уже можете покопаться в машинном коде некрупных фирменных программ. Там Вы найдете множество открытий. В этом Вам очень поможет какая-либо дисассемблирующая программа, например MONITOR 16/48. Для Вас открыты и другие книги "ИНФОРКОМа" и, самое главное, наши выпуски "ZX-РЕВЮ".

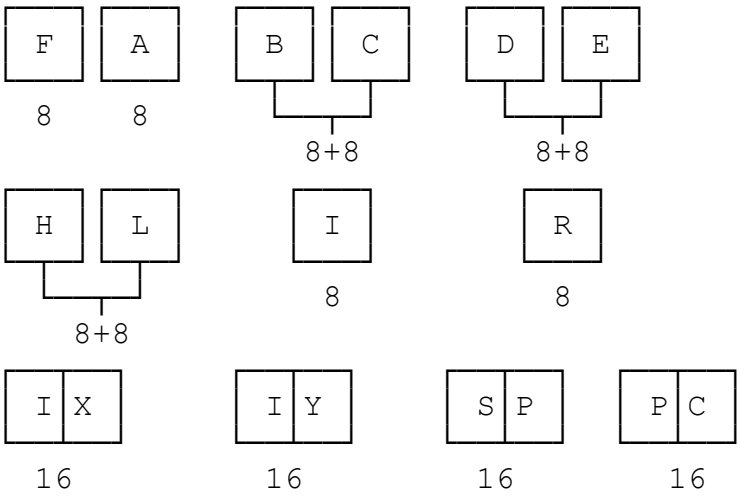
3. АРХИТЕКТУРА ПРОЦЕССОРА Z-80.

Процессор Z-80 был создан американской корпорацией ZILOG и, как оказалось, явился одним из самых удачных процессоров в мире. Он имеет 8-ми разрядную шину данных и 16-ти разрядную адресную шину. Это означает, что он может обрабатывать целые числа (байты) от 0 до 255 (т.к. 2^8-1 = 255) и напрямую обращаться к 65535 ячейкам памяти (т.к. 2^16-1=65535).

В структуру процессора входят несколько регистров. Именно с ними мы и работаем, программируя в машинном коде. Рассматривайте эти регистры как ячейки внутренней памяти процессора. Если из ячеек постоянной памяти компьютера (ПЗУ) можно только считывать информацию, а при работе с оперативной памятью (ОЗУ) информацию можно как записывать, так и считывать, то при работе с регистрами можно не только записывать в них информацию или считывать ее, но и выполнять еще большое количество различных арифметических и логических операций над содержимым регистров.

Ячейки оперативной памяти одинаковы между собой и различаются только адресом. Регистры процессора своеобразны и имеют свои имена. Они обозначаются буквами латинского алфавита:

F, A, B, C, D, E, H, L, SP, PC, IX, IY, I, R.



Это основной набор регистров. Есть еще один дополнительный набор, его называют альтернативным, его мы и рассмотрим ниже.

Некоторые из этих регистров - восьмиразрядные (однобайтные) и могут содержать целые числа от 0 до 255. Это регистры F, A, B, C, D, E, H, L, I, R.

Другие - шестнадцатиразрядные (двухбайтные). Они могут содержать целые числа от 0 до 65535. Это регистры SP, PC, IX, IY.

Некоторые однобайтные регистры могут объединяться в двухбайтные регистровые пары BC, DE, HL. В этом случае о такой паре можно говорить как об отдельном шестнадцатиразрядном регистре.

Процессор Z-80 является весьма асимметричным процессором, т.е. каждый регистр обладает специфическими чертами, делающими его не похожим на остальные, и различные регистры предназначены для выполнения различных функций.

Рассмотрим регистры основного набора.

Регистр A - АККУМУЛЯТОР.

Этот регистр называют аккумулятором, т.к. во многих случаях результат исполнения операции остается в нем. В этом регистре выполняется наибольшее количество арифметических и логических команд. Это основной однобайтный регистр процессора. Во многих командах даже не указывается, к какому регистру они относятся: предполагается, что они относятся к регистру A.

Регистры B, C, D, E, H, L - РЕГИСТРЫ ОБЩЕГО НАЗНАЧЕНИЯ.

Эти регистры довольно широко используются процессором при выполнении различных команд. Их характерная особенность состоит в том, что они могут использоваться и как однобайтные одиночные регистры и как двухбайтные шестнадцатиразрядные регистровые пары: BC, DE, HL.

В этом случае наибольшее количество команд имеет регистр HL. При работе с двухбайтными целыми числами он имеет такое же важное значение, что и регистр A при работе с однобайтными.

Все эти пары используются, как правило, для хранения адресов. При этом в регистре HL часто помещается адрес того байта, с которым производится операция.

К регистру DE подходит сокращение от слова DESTINATION

(МЕСТО НАЗНАЧЕНИЯ). И действительно, во многих операциях, связанных с перемещением чего-либо куда-либо, регистр HL указывает, откуда взять, а регистр DE - куда отправить.

К регистру BC можно подобрать сокращение BACK COUNTER (ОБРАТНЫЙ СЧЕТЧИК). В тех случаях, когда определенная последовательность команд должна повториться N раз, т.е. когда организуется цикл вычислений, в регистре BC хранится счетчик этого цикла.

При объединении одиночных регистров в пары, в первом хранится старший байт, а во втором - младший. Итак, B, D, H - старший, а C, E, L - младший. Для тех, кто немного знает английский язык, напоминанием будет название регистра HL. Сначала H - HIGH (СТАРШИЙ), а затем L - LOW (МЛАДШИЙ).

Регистр SP - УКАЗАТЕЛЬ СТЕКА (STACK POINTER).

В тех случаях, когда необходимо на некоторое время освободить какой-либо регистр (или регистры) и запомнить его содержимое впредь до дальнейшего использования, компьютер может использовать в качестве места временного хранения данных особый участок оперативной памяти, называемый машинным стеком. Обратите внимание на слово "машинный". Бывают и другие стеки, например стек калькулятора или, скажем, пользовательский стек, который Вы создадите сами для своих собственных нужд, но об этом речь еще впереди, а пока поговорим о машинном стеке. Стек - это такая форма организации памяти, при которой загрузка и выгрузка данных выполняется по принципу "последним пришел - первым уйдешь".

После включения компьютера в сеть, машинный стек в нем организуется автоматически. Его создают те процедуры, которые записаны в ПЗУ компьютера. Это и не удивительно, ведь он нужен им для работы, а работать им приходится очень много, хотя бы для того, чтобы компьютер сразу после включения уже мог понимать Ваши команды. Находится машинный стек в верхних областях памяти, выше адреса, установленного в системной переменной RAMTOP, чтобы написанная Вами БЕЙСИК-программа не затерла и не повредила его. Но если Вы работаете с машинным кодом, можно

организовать стек и в другом месте оперативной памяти, причем почти в любом /1.

Стек"растет" сверху вниз. Т.е. если вершина стека находится по адресу 60000, то следующее число, помещенное на стек, будет записано в ячейку 59999, а следующие за ним - в 59998. При этом перемещается и его вершина.

Вот здесь-то и нужен регистр SP. В нем всегда содержится адрес вершины стека. При помещении очередного числа на стек, регистр SP автоматически уменьшает свое значение, а при снятии числа со стека - увеличивает. Заслав свой адрес в указатель стека, Вы можете организовать стек в другом месте оперативной памяти.

В принципе, в качестве временного места хранения промежуточных результатов вычислений можно использовать обычные методы адресации типа LD (NN),A, но при хранении данных на стеке доступ к ним проще, понятнее, быстрее и экономичнее.

/1 Стек можно организовать даже в экранной области, и некоторые фирмы этим пользуются. Программа организована так, что картинка, которая загружается перед началом программы содержит стек. После загрузки основного блока (блоков) машинного кода со стека снимаются данные, которые обеспечивают нормальный старт программы. Если Вы удалили, заменили или изменили фирменную заставку, то программа работать не будет. Чтобы Вы визуально не видели на экране, что в картинке есть какие-то данные в виде точек и полос, их можно спрятать за черным цветом PAPER экрана. Такая защита есть, например, в программе "NETHEREARTH".

Другой хитроумный прием, используемый профессионалами, состоит во временной организации стека в той области памяти, куда Вы хотите быстро перебросить большое количество данных. Тогда вместо того, чтобы перетаскивать блоки памяти из одного места в другое, их просто отправляют на стек. Такая операция происходит чрезвычайно быстро и переброска картинки из памяти на экран, например, происходит в 2 раза быстрее, чем при использовании других, самых быстрых команд процессора (так, например, в программе "STARION" этим приемом обеспечена необычайная гладкость и плавность мультипликации.)

Еще одно назначение стека связано с использованием подпрограмм. Так, при изменении нормального порядка исполнения программы, (когда команды исполняются последовательно одна за другой) и при переходе на исполнение подпрограммы необходимо запомнить адрес возврата, чтобы потом продолжить вычисления с точки, находящейся после места вызова подпрограммы. Этот адрес (в виде двух байтов) и помещается на стек. По окончании исполнения подпрограммы по команде RET он снимается со стека и отправляется в программный счетчик PC, что и обеспечивает возврат.

Разумеется, если во время работы подпрограммы Вы что-то будете помещать на стек, то при возврате программа не найдет нужного адреса, т.к. изменится содержимое вершины стека. Значит, если при работе в подпрограмме Вам необходимо поместить сколько-то чисел на стек, то столько же надо оттуда снять до того, как встретится команда RET.

РЕГИСТР PC - ПРОГРАММНЫЙ СЧЕТЧИК (PROGRAM COUNTER).

Этот регистр также, как и регистр SP - шестнадцатиразрядный. Он служит для хранения адреса той операции, которая должна выполняться следующей. Когда процессор начинает выполнять эту операцию, программный счетчик получает приращение на единицу, если команда однобайтная, на два - если двухбайтная и т.д. Таким образом, по окончании этой операции процессор знает, какую операцию выполнять следующей. Так выдерживается последовательное исполнение программы.

Если в программе имеется переход в иное место (аналогично GO TO БЕЙСИКа) или вызов подпрограммы (аналогично GO SUB БЕЙСИКа), то именно программный счетчик и обеспечивает такой скачок. В него засылается адрес, в который должен быть выполнен переход и процессор теперь знает, откуда продолжать вычисления.

РЕГИСТР F - ФЛАГОВЫЙ РЕГИСТР.

Этот регистр отличается от всех остальных и играет весьма своеобразную роль. Его не рассматривают как отдельный регистр,

хранящий восьмиразрядное число, а скорее, как восемь отдельных флаговых битов. Каждый флаг может быть установлен или снят, т.е. соответствующий бит включен (равен 1) или выключен (равен 0). Ниже показана структура флагового регистра.

S	Z	-	H	-	O/P	N	C
7	6	5	4	3	2	1	0

Он содержит следующие флаги:

Бит 0 - флаг переноса C;

Бит 1 - флаг сложения/вычитания N;

Бит 3 - не используется;

Бит 4 - флаг полупереноса H;

Бит 5 - не используется;

Бит 6 - флаг нуля Z;

Бит 7 - флаг знака S.

Флаги C, P/O, Z, S - активно используются в различных командах. Мы рассмотрим работу с этими флагами в разделе 5.9.5.

Флаги N и H программистом не используются. Они участвуют при выполнении операций с десятиричными числами, выраженными двоичной формой. Об этом мы поговорим в следующей главе.

РЕГИСТРЫ IX, IY - ИНДЕКСНЫЕ РЕГИСТРЫ.

Это две регистровые пары, т.е. это шестнадцатиразрядные регистры. Используются они, как правило, для запоминания адресов. Характерной особенностью этих регистров является то, что благодаря им можно организовать так называемую индексную адресацию. Если Вам надо работать с какими-либо таблицами данных, то Вы можете при разыскании и выборке нужного Вам элемента таблицы (например 55-го) не задавать его адрес, а хранить в индексном регистре адрес начала таблицы. Обращение к 55-ому элементу выполняется путем обращения к этому индексному регистру с указанием величины "смещения" нужного Вам элемента от начала таблицы, т.е. в данном случае числа 55.

В "Спектруме" относительно регистра IY есть неофициальное соглашение о том, что он содержит адрес 23610, который является

адресом системной переменной ERR_NR (см. нашу разработку "БОЛЬШИЕ ВОЗМОЖНОСТИ ВАШЕГО СПЕКТРУМА"). Это позволяет легко обращаться к любой системной переменной. Первоначальную установку содержимого регистра IY на этот адрес выполняет после включения компьютера в сеть программа инициализации, находящаяся в ПЗУ, но Вы, конечно, можете изменить эту установку, заслав туда какой-либо свой адрес, если работаете в машинных кодах. Не забудьте восстановить исходное значение 23610, если Вы хотите, чтобы Ваша программа, исполнив машинный код, нормально вернулась в БЕЙСИК.

Регистр IX широко используется компьютером при исполнении команд загрузки/выгрузки программ: LOAD, SAVE, VERIFY, MERGE. В нем хранится адрес начала места загрузки (выгрузки) блока. Когда Вы выполняете загрузку/выгрузку из БЕЙСИКа, Вам об этом думать не надо. Процедуры, расположенные в ПЗУ, все за Вас сделают, но если Вы выполняете эти операции из программы в машинных кодах, то надо установить требуемое значение в этом индексном регистре.

РЕГИСТР R - РЕГИСТР РЕГЕНЕРАЦИИ.

Это внутренний системный регистр, который используется процессором автоматически, без участия программиста. Электрические заряды в микросхемах оперативной памяти имеют тенденцию "утекать", поэтому для нормальной работы компьютера необходимо производить подзарядку (регенерацию) оперативной памяти. В этих целях и используется регистр R.

РЕГИСТР I - ВЕКТОР ПРЕРЫВАНИЙ.

Для компьютеров иных систем, собранных на базе процессора Z-80, этот регистр используется для организации обмена с периферийными устройствами. В "Спектруме" эта возможность не используется и этот регистр участвует только в формировании телевизионного сигнала.

АЛЬТЕРНАТИВНЫЙ НАБОР РЕГИСТРОВ.

Одной из отличительных черт процессора Z-80 является наличие альтернативного набора регистров. Эти регистры F', A', B', C', D', E', H', L'. Они полностью идентичны регистрам F, A, B, C, D, E, H, L.

Наличие такого альтернативного набора позволяет быстро переключаться с одного набора на другой. Когда нужно произвести какие-то операции с регистрами общего назначения, а они уже заняты и потерять содержащиеся в них данные нельзя, перед Вами есть три пути:

1. Отправить содержимое в память, а потом вернуть его назад.

2. Поместить содержимое на стек, а потом вернуть его оттуда.

3. Переключиться на альтернативный набор регистров и работать с ним, а потом переключиться назад на основной набор.

Третий путь самый оперативный.

Благодаря наличию альтернативного набора процессор может одновременно обслуживать две задачи. В то время, когда Вы переключились на альтернативный набор, он становится основным, а бывший основным - становится альтернативным. Физически эти наборы совершенно идентичны и потому нет никаких программных средств, чтобы определить, с каким именно набором Вы в данный момент работаете. Программа может получаться довольно хитросплетенной и от программиста требуется определенная внимательность.

В "Спектруме" есть небольшая особенность по использованию альтернативных регистров. Она касается альтернативной пары H'L'. Дело в том, что для нормальной работы компьютера в БЕЙСИКе в этой паре должен стоять адрес 10070 (2756H в шестнадцатиричной системе). Поэтому при работе с альтернативным набором, если Вы рассчитываете впоследствии возвратиться в БЕЙСИК, Вам либо не надо трогать содержимое H'L', либо перед возвратом надо восстанавливать там число 10700. Если же Вы в БЕЙСИК возвращаться не собираетесь, делайте с этой парой все, что хотите.

4. ФОРМЫ ПРЕДСТАВЛЕНИЯ ЧИСЕЛ В ПРОЦЕССОРЕ Z-80.

4.1. Числовые системы.

Для того, чтобы освоить программирование в машинный кодах или на языке АССЕМБЛЕРа, необходимо знать, как организовано хранение информации в ячейках памяти и в регистрах процессора. Основной единицей информации является байт, который в свою очередь состоит из восьми битов. Бит может быть включен (равен 1) или выключен (равен 0), т.е. может принимать только два значения, а байт, соответственно, может принимать 256 (два в восьмой степени) значений и имеет диапазон от 0 до 255.

Те числа, с которыми мы имеем дело в повседневной практике, называются десятиричными. Они строятся из разрядов (единицы, десятки, сотни и т.д.), каждый из которых может быть выражен степенью числа 10.

Так, 567 = 5*10^2 + 6*10^1 + 7*10^0

Компьютеру работать в десятиричной системе очень неудобно. Поскольку информация в нем представлена электрическими зарядами и можно выделить только два устойчивых состояния - есть заряд / нет заряда, то самое удобное - это хранить числа в компьютере в двоичной системе. Например, обычное десятиричное число 156 может быть записано в двоичной форме как 1001 1100B. Здесь буква B на конце означает, что число записано в двоичной (BINARY) системе. Преобразовать его в десятиричную форму можно разложением по степеням числа 2 так, как мы это делали для числа 567 выше, раскладывая его по степеням числа 10.

```
10011100B = 1*2^7 + 0*2^6 + 0*2^5 + 1*2^4 + 1*2^3 + 1*2^2
          + 0*2^1 + 0*2^0 = 128 + 0*64 + 0*32 + 16 + 8 + 4 +
          0*2 + 0*1 = 128 + 16 + 8 + 4 = 156
```

Вы видите очевидные недостатки двоичной формы записи. Во-первых, это очень громоздкая (длинная) запись, а потому и очень утомительная. Во-вторых, она с трудом переводится в привычную нам десятиричную форму. Все это вызывало бы массу ошибок, если бы программисты, разрабатывая программу, применяли бы двоичное

представление целых чисел.

Гораздо удобнее в работе шестнадцатиричная система. Она имеет в качестве основания число 16, поэтому уже двумя разрядами можно выразить 256 целых чисел от 0 до 255 (т.к. 16^2 = 256).

Поскольку для шестнадцатиричной системы не хватает арабских цифр, то для выражения шестнадцатиричных цифр 10...15 приходится прибегать к буквенным обозначениям:

0 - 0	1 - 1	2 - 2	3 - 3
4 - 4	5 - 5	6 - 6	7 - 7
8 - 8	9 - 9	10 - A	11 - B
12 - C	13 - D	14 - E	15 - F

То же самое число 156 в шестнадцатиричной форме запишется как 9CH. Здесь буква H в конце числа говорит о том, что оно записано в шестнадцатиричной (HEXADECIMAL) системе.

9C = 9*16^1 + 12*16^0 = 144 + 12 = 156

Перед начинающим программистом встает вопрос, в какой системе работать - десятиричной или шестнадцатиричной? За десятиричную есть только один довод - многолетняя привычка. Шестнадцатиричная система удобна той простотой, с которой двоичная форма в нее переводится, а ведь двоичная форма отражает физическую суть операции, что делает шестнадцатиричную систему основной для программистов-профессионалов, а также для специалистов в электронике и схемотехнике. Простота перевода из двоичной системы в шестнадцатиричную и наоборот обеспечивается вследствие того, что число шестнадцать может быть выражено четвертой степенью числа два: 2^4 = 16. Поэтому, чтобы перевести число из двоичной формы в шестнадцатиричную, можно перевод выполнять по каждому полубайту (по каждым четырем битам отдельно):

1001 1100 B = 9C H /1

/1 Вы, очевидно обратили внимание на то, что после числа, записанного в шестнадцатиричной системе ставят букву "H". Иногда для этой цели используют символ "хэш" ("#"). Его еще называют "решеткой". Тогда запись выглядит так: #9C.

Для обозначения десятиричного числа ставят символ "D", либо ничего не ставят вообще: 156 = 156D = 9CH = #9C.

Таблица для связи между системами приведена ниже.

Выбирая, в какой системе Вам в дальнейшем работать, надо также учесть, что большинство прикладных программ, поддерживающих программирование в машинных кодах - АССЕМБЛЕРов и ДИСАССЕМБЛЕРов, работают в шестнадцатиричной системе.

Связь между системами.

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

4.2 Двоичная дополнительная форма записи.

Приведенная выше двоичная форма позволяет работать с целыми положительными числами от 0 до 255. Такая двоичная форма называется АБСОЛЮТНОЙ, а операции с такими числами - АБСОЛЮТНОЙ ДВОИЧНОЙ АРИФМЕТИКОЙ. Рассмотрим, например, что происходит с содержимым аккумулятора, если его постоянно увеличивать на единицу.

Флаг С регистра F	Регистр А	Шестнадц.	Десятирич.
0	0000 0000	00	00
0	0000 0001	01	01
. . . . .			
0	1001 1100	9C	156
. . . . .			
0	1111 1111	FF	255
1	0000 0000	00	00
. . . . .	и т.д.	. . . . .	

В то же время, есть операции, в которых необходимо наличие и целых отрицательных чисел. Например, это операции перехода (аналог GO TO). Переход может быть как вперед на N шагов (N - положительное), так и назад (N - отрицательное).

Когда процессор встречает такую команду, он воспринимает следующий за кодом операции операнд N не как записанный в абсолютной двоичной форме, а как записанный в ДОПОЛНИТЕЛЬНОЙ ДВОИЧНОЙ ФОРМЕ. В этой форме могут записываться целые числа от 0 до 127 и от -128 до -1. Таким образом, она может служить для записи целых чисел СО ЗНАКОМ.

Рассмотрим наш предыдущий пример с постепенным наращиванием содержимого аккумулятора, но для двоичной дополнительной арифметики.

Флаг С регистра F	Регистр А	Шестнадц.	Десятирич.
0	0000 0000	00	00
0	0000 0001	01	01
. . . . .			
0	0111 1111	7F	127
0	1000 0000	80	-128
0	1000 0000	81	-127
. . . . .			
0	1111 1111	FF	-1
(1)	0000 0000	00	0
. . . . . и т.д.			

Запомните простое правило двоичной дополнительной арифметики: чтобы поменять знак числа, надо все его единицы заменить на нули, а все нули - на единицы и к результату прибавить 1.

+ 5 - это 0000 0101B (05H)

Тогда -5 - это 1111 1010B + 1 = 1111 1011B (FBH)

Сложение и вычитание в этой системе выполняются как обычно (поразрядно), но перенос при переполнении старшего разряда игнорируется. Так +5 + (-5) = 0.

```
Флаг переноса          0000 0101
регистра F            +1111 1010
   (1)                 0000 0000
игнорируется
```

Ниже показаны значения каждого бита восьмиразрядного числа в абсолютной и в дополнительной двоичных формах.

	7	6	5	4	3	2	1	0
Абсол.двоичн.форма	128	64	32	16	8	4	2	1
Дополнит.двоич.форма	-128	64	32	16	8	4	2	1

4.3 Десятиричная арифметика в двоичном выражении.

Это особый вид представления целых чисел в регистрах процессора. Он называется BCD-арифметикой. BCD - BINARY CODED DECIMAL (ДВОИЧНЫЙ КОД ДЕСЯТИРИЧНЫХ ЧИСЕЛ).

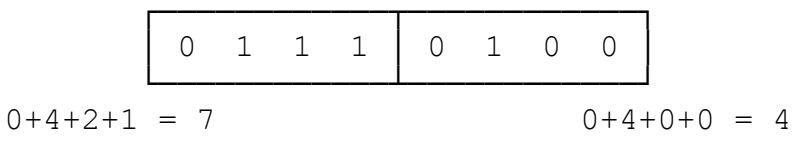
Эта форма основана на том, что каждый разряд десятиричного числа можно представить в виде четырех двоичных битов:

0 - 0000 1 - 0001 2 - 0010 3 - 0011
4 - 0100 5 - 0101 6 - 0110 7 - 0111
8 - 1000 9 - 1001

Значения от 1010 до 1111 - не используются.

Четырехбитную группу называют полубайтом и, таким образом, один байт в этой форме может содержать двухразрядное десятиричное число от 0 до 99.

На рисунке показано представление в этой форме десятиричного числа 74.



Как Вы видите, один полубайт в BCD-арифметике может содержать число от 0 до 9, в то время как в абсолютной двоичной ариметике - от 0 до 15. Очевидно, что BCD-арифметика является довольно расточительной, но у нее есть свои преимущества.

Для двух чисел, записанных в BCD-ариметике, обычные прин-

ципы сложения и вычитания неприменимы. Это происходит потому, что в абсолютной двоичной арифметике полубайт является заполненным, когда он равен 1111 и тогда выполняется переход к более старшему разряду. В BCD-арифметике полубайт заполнен, когда он равен 1001 и уже здесь выполняется переход к более старшему разряду.

```
Абс. двоичная арифметика        BCD-ариметика
      1111                          1001
 +       1                     +      1
     -----                         -----
  1   0000                       1 0000
```

В наборе команд процессора есть всего только три команды, которые работают с числами, представленными в этой форме, но они достаточно часто встречаются, поскольку их применение значительно упрощает преобразование чисел перед выдачей их на экран в десятиричной форме.

В предыдущей главе мы рассматривали флаги регистра F процессора и указали на то, что флаги N и H программистом не используются. Они участвуют в операциях с BCD-арифметикой.

Флаг N - флаг сложения/вычитания. Он равен 1 для всех операций вычитания и равен 0 для всех операций сложения.

Флаг H - флаг полупереноса. Он включается при переполнении младшего полубайта, когда начинается заполнение старшего полубайта.

5. СИСТЕМА КОМАНД ПРОЦЕССОРА.

Система команд процессора Z-80 довольно обширна и потому наилучший способ ее изучения состоит в том, чтобы сгруппировать похожие по своему действию команды и осваивать их по группам. Мы выделим 19 основных групп команд, но некоторые из этих групп будут иметь подгруппы.

Начиная освоение машинных кодов в главе 1 мы использовали десятиричную систему для записи кодов операций, операндов и адресов. Это делалось для более быстрой адаптации читателей к особенностям программирования на уровне процессора. Теперь же,

когда мы установили, что профессионально эффективнее работать в шестнадцатиричной системе, мы будем применять ее. Указывать суффикс "H" в записи чисел и адресов уже не будем.

5.1. ЗАГРУЗКА ЧИСЛА В РЕГИСТР

5.1.1. Загрузка одиночного регистра.

Эти команды существуют для всех одиночных регистров. Команды - двухбайтные. Сначала следует код операции, за ним - само число N. При засылке числа в регистр, старое значение, находившееся в нем, утрачивается.

Мнемоника	Код	Мнемоника	Код
LD A,N	3E N	LD B,N	06 N
LD C,N	0E N	LD D,N	16 N
LD E,N	1E N	LD H,N	26 N
LD L,N	2E N		

5.1.2. Загрузка регистровой пары.

В регистровую пару может быть загружено двухбайтное число, т.е. от 0 до 65535. Эти команды для всех регистров, кроме IX и IY - трехбайтные. За кодом операции следуют два байта, задающие число. Для операций с индексными регистрами IX и IY эти команды - четырехбайтные, т.к. код операции включает в себя префикс DD (для регистра IX) или FD (для регистра IY).

Наиболее часто двухбайтное число, засылаемое в регистровую пару, является адресом, но это может быть и просто целое число. Следует помнить, что всегда первый байт числа загружается в младший регистр пары, а второй - в старший.

Мнемоника	Код	Мнемоника	Код
LD HL,NN	21 N N	LD BC,NN	01 N N
LD DE,NN	11 N N	LD IX,NN	DD 21 N N
LD IY,NN	FD 21 N N	LD SP,NN	31 N N

5.2. КОПИРОВАНИЕ И ОБМЕН СОДЕРЖИМОГО РЕГИСТРОВ

5.2.1. Копирование одиночных регистров.

Для аккумулятора и шести регистров общего назначения существуют 49 команд, позволяющих копировать содержимое одного регистра в другой. Ни одна из этих команд на флаги не влияет.

Все эти команды - однобайтные. Коды этих команд приведены в таблице ниже.

	LD A,r	LD H,r	LD L,r	LD B,r	LD C,r	LD D,r	LD E,r
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

Для регистров I и R существуют команды копирования только в паре с аккумулятором.

Мнемоника	Код	Мнемоника	Код
LD A,I	ED 57	LD A,R	ED 5F
LD I,A	ED 47	LD R,A	ED 4F

5.2.2. Копирование содержимого регистровых пар.

Эти команды работают только с указателем стека. В этой группе только три команды. Они также не изменяют содержимого флагов.

Мнемоника	Код	Мнемоника	Код
LD SP,HL	F9	LD SP,IX	DD F9
LD SP,IY	FD F9		

В тех случаях, когда Вам надо скопировать содержимое произвольной регистровой пары в другую произвольную, например из BC в DE, Вы можете воспользоваться двумя командами копирования одиночных регистров - LD D,B и LD E,C.

Второй способ для этой цели - произвести переброску через стек, но для этого надо воспользоваться командами для работы со стеком см. раздел 5.11.

5.2.3. Команды обмена.

В этой подгруппе три команды:

EX DE, HL (код ED) - обмен между собой содержимым регистров HL и DE. На флаги не влияет.

EXX (код D9) - переключить основной набор регистров на альтернативный.

EX AF,A'F' (код 08)- переключить аккумулятор и флаговый регистры на альтернативные.

5.3. ЗАГРУЗКА РЕГИСТРОВ ИЗ ПАМЯТИ

Эти команды требуют указания адреса, откуда данные могут быть загружены в регистр. Существуют три основных метода адресации:

- прямая адресация;
- косвенная адресация;
- индексная адресация.

В соответствии с этими тремя методами мы и рассмотрим три подгруппы в этой группе.

5.3.1. Загрузка регистра прямой адресацией.

После кода команды следует двухбайтный ядрес ячейки NN, содержимое которой загружается в данный регистр. Единственный случай, когда такой командой загружается одиночный регистр - это загрузка аккумулятора, во всех остальных случаях загружаются два байта в регистровую пару. Предполагается, что второй байт берется из ячейки NN+1. При этом содержимое указанного

адреса идет в младший регистр, а следующего - в старший. Интересно отметить, что загрузка пары HL может выполняться двумя способами.

Мнемоника	Код	Мнемоника	Код
LD A,(NN)	3A N N	LD SP,(NN)	ED 7B N N
LD HL,(NN)	2A N N	вторая форма -	ED 6B N N
LD BC,(NN)	ED 4B N N	LD DE,(NN)	ED 5B N N
LD IX,(NN)	DD 2A N N	LD IY,(NN)	FD 2A N N

5.3.2. Загрузка регистра косвенной адресацией.

Здесь не указывается, из какого адреса надо взять данные, зато указывается, в какой регистровой паре находится этот адрес. Еще раз напомним, что загружается не этот адрес, а то, что в нем находится.

Пример: LD A,(HL) - код 7E - загрузить в регистр A то число, которое находится в ячейке памяти, адрес которой находится в регистре HL.

Прочие команды этой подгруппы.

Мнемоника	Код	Мнемоника	Код
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

5.3.3. Загрузка регистра индексной адресацией.

Этим методом из памяти могут загружаться однобайтные данные в аккумулятор или в регистры общего назначения. Адрес требуемого байта задается следующим образом:

- в регистре IX или IY находится базовый адрес таблицы или массива данных;

- в команде после кода операции указывается операнд "смещение" S требуемого адреса относительно базового. Величина S может быть только однобайтной от 0 до 255.

Обратите внимание на возможную ошибку: если Вам надо загрузить в регистр общего назначения тот байт, который находится по базовому адресу таблицы, хранящемуся в IX или в IY, то смещение ВСЕ РАВНО УКАЗЫВАТЬ НАДО. Просто в этом случае S=0.

Мнемоника	Код	Мнемоника	Код
LD A,(IX+S)	DD 7E S	LD A,(IY+S)	FD 7E S
LD H,(IX+S)	DD 66 S	LD H,(IY+S)	FD 66 S
LD L,(IX+S)	DD 6E S	LD L,(IY+S)	FD 6E S
LD B,(IX+S)	DD 46 S	LD B,(IY+S)	FD 46 S
LD C,(IX+S)	DD 4E S	LD C,(IY+S)	FD 4E S
LD D,(IX+S)	DD 56 S	LD D,(IY+S)	FD 56 S
LD E,(IX+S)	DD 5E S	LD E,(IY+S)	FD 5E S

Ни одна из команд загрузки регистров из памяти не меняет флаги. В Справочнике (часть 3) указано время исполнения команд в тактовых циклах. При частоте 3,5 МГц один тактовый импульс длится 0,000000286 сек. Минимальная продолжительность команды в процессоре Z-80 - 4 тактовых цикла. Из рассмотренных в данной группе команд самыми быстрыми являются команды загрузки регистров косвенной адресацией. Они занимают в основном 7 циклов. Команды загрузки прямой адресацией длятся 16 - 20 циклов, а при индексной адресации - 19 циклов.

5.4. КОМАНДЫ ЗАПИСИ ДАННЫХ ИЗ РЕГИСТРОВ В ПАМЯТЬ

Команды этой группы прямо противоположны командам предыдущей группы. Также как и там, здесь можно выделить три подгруппы в зависимости от метода адресации.

5.4.1. Запись в память при прямой адресации.

Мнемоника	Код	Мнемоника	Код
LD (NN),A	32 N N	LD (NN), SP	ED 73 N N
LD (NN),HL	22 N N	вторая форма -	ED 63 N N
LD (NN),BC	ED 43 N N	LD (NN), DE	ED 53 N N
LD (NN),IX	DD 22 N N	LD (NN), IY	FD 22 N N

Обратите внимание на то, что в этой подгруппе нет команд

на запись в память целых чисел. Когда необходимо выполнить такую операцию, ее делают поэтапно. Сначала число заносят в регистр командой 1-ой группы, а затем отправляют из регистра в память командой 4-й группы.

Поскольку эти команды являются противоположными командами 3-й группы, они часто употребляются совместно. Командами 4-й группы программные переменные отправляются на временное хранение в отведенное им в памяти место, а командами 3-й группы - вызываются из памяти в процессор для обработки.

5.4.2. Запись в память при косвенной адресации.

Мнемоника	Код	Мнемоника	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),N	36 N

5.4.3. Запись в память при индексной адесации.

Команды этой подгруппы существуют для всех одиночных регистров общего назначения и аккумулятора. Базовый адрес может находиться как в регистровой паре IX, так и в IY.

Мнемоника	Код	Мнемоника	Код
LD (IX+S),A	DD 77 S	LD (IY+S),A	FD 77 S
LD (IX+S),H	DD 74 S	LD (IY+S),H	FD 74 S
LD (IX+S),L	DD 75 S	LD (IY+S),L	FD 75 S
LD (IX+S),B	DD 70 S	LD (IY+S),B	FD 70 S
LD (IX+S),C	DD 71 S	LD (IY+S),C	FD 71 S
LD (IX+S),D	DD 72 S	LD (IY+S),D	FD 72 S
LD (IX+S),E	DD 73 S	LD (IY+S),E	FD 73 S
LD (IX+S),N	DD 36 S N	LD (IY+S),N	FD 36 S N

5.5. КОМАНДЫ СЛОЖЕНИЯ

Назначение команд этой группы вытекает из названия - прибавить число или содержимое регистра (пары) или содержимое индексированного адреса к содержимому регистра (пары). Это первая из групп арифметических команд, с которыми мы еще столкнемся. Сразу отметим, что команды этой группы работают в абсолютной двоичной арифметике.

5.5.1. Простое сложение.

Все команды этой подгруппы влияют на флаг переноса (C-флаг). В зависимости от того, превосходит ли результат сложения FF (255) для однобайтногшо сложения или FFFF (65535) для двухбайтного сложения содержимого двух регистровых пар, C-флаг либо включается, либо выключается.

Например: 2D + C0 = ED (45+192=237) -> C=0;
5D + C0 = 1D (93+192=29) -> C=1.

Однобайтное сложение.

Мнемоника	Код	Мнемоника	Код
ADD A,N	C6 N	ADD A,A	87
ADD A,H	84	ADD A,L	85
ADD A,B	80	ADD A,C	81
ADD A,D	82	ADD A,E	83
ADD A,(HL)	86	ADD A,(IX+S)	DD 86 S
ADD A,(IY+S)	FD 86 S		

Двухбайтное сложение.

Мнемоника	Код	Мнемоника	Код
ADD HL,HL	29	ADD HL,BC	09
ADD HL,DE	19	ADD HL,SP	39
ADD IX,IX	DD 29	ADD IY,IY	FD 29
ADD IX,BC	DD 09	ADD IY,BC	FD 09
ADD IX,DE	DD 19	ADD IY,DE	FD 19
ADD IX,SP	DD 39	ADD IY,DE	FD 39

5.5.2. Приращение.

Приращение (инкремент) - это увеличение содержимого регистра или ячейки памяти на единицу. В принципе это частный случай сложения, но он настолько широко применяется, что для этих целей была создана особая группа команд. Они нужны в первую очередь для организации разного рода счетчиков, когда при всяком проходе этой операции счетчик получает приращение на единицу.

Поскольку назначение команд "инкремент" состоит не в выполнении операций сложения, а в организации счетчиков, они не влияют на флаг переноса.

Мнемоника	Код	Мнемоника	Код
INC A	3C	INC H	24
INC L	2C	INC B	04
INC C	0C	INC D	14
INC E	1C	INC (HL)	34
INC (IX+S)	DD 34 S	INC (IY+S)	FD 34 S
INC HL	23	INC BC	03
INC DE	13	INC SP	33
INC IX	DD 23	INC IY	FD 23

Обратите внимание на разницу команд INC HL и INC (HL).

INC HL - прямая команда для регистра HL, она гласит "увеличить на единицу содержимое регистровой пары HL"-

INC (HL) - пример применения косвенной адресации; эта команда означает "увеличить на единицу содержимое ячейки памяти, адрес которой находится в регистре HL".

5.5.3. Сложение с учетом переноса.

Команды этой подгруппы имеют мнемонику ADC. Она отличается тем, что к результату сложения двух чисел еще прибавляется единица, если флаг переноса перед началом операции был установлен.

После окончания операции флаг переноса устанавливается в соответствии с результатом (есть перенос - включен, нет - выключен).

Применяются они при сложении многобайтных чисел. Так, на-

пример, при сложении двухбайтных чисел 035D и A0C0:

```
    03 5D
+   A0 C0
  -------
    A4 1D
```

возникает ситуация, когда сумма младших байтов дает число, большее 255 (FF). В этот момент и устанавливается флаг переноса. При сложении же старших байтов эта единица будет к ним прибавлена, если сложение производится командой ADC.

Мнемоника	Код	Мнемоника	Код
ADC A,N	CE N	ADC A,A	8F
ADC A,H	8C	ADC A,L	8D
ADC A,B	88	ADC A,C	89
ADC A,D	8A	ADC A,E	8B
ADC A,(HL)	8E	ADC A,(IX+S)	DD 8E S
ADC HL,BC	FD 8E S	ADC HL,HL	ED 6A
ADC HL,BC	ED 4A	ADC HL,DE	ED 5A
ADC HL,SP	ED 7A		

5.6. КОМАНДЫ ВЫЧИТАНИЯ

Команды этой группы прямо противоположны командам предыдущей группы. Они также работают в абсолютной двоичной арифметике. Их тоже можно разбить на три подгруппы.

5.6.1. Простое вычитание.

Команды этой подгруппы начинаются с мнемоники SUB и влияют на флаг переноса, но здесь в отличие от команд ADD флаг переноса устанавливается не когда байт переполняется, а когда при вычитании производится заем, т.е. когда вычитаемое больше уменьшаемого.

Мнемоника	Код	Мнемоника	Код
SUB N	D6 N	SUB A	97
SUB H	94	SUB L	95
SUB B	90	SUB C	91
SUB D	92	SUB E	93
SUB (HL)	96	SUB (IX+S)	DD 96 S
SUB (IY+S)	FD 96 S		

Обратите внимание на то, что действие всех этих команд относится к регистру A (аккумулятору), поэтому в мнемониках на него нет указаний, это принимается по умолчанию.

Так, SUB (HL) означает SUB A,(HL) - "вычесть из содержимого аккумулятора то число, которое находитсяв ячейке памяти, адрес которой находится в регистровой паре HL".

5.6.2. Уменьшение (декремент).

Декремент - уменьшение содержимого регистра или ячейки памяти на единицу. Операция прямо противоположна инкременту. Также, как и последняя, она служит не столько для вычитания, сколько для организации каких-либо счетчиков, а потому она тоже не влияет на флаг переноса.

Мнемоника	Код	Мнемоника	Код
DEC A	3D	DEC H	25
DEC L	2D	DEC B	05
DEC C	0D	DEC D	15
DEC E	1D	DEC (HL)	35
DEC (IX+S)	DD 35 S	DEC (IY+S)	FD 35 S
DEC HL	2B	DEC BC	0B
DEC DE	1B	DEC SP	3B
DEC IX	DD 2B	DEC IY	FD 2B

5.6.3. Вычитание с учетом "займа".

Команды этой подгруппы начинаются с мнемоник SBC. По своему смыслу они противоположны командам ADC. Отличие от команд SUB состоит в том, что если флаг переноса включен, то значит при вычитании младщего разряда был сделан заем из старшего и поэтому при операции со старшим разрядом надо еще вычесть единицу.

Эти команды применяются при вычитании многобайтных (двухбайтных и более) чисел.

Мнемоника	Код	Мнемоника	Код
SBC A,N	DE N	SBC A,A	9F
SBC A,H	9C	SBC A,L	9D
SBC A,B	98	SBC A,C	99
SBC A,D	9A	SBC A,E	9B
SBC A,(HL)	9E	SBC A,(IX+S)	DD 9E S
SBC (IY+S)	FD 9E S	SBC HL,HL	ED 62
SBC HL,BC	ED 42	SBC HL,DE	ED 52
SBC HL,SP	ED 72		

5.7. КОМАНДЫ СРАВНЕНИЯ

Мнемоники команд этой группы начинаются с CP от слова COMPARE (сравнивать).

Применяются эти команды чрезвычайно часто. С их помощью можно сравнивать содержимое аккумулятора с числом, с содержимым регистра или с байтом оперативной памяти и по результатам сравнения предпринимать те или иные действия. Поскольку эти команды однозначно относятся к регистру A, то в мнемониках он не указывается, это подразумевается по умолчанию.

В результате операции сравнения значение регистра A не изменяется. Результат операции сказывается только на флагах регистра F.

Если содержимое аккумулятора больше или равно сравниваемому числу, то флаг переноса C выключен (равен 0). Если же оно меньше - включен (равен 1).

Мнемоника	Код	Мнемоника	Код
CP A,N	FE N	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+S)	DD BE S
CP (IY+S)	FD BE S		

5.8. КОМАНДЫ ЛОГИКИ

Система команд процессора Z-80 позволяет выполнять три ви-

да логических операций - логическое "И" (AND), логическое "ИЛИ" (OR) и "ИСКЛЮЧАЮЩЕЕ ИЛИ" (XOR). В соответствии с ними мы можем выделить три подгруппы в этой группе команд.

Эти операции похожи на команды сравнения. В них также участвуют два операнда, один из которых находится в аккумуляторе.

5.8.1. Команды "И" (AND).

Данная команда двоичной логики дает результат, равный одному из операндов, если оба операнда равны, а если нет, то нули. Таким образом, результат может быть равен единице только если оба операнда равны единице.

```
Пример: Первый операнд  1010 1010    (AA)
        Второй операнд  1100 0000    (CO)
        --------------------------------
        Результат AND   1000 0000    (B0)
```

Как видите, операция производится побитно, т.е. нулевой бит первого числа сравнивается с нулевым битом второго, первый с первым, второй со вторым и т.д.

В этой подгруппе существуют 11 операций:

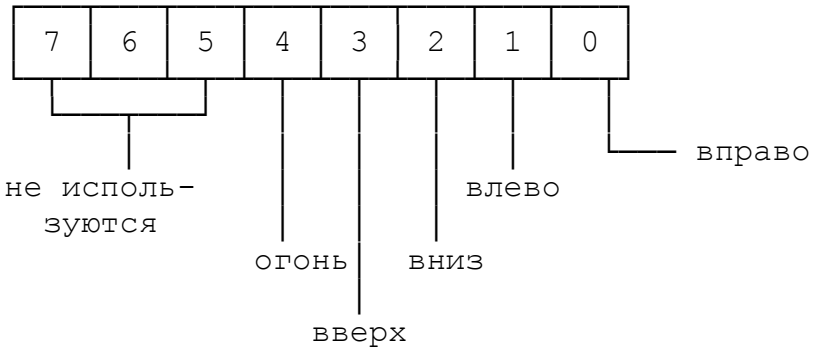
Мнемоника	Код	Мнемоника	Код
AND N	E6 N	AND A	A7
AND H	A4	AND L	A5
AND B	A0	AND C	A1
AND D	A2	AND E	A3
AND (HL)	A6	AND (IX+S)	DD A6 S
AND (IY+S)	FD A6 S		

С помощью команды AND можно довольно эффектно сбрасывать биты аккумулятора. Например, если Вам нужно его обнулить, дайте команду:

```
AND 0 (E6 0)
```

В программах очень часто бывает нужно проверить состояние некоторых битов, при этом остальные желательно занулить. Например, если в игровой программе управление ведется от Кемпстон-джойстика, то сигнал от кнопки "огонь" подается по 4-му биту порта 31.

ПОРТ 31 (1F)



Можно было бы проверить, не включена ли эта кнопка командой CP 10 (десятиричное 16), но из этого ничего не получится, т.к. часто кнопка "ОГОНЬ" включается одновременно с движением объекта. Например, если объект движется вправо-вверх и при этом "стреляет", то включены биты 1,2 и 4 и код равен 2 + 4 + 10 = 16. Такая "проверка" показала бы, что кнопка якобы не включена. Поэтому перед проверкой все остальные биты надо погасить. Это легко сделает команда AND 10, а потом можно давать и CP 10. Процесс гашения ненужных битов называется маскированием.

5.8.2. Команда "ИЛИ" (OR).

Эта команда также исполняется побитным сравнением двух двоичных чисел. Ее результат равен единице, если данный бит включен в первом или во втором операнде или в обоих вместе. Таким образом, в результате может быть 0 только если в обоих операндах 0, а в противном случае - 1.

```
Пример: Первый операнд    1010 1010    (AA)
        Второй операнд    1100 0000    (C0)
        ----------------------------------
        Результат OR      1110 1010    (EA)
```

Эта команда позволяет Вам столь же эффективно включать нужные биты в аккумуляторе (или проверять их включение), как команда AND позволяла их выключать. Например, если Вам надо,

чтобы в аккумуляторе были гарантированно включены биты 5, 3, 2 - дайте команду OR 2C (20 + 8 + 4 = 2C).

Мнемоника	Код	Мнемоника	Код
OR N	F6 N	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+S)	DD B6 S
OR (IY+S)	FD B6 S		

Интересна команда OR A. В результате ее действия производится сравнение по "ИЛИ" содержимого аккумулятора с ним же самим. Разумеется, в результате в аккумуляторе ничего не изменится, но зато гарантированно будет сброшен флаг переноса (флаг C регистра F). Такой же результат дает и применение команды AND A. Обе эти команды очень часто используются в программах именно для этой цели.

5.8.3. Команды "ИСКЛЮЧАЮЩЕЕ ИЛИ" (XOR).

В отличие от "ИЛИ" здесь результат равен 1, если хотя бы один из операндов равен 1, но не оба вместе. В остальных случаях он равен нулю.

```
Пример: Первый операнд   1010 1010  (AA)
        Второй операнд   1100 0000  (C0)
        --------------------------------
        Результат XOR    0110 1010  (6A)
```

Команда XOR A часто используется для обнуления аккумулятора. Это же можно сделать и командой LD A,0, но XOR короче (один байт вместо двух), а кроме того при команде XOR, как и при всех командах логики AND, OR, XOR сбрасывается лаг переноса, а команды группы загрузки LD на флаги влияния не оказывают.

Мнемоника	Код	Мнемоника	Код
XOR N	EE N	XOR A	AF
XOR H	AC	XOR L	AD
XOR B	AB	XOR C	A9
XOR D	AA	XOR E	AB
XOR (HL)	AE	XOR (IX+S)	DD AE S
XOR (IY+S)	FD AE S		

5.9. КОМАНДЫ ПЕРЕХОДА

Эта группа команд также относится к наиболее часто встречающимся в программах. Переходы нужны для того, чтобы отойти от последовательного выполнения программы и начать выполнение какого-либо блока с другого местп. В значительной степени эти команды эквивалентны командам БЕЙСИКа GO TO.

Переходы бывают: - условные и безусловные;
- относительные и абсолютные.

УСЛОВНЫЙ переход выполняется (или не выполняется) в зависимости от того, выполняется или нет какое-либо условие. Аналогичная конструкция БЕЙСИКа выглядит так:

IF .....THEN GO TO ....

БЕЗУСЛОВНЫЙ переход выполняется всегда, когда он встречается в программе. Он не связан никакими условиями.

АБСОЛЮТНЫЙ переход выполняется в заданный адрес. Адрес (двухбайтный) задается после кода операции.

ОТНОСИТЕЛЬНЫЙ переход выполняется на сколько-то шагов вперед или назад от адреса, в котором стоит команда, следующая за командой перехода. Здесь диапазон возможных переходов ограничен. Он может составлять от -128 до +127 байтов и называется смещением. Однобайтная величина смещения S задается вслед за кодом операции. Она всегда задана в дополнительной двоичной форме.

Мнемоники команд абсолютного перехода всегда начинаются с JP (JUMP = СКАЧОК), а относительного перехода - с JR (JUMP RELATIVE = скачок относительный).

Всего в этой группе 17 команд и мы рассмотрим их в четырех подгруппах.

5.9.1. Команды абсолютного безусловного перехода.

Мнемоника	Код	Мнемоника	Код
JP NN	C3 N N	JP (HL)	E9
JP (IX)	DD E9	JP (IY)	FD E9

Команда JP NN вызывает переход к адресу, заданному двумя

байтами NN. Этот адрес автоматически загружается в программный счетчик (регистровую пару PC) и программа продолжается, начиная с этого адреса.

Команды JP (HL), JP (IX), JP (IY) используют косвенную адресацию. Так, например, по команде JP (HL) выполняется переход к тому адресу, который содержится в регистровой паре HL.

5.9.2. Относительный безусловный переход.

В этой подгруппе всего одна команда: JR S, - ее код 18, за ним следует величина смещения S от -128 до +127, заданная в дополнительной двоичной форме. Напомним, что в этой форме к примеру FE (254) означает не переход на 254 байта вперед, а переход на 2 байта назад, т.е. (-2).

Другой особенностью относительного перехода является то, откуда начинает отсчитываться переход вперед или назад. Запомните, он отсчитывается не от того адреса, в котором размещалась команда JR, а от того, в котором стоит следующий за ней код операции, т.к. когда встретилась команда JR, в программный счетчик PC уже был заслан адрес начала следующей команды, и от него-то и начинается отсчет. Например:

```
Мнемоника     Адрес (десят.)     Код    Комментарий
.........
LD A,FF       60000               3E    Загрузить в акку-
              60001               FF    мулятор число 255.
DEC A         60002  <----┐       3D    Уменьшить акк-р на 1
JR FD         60003      |       18    Переход назад на
              60004      |       FD    три байта.
NOP           60005 -----┘       00    Пауза.
```

Применение относительных переходов дает ряд ценных преимуществ по сравнению с абсолютными как при написании, так и при эксплуатации программ. Обратите внимание на то, что в вышеприведенном примере нет никаких указаний на адреса. Мы его разместили начиная с адреса 60000, но если его разместить в другом месте, он будет точно также работать, выполняя те же действия. Отсутствие абсолютных переходов делает подпрограмму релоцируемой (перемещаемой). При наличии внутри нее абсолютных переходов она становится нерелоцируемой. Правда, надо не забывать, что

релоцируемая процедура может не допускать обращений к ней из других частей программы (см. 5.12), либо перед всяким таким обращением должно быть указано, где эта процедура находится. Может быть выделена специальная программная переменная, которая указывает на местоположение процедуры.

Релоцируемость удобна при написании тем, что Вы можете не задумываться над вопросом о том, где процедура будет находиться. А при эксплуатации Вы сможете отводить ей те области памяти, которые Вам в данный момент удобны в зависимости от прочего окружения /1.

5.9.3. Команды абсолютного условного перехода.

С помощью этих команд выполняется переход по заданному адресу, если выполняется какое-либо условие. В системе команд Z-80 это условие связано с состоянием флагов регистра F, а именно: флаг переноса (C), флаг нуля (Z), флаг знака (S), и флаг переполнения/четности (P/O).

Мнемоника	Код	Комментарий.
JP C,NN	DA N N	Переход по заданому адресу выполняется, если флаг переноса включен.

/1 Справка: Известные дисассемблирующие программы тоже бывают релоцируемыми и нерелоцируемыми. Так, например, для дисассемблера MONS 3 Вы сами можете задать адрес, в который он будет загружаться и работать. Это значит, что либо он внутри себя не содержит абсолютных переходов, либо они есть, но настраиваются после старта программы с помощью специальной инициализирующей процедуры.

Дисассемблеры MONITOR 16 и MONITOR 48 - нерелоцируемы. Поэтому желательно иметь и тот и другой. Для работы с блоками, загружаемыми в верхние области памяти, применяют MONITOR 16, который загружается в нижние области, и наоборот.

Есть, конечно, блоки, для которых неприменим ни тот, ни другой. В этом случае их надо либо резать на куски выгрузкой по частям SAVE "имя: CODE m,n, либо применять дисассемблер-MONS3, который, конечно, тоже имеет определенные пределы сво их возможностей.

Мнемоника	Код	Комментарий.
JP NC,NN	D2 N N	Переход по заданному адресу выполняется, если флаг переноса сброшен.
JP Z,NN	CA N N	Переход по заданному адресу выполняется, если флаг нуля включен.
JR NZ,NN	C2 N N	Переход по заданному адресу выполняется, если флаг нуля сброшен.
JR P,NN	F2 N N	Переход по заданному адресу выполняется, если флаг знака включен.
JR M,NN	FA N N	Переход по заданному адресу выполняется, если флаг знака сброшен.
JP PO,NN	E2 N N	Переход по заданному адресу выполняется, если включено нечетное число битов или если нет переполнения.
JP PE,NN	EA N N	Переход по заданному адресу выполняется, если включено четное число битов или если есть переполнение.

5.9.4. Команды относительного условного перехода.

Эти команды обеспечивают переход вперед или назад на заданное количество байтов в зависимости от состояния флагов C и Z.

Мнемоника	Код	Комментарий.
JR C,S	38 S	Переход на S байтов, если флаг C включен.
JR NC,S	30 S	Переход на S байтов, если флаг C сброшен.
JR Z,S	28 S	Переход на S байтов, если флаг Z включен.
JR NZ,S	20 S	Переход на S байтов, если флаг Z сброшен.

5.9.5. Комментарий к работе с флагами.

ФЛАГ ПЕРЕНОСА (C). Этот флаг находится в нулевом бите регистра F. Он показывает было или нет переполнение аккумулятора в абсолютной двоичной арифметике (т.е. был ли результат сложения больше 255 или результат вычитания меньше 0). На состояние

этого флага влияют далеко не все команды. Вы можете уточнить влияние команд на различные флаги по таблицам нашего "Справочника".

В двух словах:

1) Все команды сложения, вычитания и сравнения ADD, ADC, SUB, SBC, CP влияют на флаг переноса. Он включается, если было переполнение при сложении, заем при вычитании или если при сравнении содержимое операнда оказалось большим, чем содержимое аккумулятора.

2) Все команды логики AND, OR, XOR сбрасывают флаг переноса.

3) Команды сдвига (см. разд. 5.14.) влияют на флаг переноса.

ФЛАГ НУЛЯ (Z). Этот флаг находится в шестом бите регистра F. Он включается, если результат предыдущей операции был равен нулю, в противном случае он выключается.

Обратите внимание на мнемоническое противоречие. Флаг нуля включен и НЕ РАВЕН НУЛЮ, когда результат операции РАВЕН НУЛЮ.

Далее кратко:

1) При работе с одиночными регистрами на флаг нуля влияют результаты операций сложения (ADD, ADC, INC), вычитания (SUB, SBC, DEC), сравнения (CP), а также логики (AND, OR, XOR).

2) При работе с регистровой парой на флаг нуля влияют только арифметические операции ADC и SBC.

3) Команды загрузки регистров LD не влияют на флаг нуля, за исключением очень редко встречающихся для "Спектрума" команд LD A,I; LD A,R.

4) На флаг нуля влияют также команды, с которыми мы познакомимся несколько позже:

- команды сдвига (разд.5.14.);
- команды проверки битов (5.15.);
- команды блочного поиска (5.16.).

ФЛАГ ЗНАКА (S). Этот флаг находится в старшем (седьмом) бите регистра F. Он включается, если результат отрицательный и сбрасывается, если положительный.

Интересно, что для операций, выполняемых в дополнительной

двоичной арифметике, он равен старшему (седьмому) биту аккумулятора, который тоже определяет знак содержимого.

Основные особенности:

1) При работе с одиночными регистрами на флаг знака влияют результаты операций сложения (ADD, ADC, INC), вычитания (SUB, SBC, DEC), сравнения (CP), а также логики (AND, OR, XOR).

2) При работе с регистровой парой на флаг знака влияют только операции ADC и SBC.

3) Команды загрузки регистров LD не влияют на флаг знака, за исключением команд LD A,I; LD A,R.

4) На флаг знака влияют также команды, с которыми мы познакомимся несколько позже:

- команды сдвига (разд.5.14.);
- команды блочного поиска (5.15.).

ФЛАГ ПЕРЕПОЛНЕНИЯ/ЧЕТНОСТИ. Этот флаг находится во втором бите регистра F. Он имеет двойное назначение. Для одних команд он указывает на наличие четности числа включенных битов, для других - на наличие в результате операции переполнения в дополнительной двоичной арифметике.

ПОЯСНЕНИЕ: Понятие четности относится здесь не к числу, находящемуся в аккумуляторе, а к количеству его включенных битов. Например, число 33 (0011 0011) имеет четыре включенных бита, и, следовательно, флаг включен, а у числа 34 (0011 0100) - три включенных бита и флаг выключен.

Понятие "переполнение" здесь относится к дополнительной двоичной арифметике. Флаг включается, если в результате операции сложения возникает переход от числа, лежавшего в диапазоне 00...7F к числу из диапазона 80...FF или при вычитании - наоборот. Одним словом, он включается, если по правилам дополнительной двоичной арифметики возникает смена знака содержимого аккумулятора, неважно в какую сторону.

Точно установить какие операции влияют на этот флаг как на флаг четности, какие как на флаг переполнения, а какие вообще не влияют, Вы можете по таблицам нашего "Справочника...".

5.10. ОПЕРАЦИИ В ЦИКЛЕ

Очень часто в программах бывает нужным повторить какой-либо блок вычислений N раз. В БЕЙСИКе для этих целей служит оператор цикла:

```
FOR i = 1 TO N
..............
NEXT i
```

Все, что находится между операторами FOR и NEXT будет повторено N раз. При этом параметр i, называемый параметром цикла, будет с каждым проходом увеличиваться на единицу и, когда достигнет N, выполнение цикла будет прервано и работа программы продолжится с оператора, следующего за NEXT.

В машинных кодах для этой цели служит мощная команда DJNZ S. Эта команда относится к регистру B процессора. Когда процессор встречает эту команду, он уменьшает на единицу содержимое регистра B, проверяет его на ноль, и, если ноль пока не достигнут, выполняет переход на S байтов. Величина смещения S задана в двоичной дополнительной форме и переход может быть как вперед, так и назад, но чаще, конечно, назад. Вычисления повторяются до тех пор, пока не будет достигнут 0 в регистре B.

Мнемоника - DJNZ S; Код - 10 S.

Поясним это на примере.

Вы уже знаете, что когда процессор встречает команду NOP, он ничего не делает, т.е. просто выдерживается пауза. Продолжительность ее - 4 тактовых цикла. Длительность такта зависит от частоты задающего кварца Вашего компьютера и может несколько меняться от машины к машине. Но, поскольку она примерно 3,5 МГц, то продолжительность одного цикла - доли микросекунды. Предположим, что Вы хотите, чтобы процессор выдержал паузу примерно 200 тактовых циклов. Вы можете, конечно, поместить подряд 50 команд NOP, но это очень грубо и, к тому же, наносит большой ущерб количеству свободной оперативной памяти.

Воспользуемся возможностью организации цикла вычислений и многократным повторением команды NOP. По таблицам "Справочника..." можно найти, что время выполнения команды DJNZ занимает 13 тактовых циклов, если в регистре S не 0 и 8 циклов, если 0.

Всего, чтобы пауза длилась примерно 200 тактов, надо пов-

торить команду NOP 12 раз:

12*4 + 11*13 + 1*8 = 48 + 143 + 8 = 199

Тогда процедура будет выглядеть так:

Мнемоника	Код	Комментарий.
LD B,12	06	Загрузка в регистр B числа 12
	0C	0C HEX = 12 DEC
NOP	00	Пауза
DJNZ 253	10	Уменьшить содержимое регистра B
		на единицу
	FD	и переход назад на 3 байта.

5.11. КОМАНДЫ РАБОТЫ СО СТЕКОМ

Эти команды позволяют программисту копировать содержимое регистровых пар на стек и, наоборот, вызывать их оттуда в процессор. Таким образом, за один прием пересылаются сразу два байта, т.е. двухбайтное число (как правило, это адрес, но не всегда). Кроме того, в этой же группе есть команды, которые позволяют производить обмен содержимого некоторых регистров процессора.

Рассмотрим в этой группе три подгруппы команд.

5.11.1. Команды сохранения данных на стеке.

Эти команды начинаются с мнемоники PUSH, которой подходит несколько несерьезный русский эквивалент ЗАТОЛКНУТЬ (в дальний ящик). Итак, они применяются, когда надо временно сохранить до дальнейшей потребности содержимое регистровых пар.

Мнемоника	Код	Мнемоника	Код
PUSH AF	F5	PUSH HL	E5
PUSH BC	C5	PUSH DE	D5
PUSH IX	DD E5	PUSH IY	FD E5

Сначала на стек переносится старший байт, а затем младший. После операции PUSH указатель стека (регистр SP) уменьшается на две единицы. Уменьшается потому, что стек "растет" сверху вниз. Мы об этом уже говорили.

5.11.2. Команды вызова данных со стека.

По этим командам производится снятие данных со стека и загрузка их в необходимый регистр. Следует подчеркнуть, что данные, находящиеся на стеке, ни к какому регистру не привязаны. Независимо от того, из какого регистра они были выгружены, загружать их можно в любой другой регистр и вообще нет никакой физической возможности узнать из какого регистра эти данные выгружались. Если это важно, программист сам должен за этим следить.

Мнемоника команд, вызывающих данные со стека, начинается со слова POP ("ВЫТОЛКНУТЬ").

Мнемоника	Код	Мнемоника	Код
POP AF	F1	POP HL	E1
POP BC	C1	POP DE	D1
POP IX	DD E1	POP IY	FD E1

5.11.3. Команды обмена со стеком.

Этих команд три. Они начинаются с мнемоник EX (EXCHANGE - ОБМЕНИВАТЬ) и служат для того, чтобы отправить на стек содержимое регистровой пары, а содержимое вершины стека отправить в эту регистровую пару. То же самое можно сделать и использовав еще один регистр в качестве временного места хранения, но применением команд обмена это делается проще.

Мнемоника	Код
EX (SP),HL	E3
EX (SP),IX	DD E3
EX (SP),IY	FD E3

5.11.4. Замечания к операциям со стеком.

Стек процессора - машинный стек. Его не следует путать со стеком GO SUB, который организуется при работе в БЕЙСИКе и служит для запоминания номера строки, из которой вызывалась подпрограмма, Его также не следует путать со стеком калькулятора. "Спектрум" имеет калькулятор, который управляется процедурами, размещенными в ПЗУ, и может использоваться при программировании

в машинных кодах. С ним мы познакомимся довольно глубоко в разделе 5.13.

Машинный стек служит не только для того, чтобы обеспечивать программисту удобное место для временного хранения данных. Его основное назначение - хранить адреса, из которых вызываются подпрограммы. Это нужно для того, чтобы по завершении подпрограммы процессор знал, куда ему надо вернуться для продолжения вычислений. В связи с этим, при работе со стеком, от Вас требуется определенная внимательность.

Если Вы находитесь в подпрограмме, то все, что Вы поместите на стек, должно быть снято оттуда до того, как произойдет выход из подпрограммы в вызывающую программу. С другой стороны, поскольку выход выполняется по адресу, находящемуся в двух байтах вершины стека, Вы можете искусно управлять логикой работы программы, манипулируя числами, находящимися на вершине стека, в частности, так организуются сложные вычисления в программах.

5.12. ВЫЗОВ ПОДПРОГРАММ

В БЕЙСИКе подпрограммы вызывались с помощью оператора GO SUB, а возврат после исполнения подпрограммы выполнялся командой RETURN.

В машинных кодах им эквивалентны команды CALL - ВЫЗОВ и RET - ВОЗВРАТ.

5.12.1. Команды вызова.

Вызов может быть безусловным или сопровождаться каким-либо условием, связанным с проверкой состояния флагов регистра F. Всего имеется 9 команд для вызова подпрограмм.

Мнемоника	Код	Комментарий
CALL NN	CD NN	Это безусловный вызов процедуры, находящейся по адресу NN.
CALL C,NN	DC NN	Вызов подпрограммы, если флаг переноса включен.
CALL NC,NN	D4 NN	Вызов подпрограммы, если флаг переноса выключен.

Мнемоника	Код	Комментарий
CALL Z,NN	CC NN	Вызов подпрограммы, если флаг нуля включен.
CALL NZ,NN	C4 NN	Вызов подпрограммы, если флаг нуля выключен.
CALL N,NN	FC NN	Вызов подпрограммы, если флаг знака включен.
CALL P,NN	F4 NN	Вызов подпрограммы, если флаг знака выключен.
CALL PE,NN	EC NN	Вызов подпрограммы, если флаг переполнения/четности включен.
CALL PO,NN	E4 NN	Вызов подпрограммы, если флаг переполнения/четности выключен.

5.12.2. Возврат из подпрограммы.

Этих команд тоже 9. Они начинаются с мнемоники RET. Когда процессор встречает такую команду, он снимает два верхних числа с машинного стека и отправляет их в программный счетчик (регистр PC). Имейте в виду, что восстановленный адрес из машинного стека может быть и не тем, который был туда отправлен по команде CALL. Вы могли его и изменить как намеренно, так и ненамеренно.

Мнемоника	Код	Комментарий
RET	C9	Безусловный возврат в вызывающую программу.
RET C	D8	Возврат, если флаг переноса включен.
RET NC	D0	Возврат, если флаг "C" выключен.
RET Z	C8	Возврат, если флаг нуля включен.
RET NZ	C0	Возврат, если флаг нуля выключен
RET M	F8	Возврат, если флаг знака включен.
RET P	F0	Возврат, если флаг знака выключен.
RET PE	E8	Возврат, если флаг переполнения/ четности включен.
RET PO	E0	Возврат, если флаг переполнения/ четкости выключен.

5.13. КОМАНДЫ ОБРАЩЕНИЯ К ПЗУ

Разработчики процессора Z-80 предусмотрели в нем возможность легкого и быстрого вызова наиболее часто употребляемых подпрограмм, если они размещены в нижних областях памяти. У "Спектрума" в нижних областях находится ПЗУ и эти команды позволяют к нему обращаться без использования конструкции CALL NN. Эти команды начинаются с мнемоники RST (RESTART - ПОВТОРНЫЙ СТАРТ). В этой подгруппе 8 команд. Каждая из них обращается к ПЗУ и стартует процедуру, расположенную по адресу, указанному в команде.

Мнемоника	Код	Эквивалент
RST 0	C7	CALL 0000
RST 8	CF	CALL 0008
RST 10	D7	CALL 0010
RST 18	DF	CALL 0018
RST 20	E7	CALL 0020
RST 28	EF	CALL 0020
RST 30	F7	CALL 0030
RST 38	FF	CALL 0038

Некоторыее из этих процедур используются интерпретатором БЕЙСИКа и при работе в машинных кодах необходимость в них практически не возникает. Другие же используются чрезвычайно активно, без них трудно обойтись.

5.13.1. Команда RST 0.

~~~~~~~~~~~~~~~~~~~~~

По этой команде происходит запуск процедуры инициализации компьютера. Эффект от действия этой команды такой же, как и от выключения и повторного включения питания. Из БЕЙСИКа этот результат достигается командой RANDOMIZE USR 0.

5.13.2. Команда RST 8.

~~~~~~~~~~~~~~~~~~~~~

Это старт процедуры обработки ошибки. В результате ее работы в системную переменную ERR NR засылается код ошибки. Машинный стек очищается.

5.13.3. Команда RST 10.

Это точка входа системной программы, выполняющей печать символов на экране или на принтере, но она может применяться и для печати токенов ключевых слов, цветовых элементов, а также управляющих символов, например AT, TAB.

В ПЗУ "Спектрума" имеются и более мощные средства, способные выполнить печать сразу целых строк, но они в своей работе обязательно используют команду RST 10.

Выбор, куда выполнять печать - на экран или на принтер, зависит от того, какой канал открыт в данный момент. Канал может открываться из БЕЙСИКа командой OPEN$ или из машинного кода путем вызова соответствующей процедуры из ПЗУ.

Более подробно на примерах мы рассмотрим работу команды RST 10 во второй части - "Практикум...". Весьма подробно эта команда рассмотрена вместе с соответствующими процедурами ПЗУ в нашей книге "Элементарная графика".

5.13.4. Команда RST 18.

Процедура, находящаяся по этому адресу, выполняет при работе интерпретатора прем очередного интерпретируемого символа, находящегося в системной переменной CHADD и проверку его на то, является ли он печатным или нет (управляющим кодом, цветовым кодом и т.п.).

5.13.5. Команда RST 20.

Эта процедура выполняет последовательный просмотр символов вплоть до конца БЕЙСИК-строки. Так же, как и RST 18, используется в интерпретаторе БЕЙСИКа.

5.13.6. Команда RST 28.

Инициализирует внутренний калькулятор компьютера. Это точка входа обширного пакета программ. Поскольку в повседневной работе программисту не обойтись без выполнения математических расчетов, а работа калькулятора "Спектрума" в машинных кодах

чрезвычайно слабо освещена, мы здесь даем подробнейшие указания по работе с ним, несмотря на то, что эта информация относится скорее к ПЗУ, а не к процессору.

Итак, встроенный калькулятор "Спектрума" дает Вам возможность выполнения сложных математических вычислений без выхода из машинных кодов. Более того, можно сказать, что он позволяет Вам выполнять многое такое, что в БЕЙСИКе просто невозможно.

Очевидным недостатком процессора Z-80 является то, что его система команд не позволяет выполнять математические и арифметические операции более сложные, чем сложение и вычитание небольших целых чисел. Хотя те, кто работает со "Спектрумом" в машинных кодах, могут воспользоваться процедурой ПЗУ, размещенной по адресу 30A9 и предназначенной для перемножения двух небольших целых чисел. Вызывается она CALL A9 30. Первый сомножитель должен находиться в регистровой паре HL, второй - в регистровой паре DE. Результат помещается в регистровую пару HL. Во время работы этой процедуры не портится содержимое других регистров, кроме регистра A. Если при выходе из этой процедуры результат оказывается слишком большим, включается флаг переноса (флаг C регистра F).

Даже если принять во внимание наличие этой процедуры, как быть с делением, с операциями над дробями, квадратными корнями, тригонометрическими функциями и т.п.?

Фактически такая ограниченность машинных кодов связана с тем, что нормально регистры процессора предназначены для работы с малыми целыми числами от 0 до 255. Вы не можете разделить 1 на 2 потому, что нет регистра, который мог бы хранить число 0.5. Калькулятор же обходит все эти проблемы за счет того, что работает с числами, записанными не в двухбайтной, а в пятибайтной форме, что позволяет выражать не только целые, но и действительные числа.

Такая форма представления действительных и целых чисел называется ИНТЕГРАЛЬНОЙ. Кстати, при работе в БЕЙСИКе все числа, которые участвуют в Вашей программе, кроме номеров строк, тоже представлены в интегральной форме. Проверьте это. Введите строку 10 LET AAA=10*20, а теперь посмотрите, как она вписалась в память: FOR I=23755 TO 23781: PRINT I, PEEK I: NEXT I .

Давайте рассмотрим, как эти числа представляются в пяти-

байтной форме.

ПОЛОЖИТЕЛЬНЫЕ ЦЕЛЫЕ ЧИСЛА.

Они могут принимать значения от 0 до 65535. В принципе для их записи хватило бы и двух байтов. Если старший байт мы назовем hh, а младший - ll, то в пятибайтной форме такое число запишется:

00 00 ll hh 00

Как видите, опять младший байт стоит впереди старшего.

ОТРИЦАТЕЛЬНЫЕ ЧИСЛА.

Они могут принимать значения от -1 до -65535. При этом -1 выглядит как FFFF; -2 как FFFE и т.д. В пятибайтной форме они записываются:

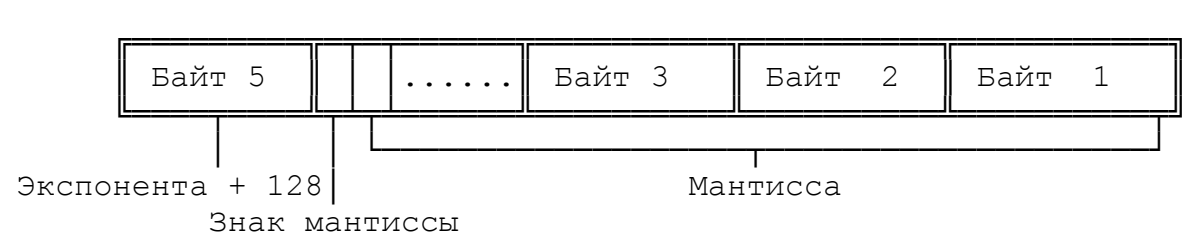
00 FF ll hh 00

ДЕЙСТВИТЕЛЬНЫЕ ЧИСЛА.

"Спектрум" в состоянии работать с действительными числами из диапазона от -1.7014118*10^38 до +1.7014118*10^38. Числа, выходящие за пределы этого диапазона, недоступны для компьютера и вызывают сообщение об ошибке:

6 Number too big (6 Число слишком велико)

В пятибайтной (интегральной) форме такие большие числа хранятся в экспоненциальном представлении. В старшем (пятом) байте хранится экспонента числа, увеличенная на 128, а в остальных четырех байтах - мантисса. Причем старший бит четвертого байта указывает на знак мантиссы. Если он включен - число отрицательное, если же выключен - положительное.



Для тех, кто не помнит, что такое экспоненциальная форма, кратко поясним.

Предположим, что Вам надо записать какое-то очень большое

число. Если Вы будете его делить раз за разом на два до тех пор, пока оно не станет по абсолютной величине меньше единицы, то сколько раз Вам пришлось его делить - это и есть экспонента, а то, что осталось после последнего деления - мантисса. Обратите внимание на то, что мантисса всегда больше 0,5, но меньше 1. Мантисса хранится в четырех младших байтах, а экспонента - в пятом (старшем). Обратная операция выполняется так: мантиссу надо умножить на двойку в степени экспоненты.

Если же число было малой десятичной дробью, например 0,0000375, то делить на два его, конечно, бессмысленно. В этом случае его нужно умножать на два до тех пор, пока оно не станет больше, чем 0.5, но меньше, чем 1. Только экспонента в этом случае считается отрицательной. На знак экспоненты указывает ее величина. Если в старшем байте стоит число, большее 128, то экспонента положительная, а если меньше, то отрицательная.

"Спектрум" ограничен и в работе с очень малыми дробями. Числа от -1.469367*10^(-39) до +1.469367*10^(-39) он не воспринимает и считает их нулем.

А как же в пятибайтной форме изображается ноль? Единственная возможная и самая естественная форма: 00 00 00 00 00

Поскольку представление дробей в двоичной форме выглядит не очень очевидным, да к тому же может возникнуть путаница со знаками, мы здесь приведем пример из книги Виккерса "БЕЙСИК ZX-Спектрума" (эта книга является официальной фирменной инструкцией к компьютерам "ZX-Spectrum 16" и "ZX-Spectrum 48").

Предположим, что Вам надо записать в интегральной форме число 1/10. Оно меньше, чем 0,5. Будем последовательно умножать его на два, а когда старший разряд будет переходить за единицу, будем сносить ее в двоичную дробь.

Десятиричная форма	Двоичная форма
0.1	0.
0.1*2=0.2	0.0
0.2*2=0.4	0.00
0.4*2=0.8	0.000
0.8*2=1.6	0.0001
0.6*2=1.2	0.00011

```
0.2*2=0.4                    0.000110
0.4*2=0.8                    0.0001100
      ┌──────────────> ──────────────────┐
0.8*2=1.6                    0.00011001
```

и так далее.

Таким образом, двоичная форма десятиричного числа 0.1:

0.0001100110011001100110011001100... и т.д.

Сместим дробную точку вправо до первой единицы. Количество шагов даст нам экспоненту, а оставшаяся часть будет мантиссой. В нашем случае экспонента равна -3, а мантисса -

0.1100110011001100... .

Для хранения этого числа в компьютере оно преобразуется в интегральную форму следующим путем:

1) Запишем в пятом байте величину экспоненты, увеличенную на 128, т.е. в нашем примере это будет 125.

2) Запишем первые 8 знаков мантиссы в четвертом байте, очередные 8 - в третьем и т.д.

3) В старшем (седьмом) бите четвертого байта мантиссы поместим 0, если число положительное и 1, если число отрицательное.

Итак, мы имеем:

5 байт	4 байт	3 байт	2 байт	1 байт
01111001	01001100	11001100	11001100	11001100

Обратите внимание на то, что хоть мы и заменили значащую единицу в седьмом бите четвертого байта на 0 (чтобы показать, что число положительное), тем не менее компьютер все же знает, что здесь должна быть единица. Это происходит потому, что мантисса всегда больше, чем 0.5, а значит этот бит равен единице ВСЕГДА, просто для положительного числа не показан.

ПРЕДСТАВЛЕНИЕ СТРИНГОВ В ПЯТИБАЙТНОЙ ФОРМЕ.

Стринги (строки) можно тоже представить в пятибайтной форме, но поскольку они могут быть длинными, то текст должен храниться где-то отдельно. Например, Вам надо хранить в строковой переменной слово "Спектрум". В этом случае пятый байт всегда равен нулю. Четвертый и третий байты содержат младшую и старшую часть адреса, в котором хранится первый символ стринга (в нашем

случае "С"). Обозначим их aa, bb. Второй и первый байт содержат длину этого стринга. Обозначим их cc, dd. Итого:

00 aa bb cc dd

СТЕК КАЛЬКУЛЯТОРА.

Это область памяти, в которой калькулятор компьютера выполняет все операции. Он работает точно так же, как и машинный стек процессора. Разница состоит в том, что:

- во-первых, машинный стек "растет" сверху вниз, а стек калькулятора - снизу вверх;

- во-вторых, машинный стек оперирует с двухбайтными величинами, которые как правило выражают адреса, а стек калькулятора оперирует с пятибайтными числами, которые выражают действительные числа или символьные строки (стринги).

- в-третьих, на место нахождения вершины машинного стека указывает содержимое регистра SP (указатель стека). На основание (начало) стека калькулятора указывает системная переменная STKBOT. Ее адрес - 23651, а длина - 2 байта. Есть еще одна системная переменная STKEND, которая указывает на байт ОЗУ, следующий за вершиной стека калькулятора. Ее адрес 23653 и длина - 2 байта. Если стек пуст, то содержимое этих двух системных переменных совпадает. Самое верхнее число на стеке, таким образом, располагается в адресах от (STKEND)-5 до (STKEND)-1.

****
* Примечание: (STKEND) здесь стоит в скобках потому, что это *
* косвенная адресация. Если адрес STKEND равен 23653, то *
* адрес (STKEND) - это то, что содержится в STKEND. *
****

Применение стека позволяет следующим образом работать с калькулятором:

- помещать данные на стек;

- выполнять с ними математические операции;

- снимать результат со стека.

Те, кто знаком с языком программирования ФОРТ, обнаружат, что операции с калькулятором очень похожи на работу в ФОРТ-системах.

КОМАНДЫ КАЛЬКУЛЯТОРА.

Команды калькулятора начинаются с инструкции RST 28 (код EF) - это как бы "включение калькулятора". За этим кодом должна идти последовательность байтов, каждый из которых является командой калькулятора. Таким образом, эта последовательность байтов является последовательностью команд. Вы, конечно, знаете, что всякая последовательность команд представляет из себя программу. Итак, с команды RST 28 начинается выполнение программы калькулятора. Эта программа очевидно написана не на БЕЙСИКе, но и не в машинных кодах. У калькулятора своя система команд, отличная от системы команд процессора Z-80. Можно считать, что эта программа написана в кодах калькулятора. Завершается эта программа командой калькулятора "end calc" (конец вычислений). Ее код - 38. Таким образом, простейшая программа калькулятора выглядит так:

Код	Значение	Комментарий
EF	RST 28	Включение калькулятора
38	end calc	Выключение калькулятора

Как видите, эта программа ничего не делает, но тем не менее имеет побочный эффект. Он состоит в том, что при выходе из калькулятора в регистре DE процессора находится содержимое (STKEND), а в регистре HL - (STKEND)-5. Другими словами, регистры HL и DE становятся указателями стека калькулятора. HL указывает на первый байт (байт экспоненты) числа, находящегося на вершине стека, а DE указывает на первый свободный байт памяти ОЗУ, находящийся над стеком калькулятора.

СИСТЕМА КОМАНД КАЛЬКУЛЯТОРА

Полная таблица системы команд калькулятора приведена в нашем "Справочнике по программированию в машинных кодах Z-80", см. третью часть нашего пособия.

Здесь мы рассмотрим основные команды и укажем на те особенности, которые должны быть учтены при программировании. Каждая команда состоит из шестнадцатиричного кода и имени, которое поясняет ее назначение. Мы будем записывать имя команд калькулятора строчными буквами, чтобы отличать их от прописных букв мнемоник АССЕМБЛЕРа.

Первая команда, которую мы рассмотрим - это add (сложе-

ние). Она предназначена, как следует из названия, для сложения двух чисел. Они должны находиться на вершине стека калькулятора. Работа команды состоит в том, что эти числа снимаются со стека, а на их место помещается их сумма. Таким образом, на стеке оказывается на одно число меньше, чем было. Команда add называется бинарной потому, что для ее выполнения требуются два операнда, хотя после операции получается один результат.

БИНАРНЫЕ ОПЕРАЦИИ.

Есть еще много различных бинарных операций и функций. Как и следовало ожидать, бинарными являются операции "subtract" (вычитание), "multiply" (умножение), "devide" (деление). Есть еще команда "power" (возведение в степень), которая возводит одно число в степень второго.

УНИТАРНЫЕ ОПЕРАЦИИ.

Калькулятор имеет также и унитарные операции, которые используют только один операнд, снимая его с вершины стека и заменяя его результатом. Такова, например, операция "sqr" (вычисление квадратного корня). Унитарные операции оставляют длину стека калькулятора без изменений.

Рассмотрим в качестве демонстрационного примера программу для вычисления корня квадратного из суммы синуса и косинуса числа X. Перед началом программы величина X должна находиться в верхних пяти байтах стека калькулятора.

SQR (COS X + SIN X)

Пример достаточно прост, чтобы Вы сами смогли его проследить с помощью таблицы системы команд калькулятора (см. Справочник...) и проникнуться духом техники выполнения вычислений и работы со стеком калькулятора.

Код	Команда	Число на вершине	Число перед ним	Комментарий
EF	RST 28	X	-	Включение калькулятора
31	duplicate	X	X	Повторение числа на вершине стека.
20	cos	cos X	X	Вычисление косинуса
01	exchange	X	cos X	Поменять местами верх- два числа на стеке

Код	Команда	Число на вершине	Число перед ним	Комментарий
1F	sin	sin X	cos X	Вычисление синуса
0F	add	sinX+cosX	-	Сложение
28	sqr	sqr(sinX+cosX)	-	Вычисление квадратного корня
38	end calc	sqr(sinX+cosX)	-	Выключение калькулятора

В состав команд калькулятора входят также логические функции and и or. Они работают также, как и в БЕЙСИКе.

X and Y равно X, если Y не равен нулю, в противном случае результат - 0.

X or Y равно X, если Y не равен нулю, в противном случае результат - единица.

Логическая функция not дает единицу, если оригинал равен нулю, в противном случае - ноль.

Но здесь есть и две новые команды:

less_zero (Код 36) дает единицу, если оригинал меньше нуля, в противном случае дает ноль.

gtr_zero (Код 37) дает единицу, если оригинал больше нуля, в противном случае дает ноль.

К Вашим услугам пять готовых констант, которые могут быть помещены на стек немедленно. Их коды от A0 до A4 включительно.

stk_zero (код A0) помещает на стек число 0.

stk_one (код A1) помещает на стек число 1.

stk_half (код A2) помещает на стек число 0.5

stk_pi/2 (код A3) помещает на стек одну вторую числа "пи".

stk_ten (код A4) помещает на стек число 10 (десятиричное)

Это очень удобно, поскольку эти числа постоянно встречаются в расчетах.

Некоторые функции калькулятора могут вызвать удивление. Например, Вы можете увидеть в таблице две функции usr. Это usr (str) и usr (num), т.е. usr (стринг) и usr (число).

В БЕЙСИКе есть только одно ключевое слово USR. Разница в том, что с его помощью в БЕЙСИКе можно выполнить две разные операции. Например, USR "J" даст Вам адрес символа графики

пользователя, помещенного для использования в графическом режиме на клавишу "J". В этой команде "J" - стринг, а не число.

С другой стороны, USR_число даст Вам результат работы машиннокодовой программы, начинающейся с адреса, заданного этим числом. Попробуйте, например, PRINT 65536-USR 7962. Вы получите, сколько байтов свободной памяти у Вас осталось.

Калькулятор не настолько мудр. Он сам не в состоянии определить, является ли число на вершине стека числом или символьным стрингом. Ему надо об этом сообщить. Отсюда и необходимость в двух различных функциях usr.

Более того, команда калькулятора usr (число) одна из самых удивительных команд набора. Вот, что она делает:

- с вершины стека снимается число и отправляется в регистровую пару BC. Разумеется, если оно помещается по размеру, а помещается туда только целое положительное число. В противном случае Вы получите сообщение об ошибке;

- затем содержимое BC принимается за адрес и вызывается программа, начинающаяся с этого адреса;

- по завершении работы этой программы то значение регистра BC, которое находится там в момент выхода, помещается на вершину стека калькулятора;

- после всего этого калькулятор переходит к исполнению следующей команды.

Есть и другие инструкции, различающие стринги и числа. Например, инструкция add имеет две различные команды калькулятора и, соответственно, два различных кода для работы со стрингами и числами:

Команда add (код 0F) складывает числа нормальным образом - так, что 1 + 2 = 3, в то время, как команда s_add (код 17) выполняет слияние стрингов, так что

"SPEC" + "TRUM" = "SPECTRUM".

Интересно работает команда peek (2B). Она снимает со стека адрес, обращается по нему в память, находит то, что там содержится и помещает число на вершину стека.

ПАМЯТЬ КАЛЬКУЛЯТОРА

Калькулятор "Спектрума" имеет шесть ячеек памяти. В принципе, как мы увидим позже, их количество можно увеличить. В каждой из этих ячеек можно хранить число или стринг. Набор команд калькулятора имеет команды от C0 до C5 включительно для того, чтобы помещать в эти ячейки число, находящееся на вершине стека. При этом число с вершины стека не снимается, а только выполняется его копирование в заданную пятибайтную ячейку.

Обратные команды от E0 до E5 могут вызывать число или стринг из памяти калькулятора и помещать его на вершину стека.

В качестве примера рассмотрим опять же вычисление

SQR (SIN X + COS X),

но уже другим способом, с использованием памяти калькулятора.

Код	Команда	Число на вершине	Число перед ним	Комментарий
EF	RST 28	X	-	Включение калькулятора
C3	store M3	X	-	Помещение числа в третью ячейку памяти
20	cos	cos X	-	Вычисление косинуса
E3	recall M3	X	cos X	Вызов числа из третьей ячейки памяти
1F	sin	sin X	cos X	Вычисление синуса
0F	add	sin X + cos X	-	Сложение
28	sqr	sqr(sinX+cosX)	-	Вычисление квадратного корня
38	end calc	sqr(sinX+cosX)	-	Выключение калькулятора

Прежде, чем мы перейдем к обсуждению того, как увеличить количество ячеек памяти калькулятора, мы должны остановиться еще на одном вопросе.

Дело в том, что есть некоторые команды калькулятора и процедуры ПЗУ, которые могут нарушить содержимое ячеек памяти калькулятора. Это происходит потому, что они сами их используют при своей работе. Это надо знать, чтобы не совершить ошибку в расчетах.

sin, cos, tan, asn, acs, atn, ln нарушают содержимое ячеек памяти калькулятора M0, M1, M2.

exp - ячейки M0, M1, M2, M3.

int, mod_div, get_argt - M0.

str$ - нарушает все шесть ячеек от M0 до M5.

usr_n - может нарушить или не нарушить содержимое любой ячейки.

Печать графических символов (от CHR$128 до CHR$145) нарушает ячейки M0 и M1 памяти калькулятора.

Обращение к процедуре ПЗУ PRINT_FP, находящейся по адресу 2DE3 и служащей для печати десятичных чисел с плавающей точкой, разрушает содержимое всех ячеек памяти.

"Спектрум", строго говоря, имеет немало "жучков" (а попросту ошибок) в своем ПЗУ. Как мы указали, функция вычисления целой части числа int нарушает содержимое нулевой ячейки памяти калькулятора, но если говорить по-правде, это нарушение происходит только тогда, когда вычисляется целая часть отрицательного числа. Если аргумент положительный, то ячейка M0 останется нетронутой.

Функция mod_div предназначена для вычисления частного и остатка от деления одного числа на другое, скажем X и Y. Она должна снять со стека два числа, выполнить вычисления и заменить их на стеке двумя новыми числами - остатком X-Y*INT(X/Y) и частным INT(X/Y). К сожалению, процедура ПЗУ, которая выполняет эти действия, совершенно не принимает во внимание тот факт, что int может разрушить M0. В результате, если X - отрицательное число, функция mod div дает неверный результат /1.

РАСШИРЕНИЕ ПАМЯТИ КАЛЬКУЛЯТОРА.

Введение в работу более шести ячеек памяти основывается на использовании системной переменной MEM. Она расположена по адресу 5C68 (23656). Как Вы знаете, каждая единица хранения на стеке занимает пять байтов, поэтому и каждая ячейка памяти калькулятора должна тоже занимать пять байтов. M0 находится по

/1 Читатели, желающие более подробно ознакомиться с ошибками ПЗУ "Спектрума" могут это сделать в 9-12-м выпусках нашего периодического издания "ZX-РЕВЮ" за 1992 год, где приведен весьма детальный обзор, выполненный нами по материалам иностранной печати.

адресу, который указывает MEM. M1 - по адресу (MEM)+5; M2 - по адресу (MEM)+0A и т.д. Нормально системная переменная MEM содержит 5C92 (23698), т.е. указывает на системную переменную MEMBOT. Поскольку MEMBOT имеет 30 байтов, то в этой области можно разместить ровно 6 ячеек памяти.

Предположим, что Вы хотите иметь 32 ячейки памяти калькулятора (это максимальное возможное количество). Первое, что Вам для этого надо сделать - это выделить 160 байтов свободной памяти. Можно использовать область памяти выше RAMTOP, предварительно установив RAMTOP командой CLEAR. Другой способ - разместить эту память в рабочей области, что можно сделать из машинного кода командой RST 30. Для этого надо разместить в регистре BC необходимое количество байтов (в нашем случае это 160 десят. или A0 шестнадц.) и дать команду RST 30. Процедура отработает и указанное количество байтов будет отведено. На выходе из этой процедуры регистр DE будет содержать адрес первого, а регистр HL - адрес последнего байта выделенной памяти. После этого все, что Вам нужно - это перегрузить содержимое DE в системную переменную MEM.

Для работы с новыми ячейками памяти служат команды:

C6 - для ячейки M6 (store M6);

C7 - для ячейки M7 (store M7);

.....

DF - для ячейки M1F(store M1F);

Вызов данных из этих новых ячеек выполняется командами от E6 (recall M6) до FF (recall M1F).

Однако есть и ограничения на изменение содержимого переменной MEM. Если в ней будет содержаться адрес, отличный от предварительно установленного 5C92, то не будет работать команда калькулятора str$ (код 2E) и не будет работать процедура печати чисел с плавающей точкой PRINT_FP. Если Вам они необходимы, Вы должны восстанавливать 5C92 в переменной MEM перед их использованием.

ВЫПОЛНЕНИЕ ПЕРЕХОДОВ В КОДАХ КАЛЬКУЛЯТОРА.

Для этой цели служит команда jump. Это двухбайтная команда. В первом байте стоит ее код - 33, а за ним идет величина

"смещения", указывающая на сколько байтов выполняется переход. Инструкция очень похожа на команду машинного кода JR S, но имеет отличие. Мы уже неоднократно указывали на то, что величина "смещения" отсчитывается в машинных кодах не от адреса, в котором размещена команда JR, а от адреса, в котором находится код следующей за ней операции. В кодах же калькулятора величина "смещения" отсчитывается именно от того адреса, в котором стоит команда jump.

Величина "смещения " задается в двоичной дополнительной форме, поэтому возможны переходы как вперед, так и назад. Вперед на 0...127 (0...7F) байтов и назад на -1...-128 (FF...80) байтов.

ОБРАБОТКА УСЛОВИЙ.

Калькулятор имеет средства для организации вычислений, аналогичных тем, которые выполняет конструкция БЕЙСИКа IF...THEN. Между IF и THEN стоит выражение, которое подлежит проверке и может принимать значения TRUE (ИСТИНА) и FALSE (ЛОЖЬ). После THEN стоит выражение, которое выполняется в том случае, если условие имеет значение TRUE.

В кодах калькулятора может быть выполнено все то же самое. Кое-что, может быть, даже проще, но есть одно ограничение. Если в БЕЙСИКе после THEN выходным может быть любой оператор, то в кодах калькулятора выходной может быть только команда относительного перехода. Это означает, что можно создавать только конструкции типа IF...THEN GO TO ...

Это ограничение можно даже и не рассматривать как ограничение, потому, что достаточно немного изменить структуру программы и Вы опять можете делать что угодно.

Сама же конструкция перехода по условию выражается в калькуляторе командой jump_true. Как и команда jump это двухбайтная команда. Первым идет код операции 00, а за ним величина s ("смещение"), задаваемая одним байтом в двоичной дополнительной форме. "Смещение" отсчитывается точно так же, как и в команде jump, рассмотренной выше. Работает команда следующим образом. Со стека снимается верхнее число. Если оно имеет значение TRUE, т.е. не равно нулю, выполняется переход. Если же со стека поступил 0, то переход не выполняется, а байт, выражающий ве-

личину "смещения" игнорируется.

Идея использования ненулевых значений для выражения логического значения TRUE и нулевых значений для выражения FALSE - это не просто удобный прием, это основа логики работы "Спектрума". Фактически мы можем зайти так далеко, что признаем TRUE и FALSE иным типом данных, отличным от чисел и стрингов - логическими данными.

Например, команда калькулятора lt_z (меньше, чем 0 код - 36) заменит число, находящееся на вершине стека на TRUE, если это число отрицательное, и на FALSE, если оно положительное.

Другой пример. Команда not (код 30) заменит значение TRUE на вершине стека на FALSE или наоборот. Такой логический подход (вместо числового) упрощает понимание работы программ.

Команда калькулятора not, если на нее смотреть под таким углом зрения, имеет двойное значение. Во-первых, она меняет логическое значение, находящееся на вершине стека на противоположное. Во-вторых, если содержимое вершины стека рассматривать как число, то после этой команды там будет 0. Таким образом, эта команда может быть использована для обнуления вершины стека, если там был не ноль.

КОМАНДЫ "И" (and).

Калькулятор имеет две команды and. Одна для работы с числами - n_and (код 08), а другая - для работы со стрингами - s_and (код 10).

Команда n_and работает достаточно прямолинейно:

TRUE n_and TRUE = TRUE
TRUE n_and FALSE = FALSE
FALSE n_and FALSE = FALSE

Она может быть использована и в такой форме:

<число> n_and <логическое значение>

Это же самое можно было бы сделать и в БЕЙСИКе. Попробуйте:

```
PRINT (7 AND (X>9))
```

Подставьте вместо X числа 10 и 8.

Вторая команда - s_and. Для нее на вершине стека должен быть стринг, а за ним следовать логическое значение. Команда

работает так. Логическое значение удаляется со стека, а стринг заменяется пустым стрингом в том и только в том случае, если логическое значение было FALSE.

Таким образом, мы можем с помощью калькулятора приводить выражения к логическому результату. Другими словами, рассчитывать значение выражений в виде TRUE или FALSE, а затем выполнять переход по условию.

B-РЕГИСТР КАЛЬКУЛЯТОРА.

Микропроцессор Z-80 имеет много регистров, доступ к которым открывает программирование в машинных кодах. Калькулятор же имеет только один регистр - "B". В нем можно хранить целое число от 0 до 255.

При включении калькулятора по команде RST 28 происходит автоматический перенос содержимого регистра B процессора в регистр B калькулятора. Совершенно аналогично при окончании вычислений в регистр процессора B переносится то, что было в регистре B калькулятора перед командой end_calc. Более того, "Спектрум" имеет системную переменную под названием B_REG (23655), в которой запоминается то, что было в регистре B калькулятора перед командой end_calc.

Система команд калькулятора имеет команду execute_B (код 38). Она дает команду на исполнение той инструкции, которая содержится в регистре B. Это пример косвенного указания на команду. Таким образом, последовательностью команд:

```
RST 28       EF
execute_B    3B
end_calc     38
```

можно выполнить любую из команд калькулятора, если ее код предварительно поместить в B-регистр процессора. Разумеется, это справедливо только для однобайтных команд.

КОМАНДЫ, ЗАВИСИМЫЕ ОТ РЕГИСТРА "B"

Многие команды калькулятора требуют для своей успешной работы, чтобы в регистре B находилось какое-либо число. Наиболее широко применимы из этих команд команды сравнения: =, <, >, <=, >=, <>. В нашем "Справочнике..." все эти команды оговорены. Обращаем Ваше внимание на то, что в системе команд калькулятора

нет кода на загрузку в B-регистр числа. Чтобы это сделать, надо выйти в машинный код, загрузить это число, а затем вновь вернуться в калькулятор.

Например, Вам нужна команда, проверяющая тот факт, что число, находящееся на вершине стека меньше, чем следующее за ним число: n_lt. Эта команда снимает оба числа со стека и заменяет их на TRUE, если первое число меньше второго, а в противном случае - на FALSE. Код этой операции OD, но для того, чтобы она успешно работала, необходимо, чтобы в регистре B также находилось число 0D. Установка там этого числа выполняется с выходом из калькулятора:

Мнемоника	Код	Комментарий
.....	..	.....
end_calc	38	Выход из калькулятора
LD B,OD	06	Загрузить в регистр B число OD.
	OD	
RST 28	EF	Вызов калькулятора
n_lt	OD	Сравнение чисел
.....	..	..... и т.д.

Точно также команда s_eq (код 16), которая проверяет факт равенства двух стрингов, требует, чтобы в регистре B находилось число 16. Таких команд у калькулятора немало. Будьте внимательны при работе с ними.

КОМАНДЫ, ИЗМЕНЯЮЩИЕ СОДЕРЖИМОЕ B-РЕГИСТРА

Выше мы уже говорили, что есть команды, изменяющие содержимое некоторых ячеек памяти калькулятора. Теперь же дополнительно отметим, что есть команды, нарушающие содержимое B-регистра. Это, например, тригонометрические функции sin cos, tan и др. В таблице системы команд (см."Справочник...") все эти команды оговорены.

ОРГАНИЗАЦИЯ ЦИКЛОВ ВЫЧИСЛЕНИЙ

Аналогично программированию в машинном коде система команд калькулятора имеет команду djnz (DECREMENT and JUMP if NOT ZERO = УМЕНЬШИТЬ содержимое регистра на единицу и, если оно НЕ РАВНО НУЛЮ, ПЕРЕЙТИ на заданное количество байтов). Код этой команды 35. Также, как и команды jump и jump_true, она требует после

себя указания величины "смещения" s. Эта команда тоже двухбайтная.

Команда работает так. Когда она встречается, происходит уменьшение на 1 содержимого регистра B. Если оно не равно нулю, выполняется переход на заданную величину смещения, а если уже равно нулю, то последующий байт, задающий смещение, игнорируется и вычисления продолжаются в естественном порядке.

ПОМЕЩЕНИЕ ЧИСЕЛ НА СТЕК

Как Вы уже очевидно поняли, стек калькуллятора играет основополагающую роль при проведении вычислений. В этой ситуации совершенно необходимо иметь команду, с помощью которой можно было бы поместить нужное число на стек. Такая команда есть - stk_data (код 34). Это многобайтная инструкция. После кода операции должно идти само число, записанное в так называемой упакованной форме. Мы понимаем, что для начинающего читателя уже надоели бесконечные описания все новых и новых форм представления данных в компьютере, но ничего не поделаешь - таким его сделали создатели. Во всяком случае это последняя форма, с которой Вам приходится иметь дело, Вы теперь знаете (или можете знать) об этом все.

Правила перевода чисел из пятибайтной интегральной формы в упакованную достаточно неоднозначны и зависят от самого числа, т.е. для равных чисел эти правила различны. Мы приводим справочные таблицы для выполнения такого перевода.

УПАКОВАННАЯ ФОРМА ЦЕЛЫХ ЧИСЕЛ ОТ 0 ДО 255.

Если число представить в виде одного байта nn, то шестнадцатиричный код для помещения его на стек:

34 40 B0 00 nn

УПАКОВАННАЯ ФОРМА ЦЕЛЫХ ЧИСЕЛ ОТ 0 ДО 65535.

Если число представить в двухбайтной форме mm, nn, то код для помещения его на стек:

34 40 B0 nn mm

УПАКОВАННАЯ ФОРМА ОТРИЦАТЕЛЬНЫХ ЦЕЛЫХ ЧИСЕЛ ОТ -65535 ДО -1

Если представить в двухбайтной дополнительной двоичной

форме -1 как FFFF; -2 как FFFE и т.д., то код:

34 80 B0 FF nn mm

Если Вам трудно сразу вычислить шестнадцатиричный код отрицательного числа, то Вы можете делать так: прибавить к отрицательному числу 65535, затем результат переписать в шестнадцатиричной форме.

ПОМЕЩЕНИЕ НА СТЕК ПУСТОГО СТРИНГА

Для этого можно обойтись без команды stk_data соответственно, без упакованной формы. Калькулятор имеет команду const zero (код A0), которая помещает на стек целое число 0. Это эквивалент пустого стринга (хотя обратное утверждение справедливо вовсе не всегда).

УПАКОВАННАЯ ФОРМА СТРИНГОВ ДЛИНОЙ ДО 256 БАЙТОВ

Если текст Вашего стринга хранится, начиная с какого-то фиксированного адреса, например pp qq и имеет длину nn байтов, шестнадцатиричный код для его помещения на стек:

34 B0 qq pp nn

УПАКОВАННАЯ ФОРМА СТРИНГОВ ПРОИЗВОЛЬНОЙ ДЛИНЫ

Если адрес pp qq, а длина mm nn, то шестнадцатиричный код

34 B0 qq pp nn mm

ПРОИЗВОЛЬНЫЕ ДЕЙСТВИТЕЛЬНЫЕ ЧИСЛА

Здесь первым шагом является перевод числа в пятибайтную интегральную форму, с помощью которой можно представить действительные числа с плавающей точкой. О том, как это делается, мы писали выше. Предположим, что эта форма выглядит так:

aa ee dd cc bb

Здесь aa - это экспонента. Далее все зависит от ее величины. Если она находится в диапазоне от 51 до 8F, то упакованная форма определяется по таблице 5.13.6.1., а если aa выходит из этого диапазона, то по таблице 5.13.6.2.

В первом случае Вам надо определить, какая из четырех представленных в таблице форм Вам подходит и определить соответствующее ей значение aa". Во втором случае также надо в зависимости от величины мантиссы числа подобрать упакованную

форму.

Таблица 5.13.6.1.

Интегральная форма	Код калькулятора	Значение aa"
aa ee 00 00 00	34 aa" ee	aa-50
aa ee dd 00 00	34 aa" ee dd	aa-10
aa ee dd cc 00	34 aa" ee dd cc	aa+30
aa ee dd cc bb	34 aa" ee dd cc bb	aa+70

Таблица 5.13.6.2.

Интегральная форма	Код калькулятора	Значение aa"
aa ee 00 00 00	34 00 aa" ee	aa-50
aa ee dd 00 00	34 40 aa" ee dd	aa-50
aa ee dd cc 00	34 80 aa" ee dd cc	aa-50
aa ee dd cc bb	34 C0 aa" ee dd cc bb	aa+70

ОБЗОР ПРОЧИХ КОМАНД КАЛЬКУЛЯТОРА

Команда read_in предполагалась для ввода символа, поступающего с внешнего канала через подключенный к нему поток X, где X - число на вершине стека. Ее функция - двойная. Во-первых, поток выбирается в качестве текущего, а во-вторых, производится ввод символа, поступающего по этому потоку. Если ввод не происходит, то выдается пустой стринг. Функция была бы очень полезной для чтения символов, поступающих с клавиатуры (аналогично функции БЕЙСИКа INKEY$), но в ПЗУ "Спектрума" содержится нелепая и досадная ошибка. Она проявляет себя, когда X=0 или X=1, а оба эти потока уже назначены для канала "K" (клавиатура). Эта ошибка ведет себя так:

Сначала выбирается канал "K", что вызывает выключение пятого бита системной переменной FLAGS (23611), что означает "готовность к приему новой клавиши". В этом и состоит ошибка, потому что немедленно после этого выполняется попытка ввести символ из канала "K". Это делает подпрограмма, начинающаяся в ПЗУ

с адреса 10A8. Она проверяет пятый бит системной переменной FLAGS и немедленно выполняет возврат, если он выключен, что означает "символ поступил". Таким образом, INKEY$ #0 почти обязательно выдает пустой стринг. Единственное исключение, очень маловероятное, происходит когда между выключением пятого бита и его проверкой происходит системное прерывание. Только в этом случае нажатая клавиша будет прочитана. Так очень полезная функция стала бессмысленной из-за наличия в ПЗУ досадной ошибки.

Функция get_argt (код 39) вычисляет 2/PI*ASN(SIN(X)). Это довольно странная функция по причине своей бесполезности для пользователя, но к ней обращаются некоторые системные процедуры, "зашитые" в ПЗУ, при вычислении тригонометрических функций.

Интересна команда "trancate" (код 3A). Она служит для выделения целой части дробного числа. В этом смысле она несколько похожа на оператор БЕЙСИКа INT и команду калькулятора int (код 27), но если INT округляет дробь всегда вниз, то "trancate" отсекает дробную часть, т.е. округляет "к нулю". Пример сравнения дан в таблице.

X	int X	trancate X
5.3	5	5
5.8	5	5
-5.3	-6	-5
-5.8	-6	-5

Команда e_to_fp (код 3C) совершенно бесполезна потому, что содержит грубую ошибку, делающее невозможным ее использование из калькулятора.

По идее она предназначалась для того, чтобы переводить числа из нормализованной формы в форму с плавающей точкой, т.е. вычислять X*10^A, где X - действительное число, находящееся на вершине стека, а A - содержимое регистра A микропроцессора. Но при работе калькулятора регистр A занят другими вычислениями

и потому эта команда не дает правильного результата.

Если Вам необходимо выполнить такое действие, то Вы можете это сделать из машинного кода, не входя в калькулятор, по команде CALL 2D4F. Результат будет отправлен на вершину стека калькулятора.

Команду restack (код 3D) можно представить как противоположность int. Она преобразует целые числа в действительные.

00 00 04 00 00 - целое число 4

83 00 00 00 00 - действительное число 4

Эта команда переводит из одной формы в другую.

Калькулятор имеет еще целую группу команд, предназначенных для генерации полиномов Чебышева. Они применяются самим же калькулятором для вычисления алгебраических и тригонометрических фнукций. Дотошный пользователь, впрочем, может попробовать использовать их для создания процедур вычисления каких-то своих нужных ему функций.

На этом мы закончим рассмотрение работы с калькулятором по команде RST 28, но во второй части - "Практикум по программированию в машинных кодах Z-80" мы еще к нему вернемся и рассмотрим конкретные примеры применения калькулятора.

5.13.7. Команда RST 30.

Эта команда служит для создания (резервирования) свободной области памяти в рабочей области БЕЙСИКа. При работе в БЕЙСИКе эта процедура активно используется как системная. Без нее не обходятся такие операции как ввод строки, редактирование программы. Перед вызовом этой процедуры в регистровой паре BC должна содержаться длина резервируемой области памяти в байтах.

5.13.8. Команда RST 38.

Эта процедура служит для обработки маскируемых прерываний, о которых мы будем еще говорить немного позже. Во-первых, она выполняет наращивание системной переменной FRAMES (23672...23674), которая выполняет роль "внутренних часов"

"Спектрума" и, во-вторых, обеспечивает сканирование клавиатуры в поисках нажатой клавиши каждую пятидесятую долю секунды.

5.13.9. Замечания к использованию команд обращения к ПЗУ.

ПЗУ компьютера "Спектрум" содержит 16-ти килобайтную программу-монитор, обеспечивающую функционирование компьютера. Она включает в себя: интерпретатор БЕЙСИКа, программы, обеспечивающие ввод/вывод, связь с внешними устройствами, программу-калькулятор, различные таблицы, в том числе набор 96-ти символов и т.п.

Монитор состоит из сотен процедур, многими из которых Вы можете активно пользоваться, что очень сильно сокращает время, необходимое для разработки программ и уменьшает их размер. К использованию процедур, содержащихся в ПЗУ, фирмы, выпускающие компьютеры, практикуют различный подход.

В компьютерах, имеющих большую оперативную память типа "АМИГА", "АТАРИ СТ" и т.п. использование в программах пользователей обращений к системному ПЗУ считается дурным тоном. Разработчики компьютеров оставляют за собой право постоянно совершенствовать ПЗУ, а большой объем оперативной памяти всегда дает возможность программистам размещать необходимые им процедуры в ОЗУ. Если бы коммерческие программы имели обращения к ПЗУ, то многочисленные владельцы компьютеров с новыми моделями компьютеров не смогли бы ими пользоваться, т.е. использование в коммерческих программах многочисленных обращений к ПЗУ может серьезно нарушить совместимость программного обеспечения, если ПЗУ постоянно или периодически дорабатывается фирмой.

В Синклер-компьютерах реализован другой подход. Малый объем оперативной памяти предполагает не только возможность, но и необходимость самого широкого использования процедур ПЗУ в пользовательских программах. ПЗУ не только открыто для эксплуатации, более того, К.Синклер включил в систему команд команды группы RST, делающие этот доступ простым. Если посмотреть с таких позиций, то становится понятным, как и почему для компьютера с такой скромной памятью созданы тысячи изумительных программ, многие из которых по своей идеологии превосходят созданное для гораздо более могучих машин.

С другой стороны, здесь фирме пришлось заплатить за это отказом от доработки ПЗУ. Выше мы говорили о том, что ПЗУ имеет немало ошибок и теперь Вы должны понять, что это нельзя серьезно ставить в вину фирме. Фирма могла бы многократно доработать ПЗУ, как и поступают все фирмы, но страдали бы от этого потребители, т.к. от этого нарушилась бы совместимость компьютеров и программного обеспечения.

Конечно, от программиста это требует определенной внимательности, зато миллионы простых пользователей по всему миру очень выиграли от скромности К.Синклера. Это еще раз напоминает нам о том, что Синклер-компьютеры очень "дружественны" к потребителю, что и обусловило их невероятную популярность.

Справедливости ради, надо сказать, что есть еще третий путь, который например был реализован в компьютерах "Коммодор". Здесь фирма тоже закрывает ПЗУ и не рекомендует использовать его процедуры напрямую, периодически внося в них изменения. Но зато она выделила небольшой участок ПЗУ в качестве своеобразного диспетчера. Этот блок называется "керналь". В нем хранятся адреса основных процедур ПЗУ. Поэтому если при доработке ПЗУ что-то изменяется в адресах процедур, эти изменения тут же вносятся в "керналь", который всегда находится в одном и том же месте. Так обеспечивается совместимость программного обеспечения между старыми и новыми моделями. Программы пользователя не входят в ПЗУ напрямую, а входят в "керналь", откуда их переправляют туда, куда надо. К сожалению, в "Спектруме" такого блока нет.

"ИНФОРКОМ" получает массу писем с изложением новых и все более изощренных способов доработки ПЗУ. До сих пор многие энтузиасты прилагают свою энергию для русификации компьютера. Каждый делает это так, как ему заблагорассудится - стандартов нет, да они вряд ли и будут. Помните, что такая "доработка" серьезно нарушает совместимость компьютеров. Вы не сможете воспользоваться чужой программой, созданной на "русифицированном" компьютере, отличном от Вашего. Разработанные Вами русскоязычные программы также будут иметь коммерческую ценность близкую к нулевой, если при русификации Вы опирались на свое измененное ПЗУ. Помните, что всегда есть возможность русификации при размещении русского шрифта в оперативной памяти. Мы об этом неод-

нократно писали в разработке "Большие возможности Вашего "Спектрума" и в выпусках "ZX-РЕВЮ". Практикуют и другие доработки (например, встраивают в ПЗУ программы обслуживания внешних портов). Мы понимаем, что остановить творческий поиск невозможно, но по крайней мере не позволяйте никому брать дополнительную плату за "доработки" ПЗУ и выдавать их за преимущества. Относитесь к ним критически.

Желающим действительно сделать ценные разработки рекомендуем делать это путем подключения замещающего ПЗУ ("ТЕНЕВОГО"), как это и делается в наиболее практичных периферийных устройствах.

5.14. КОМАНДЫ СДВИГА И РОТАЦИИ БИТОВ

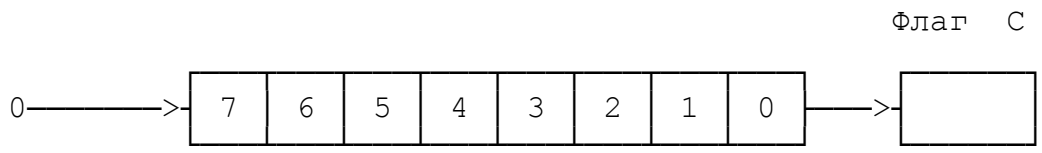
Команды этой группы работают с байтами данных, содержащимися в регистрах процессора, путем перемещения битов в этих байтах. Способ перемещения битов и определяет название и назначение команды.

5.14.1. Команды сдвига.

Команды сдвига выполняют перемещения битов в байте на соседнее место влево или вправо. Существуют два типа сдвигов - сдвиг логический и сдвиг арифметический. При логическом сдвиге байт рассматривается как набор из восьми двоичных знаков (0 или 1). При арифметическом сдвиге содержимое рассматривается как число со знаком.

КОМАНДА SRL.

Это сокращение от SHIFT RIGHT LOGICAL (ЛОГИЧЕСКИЙ СДВИГ ВПРАВО). Структурная схема:



По этой команде каждый бит смещается вправо на соседнее место. Нулевой бит поступает в регистр F, где включается флаг переноса, если бит 0 был включен и выключается, если бит 0 был

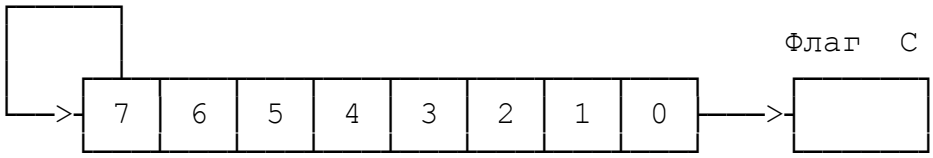
выключен. В старший (седьмой бит) засылается 0. т.е. он выключается.

Команд с мнемоникой SRL всего 10. Их коды зависят от того, к каким регистрам они применяются.

Мнемоника	Код	Мнемоника	Код
SRL A	CB 3F	SRL H	CB 3C
SRL L	CB 3D	SRL B	CB 38
SRL C	CB 39	SRL D	CB 3A
SRL E	CB 3B	SRL (HL)	CB 3E
SRL (IX+S)	DD CB S 3E	SRL (IY+S)	FD CB S 3E

КОМАНДА SRA.

Это сокращение от SHIFT RIGHT ARITHMETIC (АРИФМЕТИЧЕСКИЙ СДВИГ ВПРАВО). Структурная схема:

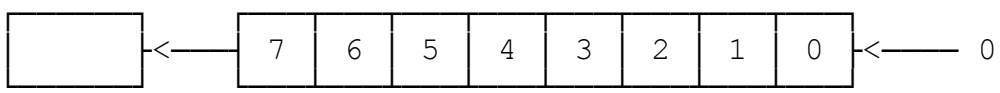


Команда очень похожа на команду SRL. Единственным отличием является то, что бит 7 устанавливается в то же положение, в котором он был до сдвига. Это означает, что если в байте содержится целое число со знаком, то знак в результате этой операции не изменится. Внимательно приглядевшись к этой операции, Вы увидите, что сдвиг вправо эквивалентен делению содержимого байта пополам. При этом если число было нечетное (бит 0 включен), то образуется остаток от деления, о чем свидетельствует включение флага C регистра F.

Мнемоника	Код	Мнемоника	Код
SRA A	CB 2F	SRA H	CB 2C
SRA L	CB 2D	SRA B	CB 28
SRA C	CB 29	SRA D	CB 2A
SRA E	CB 2B	SRA (HL)	CB 2E
SRA (IX+S)	DD CB S 2E	SRA (IY+S)	FD CB S 2E

КОМАНДА SLA

Это арифметический сдвиг влево. Структурная схема:



По этой команде все биты сдвигаются на одну позицию влево. В бит 0 (младший) засылается 0. Бит 7 сдвигается во флаг переноса. Операция эквивалентна умножению байта на два. Если при этом образуется число большее, чем 255. включается флаг C.

Поскольку при сдвиге влево арифметический и логический сдвиги совпадают, то для последнего не выделено своей команды.

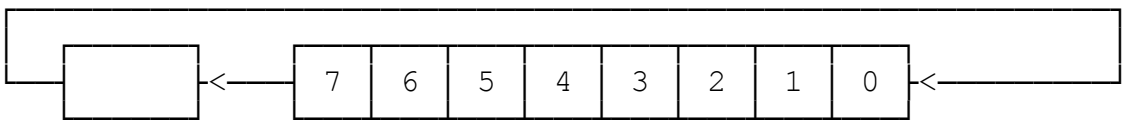
Мнемоника	Код	Мнемоника	Код
SLA A	CB 27	SLA H	CB 24
SLA L	CB 25	SLA B	CB 20
SLA C	CB 21	SLA D	CB 22
SLA E	CB 23	SLA (HL)	CB 26
SLA (IX+S)	DD CB S 26	SLA (IY+S)	FD CB S 26

5.14.2. Команды ротации.

Команды ротации очень похожи на команды сдвига. Разница состоит в том, что здесь тот бит, который выходит за пределы байта, поступает в байт с противоположной стороны. Более подробно эти операции показаны на структурных схемах.

КОМАНДЫ RL

Ротация влево. Структурная схема:

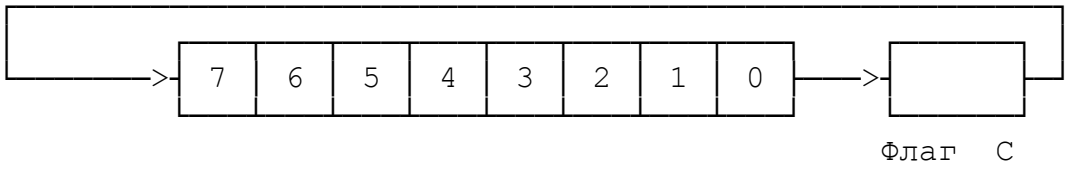


Бит 7 перемещается во флаг переноса, а флаг переноса - в бит 0.

Мнемоника	Код	Мнемоника	Код
RL A	CB 17	RL H	CB 14
RL L	CB 15	RL B	CB 10
RL C	CB 11	RL D	CB 12
RL E	CB 13	RL (HL)	CB 16
RL (IX+S)	DD CB S 16	RL (IY+S)	FD CB S 16

КОМАНДЫ RR

Ротация вправо Структурная схема:

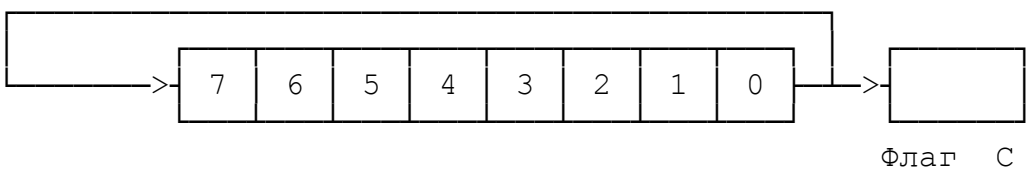


Бит 0 перемещается во флаг переноса, а флаг переноса - в бит 7.

Мнемоника	Код	Мнемоника	Код
RR A	CB 1F	RR H	CB 1C
RR L	CB 1D	RR B	CB 18
RR C	CB 19	RR D	CB 1A
RR E	CB 1B	RR (HL)	CB 1E
RR (IX+S)	DD CB S 1E	RR (IY+S)	FD CB S 1E

КОМАНДЫ RRC

Эта команды отличаются от команд RR тем, что флаг переноса не вовлекается в ротацию, хотя результат операции на него влияет. Суть операции ясна из структурной схемы.

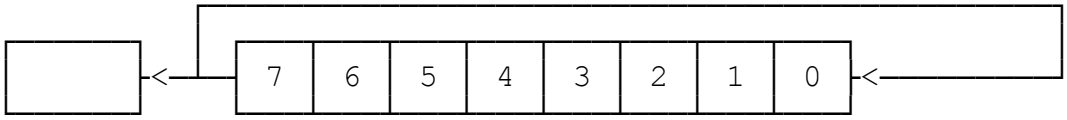


Мнемоника	Код	Мнемоника	Код
RRC A	CB 0F	RRC H	CB 0C
RRC L	CB 0D	RRC B	CB 08
RRC C	CB 09	RRC D	CB 0A
RRC E	CB 0B	RRC (HL)	CB 0E
RRC (IX+S)	DD CB S 0E	RRC (IY+S)	FD CB S 0E

КОМАНДЫ RLC

~~~~~~~~~~~~

Эта команды отличаются от команд RL тем, что флаг переноса не вовлекается в ротацию, хотя результат операции на него влияет. Суть операции ясна из структурной схемы.



Флаг C

Мнемоника	Код	Мнемоника	Код
RLC A	CB 07	RLC H	CB 04
RLC L	CB 05	RLC B	CB 00
RLC C	CB 01	RLC D	CB 02
RLC E	CB 03	RLC (HL)	CB 06
RLC (IX+S)	DD CB S 06	RLC (IY+S)	FD CB S 06

5.14.3. Однобайтные команды ротации аккумулятора.

~~~~~~

Поскольку регистр A широко используется во многих программах, для него продублированы 4 команды ротации. Они выполнены как однобайтные. Это позволяет при их использовании экономить память.

Мнемоника	Код	Примечание.
RLA	17	= RL A
RRA	1F	= RR A
RLCA	07	= RLC A
RRCA	0F	= RRC A

5.14.4. Команды ротации полубайтов.

В этой подгруппе только две команды. Они применяются для работы, когда используется двоичная форма записи десятиричных чисел (BCD-арифметика). Напомним, что здесь один байт может представлять два десятиричных разряда (числа от 0 до 99), а каждый полубайт (4 бита) может представлять число от 0 до 9. Значения от A до F - не используются. Перенос в старший полубайт выполняется уже тогда, когда младший полубайт принимает значение, большее 9 (в двоичной форме 1001).

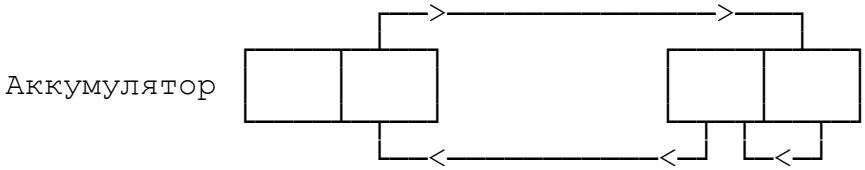
В каждой из этих операций участвуют по одному младшему полубайту регистра A и по одному полному байту, взятому из ячейки памяти, на адрес которой указывает содержимое регистровой пары HL процессора.

В отличие от команд ротации битов эти команды перемещают сразу целый полубайт, не меняя взаимного расположения битов.

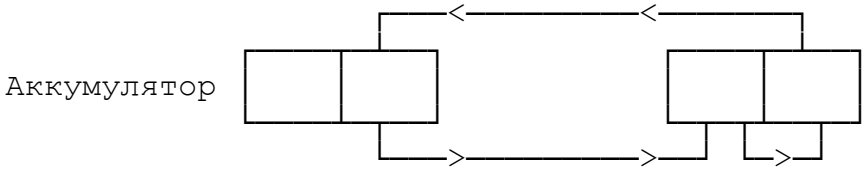
Мнемоники этих команд - RLD и RRD.

Мнемоника	Код	Мнемоника	Код
RLD	ED 6F	RRD	ED 67

Структурная схема команды RLD.



Структурная схема команды RRD.



Интересно отметить, что старший полубайт в аккумуляторе, как Вы видите, этими командами не затрагивается. Кроме того,

как Вы видите из структурных схем, эти команды различаются только направлением ротации - влево или вправо, причем понятия "влево" и "вправо" относятся не к аккумулятору, а к той ячейке памяти, на которую указывает содержимое HL.

Влияние команд сдвига и ротации на флаговый регистр Вы можете установить по таблицам "Справочника...", но основная суть такова:

1) все команды, кроме RRD И RLD влияют на флаг переноса.

2) все команды, кроме RLA, RRA, RLCA, RRCA влияют на флаг четности.

5.15. КОМАНДЫ ДЛЯ РАБОТЫ С БИТАМИ

Процессор имеет обширный набор команд для работы с отдельными битами. Этими командами биты могут включаться (SET), выключаться (RES) и проверяться (BIT).

Надо сказать, что для стандартного БЕЙСИКа операции с битами вообще невозможны, хотя существуют некоторые расширенные версии, предоставляющие такую возможность.

5.15.1. Команды включения битов.

Эти команды применяются программистом, когда надо обеспечить гарантированное включение какого-либо бита в одиночном регистре или в заданной ячейке памяти. Все команды этой подгруппы начинаются с мнемоники SET. Примеры применения команд:

SET 0,A - включить нулевой бит аккумулятора.

SET 4,(HL) - включить четвертый бит в байте, адрес которого находится в регистровой паре HL.

После включения бит принимает значение 1.

Коды команд включения битов приведены в таблице 5.15.1.1.

5.15.2. Команды выключения битов.

Команды служат для гарантированного выключения каких-либо битов в одиночном регистре или в заданной ячейке памяти. Команды этой подгруппы начинаются с мнемоники RES. После выключения

нужный бит равен нулю. Машинные коды команд выключения битов приведены в табл.5.15.2.1.

Команды включения. Таблица 5.15.1.1

	SET 0	SET 1	SET 2	SET 3	SET 4	SET 5	SET 6	SET 7
A	CB C7	CB CF	CB D7	CB DF	CB E7	CB EF	CB F7	CB FF
H	CB C4	CB CC	CB D4	CB DC	CB E4	CB EC	CB F4	CB FC
L	CB C5	CB CD	CB D5	CB DD	CB E5	CB ED	CB F5	CB FD
B	CB C0	CB C8	CB D0	CB D8	CB E0	CB E8	CB F0	CB F8
C	CB C1	CB C9	CB D1	CB D9	CB E1	CB E9	CB F1	CB F9
D	CB C2	CB CA	CB D2	CB DA	CB E2	CB EA	CB F2	CB FA
E	CB E3	CB CB	CB D3	CB DB	CB E3	CB EB	CB F3	CB FB
(HL)	CB C6	CB CE	CB D6	CB DE	CB E6	CB EE	CB F6	CB FE
(IX+S)	DD CB S C6	DD CB S CE	DD CB S D6	DD CB S DE	DD CB S E6	DD CB S EE	DD CB S F6	DD CB S FE
(IY+S)	FD CB S C6	FD CB S CE	FD CB S D6	FD CB S DE	FD CB S E6	FD CB S EE	FD CB S F6	FD CB S FE

5.15.3. Команды проверки битов.

Команды служат для проверки того, является заданный бит включенным или выключенным. Обычно за этой командой следует условный переход, вызов подпрограммы или возврат по флагу нуля. Команды этой подгруппы начинаются с мнемоники BIT.

В результате действия этой команды включается или выключается флаг нуля (Z-флаг регистра F).

Машинные коды команд этой группы представлены в табл .5.15.3.1.

Команды выключения. Таблица 5.15.2.1

	RES 0	RES 1	RES 2	RES 3	RES 4	RES 5	RES 6	RES 7
A	CB 87	CB 8F	CB 97	CB 9F	CB A7	CB AF	CB B7	CB BF
H	CB 84	CB 8C	CB 94	CB 9C	CB A4	CB AC	CB B4	CB BC
L	CB 85	CB 8D	CB 95	CB 9D	CB A5	CB AD	CB B5	CB BD
B	CB 80	CB 88	CB 90	CB 98	CB A0	CB A8	CB B0	CB B8
C	CB 81	CB 89	CB 91	CB 99	CB A1	CB A9	CB B1	CB B9
D	CB 82	CB 8A	CB 92	CB 9A	CB A2	CB AA	CB B2	CB BA
E	CB 83	CB 8B	CB 93	CB 9B	CB A3	CB AB	CB B3	CB BB
(HL)	CB 86	CB 8E	CB 96	CB 9E	CB A6	CB AE	CB B6	CB BE
(IX+S)	DD CB S 86	DD CB S 8E	DD CB S 96	DD CB S 9E	DD CB S A6	DD CB S AE	DD CB S B6	DD CB S BE
(IY+S)	FD CB S 86	FD CB S 8E	FD CB S 96	FD CB S 9E	FD CB S A6	FD CB S AE	FD CB S B6	FD CB S BE

Команды проверки. Таблица 5.15.3.1

	BIT 0	BIT 1	BIT 2	BIT 3	BIT 4	BIT 5	BIT 6	BIT 7
A	CB 47	CB 4F	CB 57	CB 5F	CB 67	CB 6F	CB 77	CB 7F
H	CB 44	CB 4C	CB 54	CB 5C	CB 64	CB 6C	CB 74	CB 7C
L	CB 45	CB 4D	CB 55	CB 5D	CB 65	CB 6D	CB 75	CB 7D

продолжение таблицы 5.15.3.1

	BIT 0	BIT 1	BIT 2	BIT 3	BIT 4	BIT 5	BIT 6	BIT 7
B	CB 40	CB 48	CB 50	CB 58	CB 60	CB 68	CB 70	CB 78
C	CB 41	CB 49	CB 51	CB 59	CB 61	CB 69	CB 71	CB 79
D	CB 42	CB 4A	CB 52	CB 5A	CB 62	CB 6A	CB 72	CB 7A
E	CB 43	CB 4B	CB 53	CB 5B	CB 63	CB 6B	CB 73	CB 7B
(HL)	CB 46	CB 4E	CB 56	CB 5E	CB 66	CB 6E	CB 76	CB 7E
(IX+S)	DD CB S 46	DD CB S 4E	DD CB S 56	DD CB S 5E	DD CB S 66	DD CB S 6E	DD CB S 76	DD CB S 7E
(IY+S)	FD CB S 46	FD CB S 4E	FD CB S 56	FD CB S 5E	FD CB S 66	FD CB S 6E	FD CB S 76	FD CB S 7E

5.16. КОМАНДЫ ОБРАБОТКИ БЛОКОВ ПАМЯТИ

Это очень мощные команды. Они предназначены для работы с целыми частями памяти. В этой группе 8 команд. Четыре команды предназначены для перемещения блоков памяти из одного места в другое и четыре команды для поиска заданного байта в заданной области.

5.16.1. Команды перемещения блоков.

В этой подгруппе 4 команды: LDIR, LDI, LDDR, LDD.

КОМАНДА LDIR

Ее код ED B0, т.е. это двухбайтная команда. Служит для перемещения блоков памяти. Для ее работы необходимо, чтобы:

- в регистровой паре HL находился адрес начала перемещае-

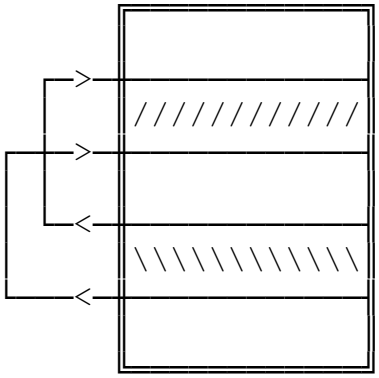
мого блока;

- в регистровой паре DE находился адрес того места, куда должен перемещаться блок;

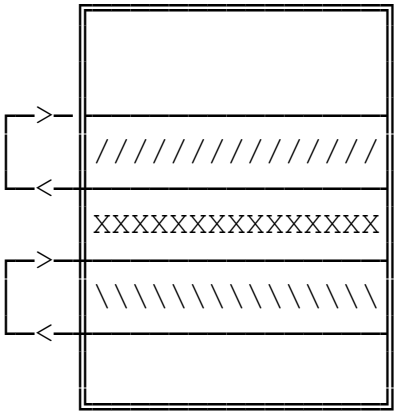
- в паре BC находилась длина перемещаемого блока.

Команда может перемещать блоки в памяти как сверху вниз, так и снизу вверх, но если перемещение памяти происходит с частичным наложением нового блока на старый, то тогда возможно перемещение только сверху вниз, иначе пересылаемые байты запортят те, которые еще не были пересланы.

Перемещение без наложения блоков



Перемещение блоков с наложением



КОМАНДА LDDR

Ее код ED B8, а назначение - то же, что и у команды LDIR. Регистры HL, DE, BC - выполняют те же функции и должны быть выставлены так же, как и для команды LDIR.

Разница состоит в том, что если по команде LDIR сначала пересылается первый байт, затем второй и т.д., то по команде LDDR - сначала последний, затем предпоследний и т.д. Таким образом, если перемещение идет с наложением нового блока на старый, то этой командой можно пересылать блоки снизу вверх.

КОМАНДА LDI

Ее код ED A0. Действие этой команды такое же, как и у команды LDIR, но если команда LDIR выполняет перемещение блока целиком байт за байтом до тех пор, пока в регистре BC не будет достигнут ноль, то команда LDI перемещает только один байт. Если при этом в BC не достигнут ноль, то флаг P/V равен 1, если достигнут, то флаг P/V равен нулю. По результатам проверки этого флага программист может принять решение о продолжении действия команды LDI или прекращении. Одним словом, если команда LDIR выполняет перенос каждого последующего байта автоматически, то LDI - неавтоматически.

КОМАНДА LDD

Ее код ED A8. Это такой же неавтоматический аналог LDDR, как LDI - аналог LDIR.

Время выполнения команд LDD и LDI - фиксированное и занимает 16 тактов микропроцессора. Время же действия автоматических команд LDIR и LDDR - неопределено, так как оно зависит от размера блока, подлежащего перемещению. Чем он больше, тем больше и время, но его можно оценить - это 21 такт на перемещение каждого байта, кроме последнего, на который потребны 16 тактов. Можно оценить, например, время на переброску экрана, т.е. 6912 байтов: 6911*21 + 16 = 145147 тактов, что при частоте 3.5 МГц составляет примерно 0.04 сек.

5.16.2. Команды поиска.

В этой подгруппе 4 команды: CPIR, CPDR, CPI и CPD.

КОМАНДА CPIR

Ее код - ED B1.

Команда просматривает заданную область памяти в поисках первого встреченного места нахождения заданной величины. Заданный байт должен быть установлен заранее в регистре A. В регистровой паре HL устанавливается начальный адрес, с которого

начинается поиск, а в регистре BC - длина блока, подлежащего проверке.

Команда исполняется до тех пор, пока либо в какой-либо ячейке не будет найден байт, равный содержимому регистра A, либо пока не будет исчерпана вся заданная область и в регистре BC не останется ноль. Многократное повторение команды выполняется автоматически. Поиск ведется снизу вверх. Если в результате поиска нужный байт найден, то действие команды прекращается и в регистре HL Вы можете найти искомый адрес, при этом включается флаг нуля (Z) и выключается флаг знака (S).

КОМАНДА CPDR

Ее код - ED B9.

Действие аналогично команде CPIR, но здесь блок просматривается сверху вниз. Эта команда также автоматическая.

КОМАНДА CPI

Ее код - ED A1.

Команда аналогична CPIR, но в отличие от нее не является автоматической. Здесь после каждого сравнения очередного байта программист может принять решение о прекращении или продолжении поиска в зависимости от состояния флагов регистра F.

Если содержимое проверяемой ячейки и аккумулятора совпадут, включается флаг нуля, в противном случае он выключен. Если в процессе поиска нужный байт еще не найден, а содержимое BC уменьшилось до нуля, то включается флаг переполнения/четности (P/V), в противном случае он включен.

Поиск по команде CPI выполняется снизу вверх, как и по команде CPIR.

КОМАНДА CPD

Ее код ED 9.

Действие команды аналогично командам CPDR и CPI, но в отличие от CPDR она неавтоматическая, а в отличие от CPI поиск по блоку памяти производится не снизу вверх, а сверху вниз.

5.17. КОМАНДЫ ДЛЯ РАБОТЫ С ВНЕШНИМИ УСТРОЙСТВАМИ

Команды этой обширной группы позволяют процессору получать данные от внешних (периферийных) устройств и выдавать данные на эти устройства точно так же, как он может загружать в регистры данные из ячеек оперативной памяти и отправлять их туда на хранение.

Обратите внимание на то, что по отношению к процессору Z-80 такие части компьютерной системы как клавиатура, магнитофон, звуковой динамик тоже являются внешними.

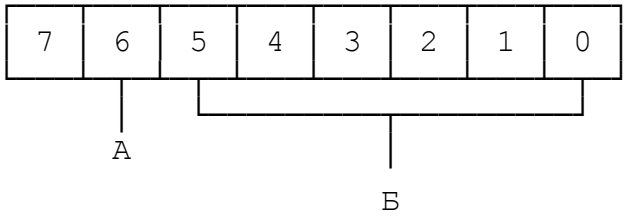
Команды этой группы начитаются с мнемоник IN (ввод) и OUT (вывод). В какой-то степени они аналогичны командам БЕЙСИКа IN и OUT.

Данные, которые передаются по командам IN и OUT являются 8-битными. Во время приема активизируются физические линии RD и IORQ, а во время выдачи - также IORQ и WR. Кроме этих линий активизируется также адресная шина. Адрес, помещенный на ней, задает адрес внешнего порта. Всего возможны 65536 адресов портов. Различные периферийные устройства активизируются состоянием различных адресных линий адресной шины.

Так, например, порт 254 (FE) служит для связи процессора с клавиатурой, магнитофоном, звуковым динамиком и телеэкраном.

Для примера рассмотрим содержимое байта данных, поступающих или выдаваемых по этому порту.

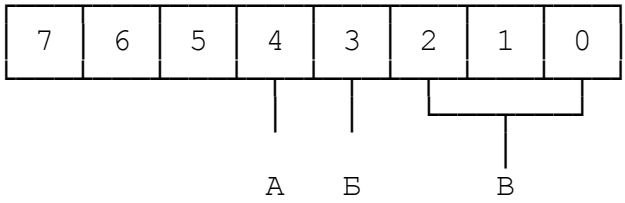
ПОРТ FE (ВВОД)



А. Шестой бит указывает на наличие сигнала на магнитофонном разъеме EAR (вход в компьютер). 1 - нет сигнала, 0 есть сигнал.

Б. Младшие пять битов определяют какая из пяти клавиш каждого полуряда клавиатуры была нажата. Бит равен нулю, если клавиша была нажата и 1, если нет.

ПОРТ FE (ВЫВОД)



А. По четвертому биту выдается сигнал на звуковой динамик.

Б. По третьему биту выдается сигнал на разъем MIC (выход на магнитофон).

В. По младшим трем битам выдается сигнал на установку цвета бордюра (один из возможных восьми цветов BORDER).

Для тех, кто не имеет книги Виккерса, напомним, как задаются адреса портов при опросе клавиатуры.

1. Мы указали, что 16-разрядная адресная шина (от A0 до A15) указывает на адрес внешнего порта.

2. Порт FE задается состоянием адресных линий A0...A7. При этом A0 - включена, A1...A7 - выключены:

1111 1110

Об остальных адресных линиях A8...A15 мы пока не говорим.

3. В то же время, мы указали, что клавиши полуряда опрашиваются по пяти битам данных, поступающим с этого порта (D0...D5).

4. При этом адресные линии A8...A15 могут указывать о каком именно полуряде идет речь.

Клавиатура "Спектрума" имеет 40 основных клавиш. Они разбиты на 8 полурядов по 5 клавиш в каждом.

1.....5	6.....0
Q.....T	Y.....P
A.....G	H.....ENTER
C.SHIFT.....V	B.....SPACE

Этим полурядам соответствует следующее состояние адресных линий A8...A15.

1111 0111 = 247	1110 1111 = 239
1111 1011 = 251	1101 1111 = 223
1111 1101 = 253	1011 1111 = 191
1111 1110 = 254	0111 1111 = 127

Таким образом, для опроса произвольного полуряда адрес его порта равен:

```
  256*(255-2^n)  + 254
  └──────┬──────┘   └┬┘
         │           │
        A8...A15    A0...A7
```

Здесь n - номер полуряда.

Например, опросить клавиши от 1 до 5 можно следующей командой из Бейсика:

```
PRINT IN (256*(255-2^3)+254)   или проще:
PRINT IN 63486
```

Младшие пять битов того байта, который поступает с этого порта, указывают, какая именно клавиша была нажата. При этом следует помнить:

- самый младший бит соответствует внешней клавише полуряда;

- при нажатой клавише соответствующий ей бит выключается, т.е. если ни одна клавиша не была нажата, то все пять младших битов включены, и Вы получите 255.

Среди команд ввода/вывода есть команды простого (однобайтного) ввода/вывода и есть команды блочного ввода/вывода, которые бывают как автоматическими, так и неавтоматическими. Далее мы рассмотрим все эти команды по подгруппам.

5.17.1. Команды простого ввода.

Мнемоника	Код	Комментарий
IN A,(N)	DB N	Поместить в регистр A то число, которое поступает с порта N.
IN A,(C)	ED 78	Поместить в регистр A то число, которое поступает с порта, номер которого находится в регистре C.
IN H,(C)	ED 60	То же, но в регистр H.
IN L,(C)	ED 68	То же, но в регистр L.
IN B,(C)	ED 40	То же, но в регистр B.
IN C,(C)	ED 48	То же, но в регистр C.
IN D,(C)	ED 50	То же, но в регистр D.
IN E,(C)	ED 58	То же, но в регистр E.
IN F,(C)	ED 70	Служит для выставления флагов в регистре F без изменения содержимого числовых регистров.

По командам этой подгруппы необходимо сделать ряд примечаний. Мы говорили о том, что адрес порта может задаваться 16-битным числом, находящимся на адресной шине. В то же время, во всех этих командах адрес порта задается однобайтным числом, т.е. определяется состоянием адресных линий A0...A7. В тех случаях, когда для функционирования внешних устройства этого достаточно, линии A8...A15 могут быть проигнорированы, но как быть, когда этого недостаточно, например при вводе с заданного полуряда клавиатуры, о чем мы только что писали?

Для команды IN A,(N) биты адреса порта A0...A7 задаются числом N, следующим за кодом операции, а биты адреса A8...A15 должны быть предварительно выставлены в самом же регистре A.

Для команд IN A,(C) и других подобных биты адреса A0...A7 задаются содержимым регистра C, а биты A8...A15 должны быть предварительно помещены в регистр B.

5.17.2. Команды простого вывода.

Мнемоника	Код	Комментарий
OUT (N),A	D3 N	Выдать содержимое аккумулятора по порту N. Если адрес порта - двухбайтная величина, то старший байт адреса должен быть предварительно помещен в регистр A.
OUT (C),A	ED 79	Передать содержимое аккумулятора на внешний порт, номер которого содержится в регистре C. Если адрес порта - двухбайтная величина, то старший байт должен быть предварительно помещен в регистр B.
OUT (C),H	ED 61	То же, но для регистра H.
OUT (C),L	ED 69	То же, но для регистра L.
OUT (C),B	ED 41	То же, но для регистра B.
OUT (C),C	ED 49	То же, но для регистра C.
OUT (C),D	ED 51	То же, но для регистра D.
OUT (C),E	ED 59	То же, но для регистра E.

5.17.3. Команды блочного ввода/вывода.

Эти команды служат для организации ввода/вывода сразу целых блоков информации. Здесь так же, как и для простых команд ввода/вывода, младший байт адреса порта предварительно помещается в регистр C. Если нужен и старший байт, то он располагается в регистре B. В регистре HL размещается начальный адрес, в который начинается загрузка или из которого начинается выгрузка (аналогично командам перемещения блоков). В регистре B находится количество байтов, подлежащих вводу/выводу.

Среди этих команд есть автоматические, выполняющие сразу прием и размещение целого блока (аналогично LDIR и LDDR) и неавтоматические, выполняющие прием по одному байту (далее по результатам проверки флага нуля, который включается, если в регистре B оказывается в результате операции 0, программист может принять решение продолжить ввод или прервать его). Неавтоматические команды аналогичны LDI и LDD.

Сразу отметим, что автоматические команды ввода/вывода имеют для "Спектрума" очень ограниченное применение, т.к. необ-

ходимо, чтобы внешнее периферийное устройство оперировало с той же скоростью, что и сам Z-80, а к большинству употребляемых устройств это не относится (принтер, магнитофон, клавиатура, интерфейс джойстика и т.д.).

Мнемоника	Код	Комментарий
INIR	ED B2	Автоматическая команда блочного ввода. Первый поступающий байт направляется туда, куда указывает адрес, находящийся в регистре HL. Следующий байт размещается за ним и так далее снизу вверх.
INDR	ED BA	То же, но размещение байтов в памяти идет сверху вниз.
OTIR	ED B3	То же, что и INIR, но не для ввода, а для вывода.
OTDR	ED BB	То же, что и INDR, но не для ввода, а для вывода.
INI	ED A2	То же, что и INIR, но неавтоматическая
IND	ED AA	То же, что и INDR, но неавтоматическая
OUTI	ED A3	То же, что и OTIR, но неавтоматическая
OUTD	ED AB	То же, что и OTDR, но неавтоматическая

5.18. КОМАНДЫ ПРЕРЫВАНИЙ

Команды прерываний позволяют с помощью внешних сигналов прерывать последовательность исполнения команд в процессоре. Процессор Z-80 имеет три режима прерываний - режимы 0,1 и 2. Сразу оговоримся, что не все они реализованы в "Спектруме", хотя мы осветим все.

Когда процессор получает прерывание, то если оно разрешено, нормальный ход исполнения программы прерывается и управление передается в специальную программу обработки прерываний. Место расположения этой обрабатывающей программы зависит от того, какой режим прерывания включен. Программа обработки прерывания прекращает свою работу, когда встречает специальную команду возврата, после чего управление вновь передается той программе, которая была прервана.

В этой группе всего 7 команд и мы последовательно их рассмотрим.

Команда EI.

~~~~~~~~~~

Код FB. Назначение - разрешение прерывания. После прохождения этой команды обработка прерываний разрешена.

Команда DI.

~~~~~~~~~~

Код - F3. Назначение - запрет прерываний. Она запрещает прерывания до тех пор, пока не пройдет команда EI.

Команда IM0.

~~~~~~~~~~

Код - ED 46. Назначение - включение прерываний в режиме 0. О режимах прерываний см. несколько ниже.

Команда IM1.

~~~~~~~~~~

Код - ED 56. Назначение - включение режима прерываний первого типа (см. ниже).

Команда IM2.

~~~~~~~~~~

Код - ED 5E. Назначение - включение режима прерываний второго рода (см. ниже).

Команда RET I.

~~~~~~~~~~

Код - ED 4D. Эта команда аналогична команде RET (возврат после вызова подпрограммы), но выполняет двойную функцию:

- возврат из подпрограммы, обслуживающей маскируемое прерывание;

- разрешение маскированного прерывания (ведь во время работы обслуживающей программы оно запрещалось, иначе новое прерывание могло бы прервать обработку прерывания и т.д.).

Команда RET N.

~~~~~~~~~~

Код - ED 45. Она аналогична команде RET I, но обеспечивает
~~~~~~~~~~

возврат при выходе из обработки немаскированного прерывания.

ПРЕРЫВАНИЯ В "СПЕКТРУМЕ" И ИХ ИСПОЛЬЗОВАНИЕ

По своей архитектуре микропроцессор Z-80 имеет возможность обработки двух видов прерываний - маскируемых и немаскируемых. Маскируемые прерывания делятся на три типа - прерывания типа 0, 1 и 2. Тип маскируемого прерывания устанавливается с помощью команд АССЕМБЛЕРА:

IM 0 - код ED 46H
IM 1 - код ED 56H
IM 2 - код ED 5EH.

IM - аббревиатура от английского INTERRUPT MODE - тип прерывания.

Z-80 имеет два входа для сигналов запроса прерываний - входы "INT" и "NMI".

На входе линии запроса прерывания "INT" (16 ножка микросхемы) внешнее устройство формирует сигнал низкого уровня, обращая "внимание" микропроцессора на этот сигнал. В СПЕКТРУМЕ сигнал "INT" формируется 50 раз в секунду. Если флаг прерывания сброшен (лог."0"), запрос воспринимается и микропроцессор прерывает обычную обработку и переходит к выполнению необходимой процедуры прерывания. Реакция на запрос прерывания определяется флагом прерывания, поэтому прерывание может быть замаскировано. В программу вводятся команды, которые устанавливают или сбрасывают флажок прерывания и, следовательно, разрешают или запрещают его обработку. В Z-80 таких команд две:

"DI" (DISABLE INTERRUPTS) - запрещение прерываний.

"EI" (ENABLE INTERRUPTS) - разрешение прерываний.

Такой способ позволяет гибко реагировать на прерывания - мы сами определяем, воспринимать их или игнорировать.

Однако, в некоторых ситуациях требуется, чтобы прерывания обслуживались незавимо от текущих действий микропроцессора. Для этих целей предусмотрена отдельная линия немаскируемого прерывания "NMI".Когда на ней появляется сигнал низкого уровня, то по фронту этого сигнала выполнение программы прерывается независимо от состояния флага прерываний, т.е. независимо от того,

разрешены прерывания или запрещены. При этом микропроцессор Z-80 начинает обработку процедуры с адреса 0066H.

К сожалению, в ПЗУ "СПЕКТРУМА" в процедуре обработки немаскируемого прерывания "NMI" содержится грубая ошибка (кто знает - случайна ли она?), благодаря которой обработка "NMI" или приводит к рестарту компьютера с нулевого адреса, или не дает никаких результатов, производя обычный возврат в точку, откуда было вызвано прерывание.

Однако, если Вы работаете или в дальнейшем собираетесь работать с видоизменным или теневым ПЗУ, то можете с адреса 0066H разместить свою программу обработки "NMI" или исправить ошибку в стандартной программе. Для этого необходимо по адресу 006DH заменить код 020H (JP NZ,) на код 028H (JP Z,). В этом случае программа будет проверять системную переменную "NMIADD" (адреса 23728 и 23729 - 5CB0H и 5CB1H) и, если там не ноль, то будет произведен переход по адресу, содержащемся в этой переменной. Таким образом, Вы сможете расположить свою программу обработки "NMI" по любому необходимому адресу.

Если Вы хотите работать с "NMI", Вам необходимо будет произвести небольшую доработку компьютера. Дело в том, что на вход "NMI" (17 ножка) Z-80 в "СПЕКТРУМЕ" постоянно подано +5В (если, конечно, у Вас не подключен контроллер дисковода, где этот вход уже используется кнопкой "MAGIC"). Вам необходимо будет отрезать вход "NMI" от +5В и подключить его к своему внешнему устройству (в самом простом варианте - это будет кнопка), с помощью которого Вы в нужное время будете подавать сигнал низкого уровня на этот вход и включать, тем самым, свою процедуру обработки немаскируемого прерывания.

Обычно запросы "NMI" используются для прерывания работы микропроцессора при "катострофических" событиях, требующих немедленной реакции, таких, как аварийное пропадание питания, обнаружение ошибки памяти и т.д. Однако, все зависит еще и от Вашей фантазии. Так, например, нередко делают с помощью "NMI" распечатку любой понравившейся картинки на принтере.

Для возврата из программы обработки немаскируемого прерывания используйте команду АССЕМБЛЕРА - RETN (RETURN FROM NON MASKABLE INTERRUPT) - код ED 45H.

Теперь рассмотрим виды маскируемых прерываний, обрабатываемых по запросу на линии "INT".

"IM 0".
~~~~~~~

Прерывания нулевого типа. В "СПЕКТРУМЕ" этот тип прерывания не реализован вообще. Поэтому мы рассмотрим его вкратце.

С помощью режима "IM 0" возможно аппаратное переключение вектора прерывания, т.е. адреса программы обработки прерываний. Для этого вводится контроллер прерываний, который отсутствует в "СПЕКТРУМЕ". Такое устройство несложно организовать на основе БИС КР580ИК14 или КР580ВН59, которые по сути являются уже готовыми контроллерами прерываний. Как это организовать, Вы можете узнать из литературы по микропроцессорному комплекту КР580.

Всего векторов прерываний обычно восемь, в стандартном варианте, при использовании адресных линий A0-A2 эти вектора попадают на адреса рестартов процессора, т.е. на "RST0 - RST38H", которые уже задействованы в ПЗУ "СПЕКТРУМА", причем одна процедура "RST38H" служит для обработки прерываний. Хотя, конечно, можно это и обойти, используя другие адресные линии.

Кстати, без контроллера прерываний включение "IM 0" равносильно включению "IM 1".

"IM 1".
~~~~~~~~

После включения питания в "СПЕКТРУМЕ" устанавливается режим прерываний 1. "IM 1" - называется маскируемым прерыванием первого типа и выполняет обращение по адресу 0038H, что обеспечивает сканирование клавиатуры и изменение встроенного системного таймер, но если подключено ПЗУ контроллера дисководов или интерфейса микродрайва, то все, что делается при включении прерывания- это немедленный возврат без сканирования клавиатуры или каких-либо других действий, т.е. по адресу 0038H в этих ПЗУ записан код возврата из прерывания "RETI" (RETURN FROM INTERRUPT) - ED 4DH.

Поскольку "IM 1" использует "RST38H", а программа, находящаяся по адресу 0038H, уже хорошо сформирована и вряд ли требует каких-либо изменений и дополнений, то и интерес к прерывани-

ям первого типа носит, скорее всего, познавательный характер.

Для радиолюбителей и программистов гораздо интереснее прерывания второго типа, таящие в себе воистину богатейшие возможности.

Но прежде, чем перейти к прерываниям второго типа, мы немного отвлечемся в сторону и поговорим о формировании телевизионного изображения в "СПЕКТРУМЕ".

Все нижеописанное справедливо для фирменного "СПЕКТРУМА", т.к. речь пойдет о контроллере дисплея на микросхеме "ULA", но, по-видимому, должно представлять интерес для всех поклонников "СПЕКТРУМА", интересующихся этим режимом прерываний, независимо от того, какой версией компьютера они пользуются, ведь во всех версиях есть контроллеры дисплея, собранные на дискретных элементах, ПЗУ или ПЛМ, и кто знает, насколько близки эти контроллеры к оригинальной "ULA"-версии. Это во-первых. А во-вторых: владение этими знаниями, возможно, поможет стандартизации отечественного программного обеспечения.

При инициализации "СПЕКТРУМА" в регистре прерываний "I" помещается число 63 (03FH) и устанавливается первый тип прерываний. В принципе, задавать регистр "I" необязательно, т.к. режим "IM 1" его не использует, потому что любые прерывания идут через "RST38H". Но регистр "I" в фирменном "СПЕКТРУМЕ" дополнительно участвует в формировании телевизионного сигнала. "ULA" задействует биты 6 и 7 регистра "I".

При каждом цикле машинных команд Z-80 обращается к ячейке памяти, адресуемой регистром "I", выводя его в старшие восемь битов адресной шины, а линия запроса памяти "MREQ" активизируется. "ULA" генерирует прерывания каждый раз, когда необходимо изменить содержимое экрана. Это заставляет Z-80 запустить программу обработки прерываний, при условии, что прерывания подключены.

Когда программа обработки прерываний выполнена, процессор возвращается в точку, в которой он был при прерывании. Если это команды считывания/записи в память между 16384 (4000H) и 32767 (7FFFH), что "ULA" проверяет, просматривая две старшие линии адресной шины и линию "MREQ", то "ULA" приостанавливает микропроцессор до окончания изменения экрана.

Если старший бит регистра "I" сброшен, а бит 6 активизиро-

ван, то "ULA" может запутаться из-за регенерации динамической памяти. Активизируется линия "MREQ", и регистр "I" помещается в старшие восемь битов адресной шины. Далее "ULA" думает, что процессор производит запись или считывание в эту область ОЗУ, и при попытке это предотвратить, "ULA" пропускает свое собственное обращение для изменения дисплея, что приводит к развалу картинки. Поэтому в регистре "I" не должно содержаться любое число от 64 до 127 (040H до 07FH) включительно, т.е. с двумя

старшими битами, установленными вышеописанным образом.

Скорее всего, это справедливо и для других версий компьютеров с разделением поля памяти на 16К и 32К (см. "ZX-РЕВЮ" N1, 1991г.). По крайней мере, при записи в регистр "I" чисел из запрещенного интервала на компьютерах версий "НОВОСИБИРСК", "БАЛТИКА", "ЛЕНИНГРАД" развала картинки не наблюдалось. Но, по-видимому, все-таки лучше придерживаться указанных ограничений, если Вы хотите, чтобы ваши программы могли работать и на других версиях "СПЕКТРУМА", особенно, если Вы разрабатываете коммерческое программное обеспечение.

"IM 2".

~~~~~~~~

Путем установки "IM 2" Вы можете использовать прерывания для Ваших собственных целей. Тут уже все зависит только от Вас. С помощью "IM 2" возможно менять вектор прерываний как программно, так и аппаратно.

Режим "IM 2" сложен. При получении прерывания по линии "INT" процессор запоминает адрес следующей команды программы на машинном стеке, затем просматривает ячейку, указанную шиной данных плюс 256, умноженное на содержимое регистра "I" и передает управление к адресу, содержащемуся в этой ячейке плюс (256, умноженное на содержимое следующей ячейки). Вообще-то, считается дурным тоном иметь бит ноль на шине данных активированным в качестве указателя в режиме "IM 2", т.к. указатель всегда будет стартовать с адреса, пронумерованного четным числом, но, к сожалению, в "СПЕКТРУМЕ" нет выбора.
~~~~~~~~

ПРИМЕР 1.

В регистре "I" содержится число 10 (0AH), а на шине данных выставлено число 255 (0FFH). При этом :

10*256=2560

2560+255=2815.

Этот адрес находится в ПЗУ "СПЕКТРУМА", поэтому точка, к которой будет сделан переход, возьмется из содержимого адреса 2815+(256, умноженное на число по адресу 2816). В ячейке 2815 содержится 34, а в 2816 - 128. В этом можно убедиться с помощью оператора "PEEK". Таким образом, адрес перехода равен :

34+(256*128)=32802.

ПРИМЕР 2.

Регистр "I" содержит 200, шина данных 255.

200*256=51200

51200+255=51455.

Переход будет сделан по адресу, который Вы поместите в ячейки 51455 и 51456.

Гораздо проще все выглядит в шестнадцатиричном виде. Тут уже не надо вычислять адрес, откуда брать точку для перехода. Просто шина данных формирует младший байт адреса, а регистр "I" - старший байт. По этому адресу берется точка перехода на программу обработки прерываний, где опять-таки данный адрес содержит младший байт, а следующий за ним - старший байт адреса программы.

Все это можно представить, как косвенную адресацию по содержимому регистровой пары, причем регистровую пару на этот раз составляет шина данных и регистр "I". Это своего рода команда CALL (I+ШД). Но, т.к. такой команды не существует, и она не может разместить собственный адрес возврата, то адрес после последней выполненной команды кладется процессором на машинный стек, и к нему будет осуществлен возврат после выполнения программы обработки прерываний.

Как Вы видите, программное изменение вектора прерываний задается с помощью перезаписи регистра "I", а аппаратное - благодаря шине данных.

Обычно в момент прерывания на шине данных содержится - 255 (0FFH), но если Вы подключите внешнее устройство, изменяющее

в этот момент состояние шины данных, то Вы можете, не переписывая регистр "I", менять в небольших пределах адрес программы обработки "IM 2". Порой такое необходимо при подключении нескольких периферийных устройств, чтобы процессор знал, какое из них ему обслуживать на данный момент. Вдобавок ко всему, Вы можете еще перехватывать управление линией "INT" для аппаратного включения или выключения прерываний в необходимый момент времени. Именно так и делает контроллер дисковода.

С "IM 2" есть одна проблема при использовании прерываний при подключенном контроллере дисковода или микродрайва. Дело в том, что при переключении на теневое ПЗУ контроллера эти ПЗУ меняют указатель регистра "I". Поэтому, по всей видимости, до включения "IM 2" необходима проверка на то, какое ПЗУ подключено на данный момент. Хорошо еще, что в распространенной версии контроллера "TR- DOS" предусмотрено, что Вы можете изменить указатель в регистре "I".

Дополнительная проблема возникает при использовании указателей из ПЗУ при создании коммерческого матобеспечения, т.к. любые изменения и дополнения в ПЗУ могут сделать ваше матобеспечение неработающим.

При размещении в регистре "I" чисел от 0 до 63 (03FH), все адреса точек переходов по прерыванию "IM 2" лежат в области ПЗУ. Список указателей для фирменной версии ПЗУ "СПЕКТРУМА" приведен в "Справочнике" (ч.III данной книги).

Типичное использование "IM 2" в играх - управление спрайтами и звуковым сопровождением программы. Зная, с какой частотой генерируются прерывания, легко расчитать скорость перемещения спрайта или задать ритм мелодии. Они будут независимы от любых других операций внутри программы. Еще раз напомним, что прерывания генерируются с частотой 50 раз в секунду.

Есть несколько правил составления программ обработки прерываний "IM 2".

1. Перед выполнением программы необходимо запретить прерывания с помощью команды "DI". Причина состоит в том, чтобы гарантировать, что программа не зациклит петлю, если процедура обработки прерываний выполняется продолжительнее, чем пауза между двумя прерываниями.

2. После выполнения программы обработки "IM 2" подключите

прерывания командой "EI". Для возврата в основную программу используйте команду "RETI".

3. Когда бы ни использовались программы прерываний, очень важно, чтобы все регистры микропроцессора, используемые программой обработки прерывания, сохранялись на входе и восстанавливались перед возвращением.

4. Не должно быть попыток пересылки данных через регистры в программу обработки прерываний или из нее.

5. Помните, что если Вы не используете "RST 38H" в прерывающей программе, то Вам необходимо сбрасывать режим "IM 2" и включать "IM 1" до возвращения в БЕЙСИК.

И еще совет. Если Вам необходимо, чтобы таймер системы обновлялся, а клавиатура сканировалась, если, например, Вы работаете в БЕЙСИКЕ со включенным прерыванием второго типа, то в конце вашей программы обработки "IM 2" восстановите содержимое регистров (см. П.3 правил) и сделайте переход "JP 0038H". Программа "RST38H" сама подключит прерывания и сделает возврат в основную программу. Это позволит Вам сократить время обработки прерывания.

Вам в своей работе надо также учитывать то обстоятельство, что Есть операторы БЕЙСИКА, которые на время своего выполнения запрещают прерывания. Это операторы : LOAD, SAVE, BEEP.

Для полноты картины нам следует также ознакомиться с примерами составления программ в машинных кодах, использующих прерывания процессора в режиме IM 2. Желающих разобрать эти примеры мы отсылаем ко второй части данной книги.

5. 19 ПРОЧИЕ КОМАНДЫ

Мы рассмотрели почти все команды процессора Z-80, но есть еще несколько команд, не вошедших ни в какие группы. Разберем эти команды по очереди.

Команда NOP.

Ее код - 0. Это команда "НЕТ ОПЕРАЦИИ". По ней процессор ничего не делает, а выдерживает паузу продолжительностью 4 такта.

Команда CPL.

~~~~~~~~~~~~

Ее код - 2F. Она относится к регистру A и инвертирует каждый бит на противоположный. В результате получается как бы дополнение содержимого аккумулятора до 255 (в абсолютной двоичной арифметике).

Команда NEG.

~~~~~~~~~~~~

Ее код - ED 44. Команда работает в дополнительной двоичной арифметике и помещает в регистр A число, равное по абсолютной величине тому, которое там было, но с противоположным знаком. Влияет на все основные флаги.

Команда SCF.

~~~~~~~~~~~~

Код - 37. Команда включает флаг переноса (флаг C флагового регистра F).

Команда CCF.

~~~~~~~~~~~~

Код - 3F. Команда переключает флаг переноса на противоположный.

Команда HALT.

~~~~~~~~~~~~~

Код - 76. Эта команда вызывает остановку исполнения программы и последовательно исполняет команду NOP до тех пор, пока не поступит маскируемое прерывание. Команда используется, например, при применении оператора БЕЙСИКа PAUSE.

Интересная особенность: если Вы предварительно отключите маскируемые прерывания командой DI (см. выше), то применение команды HALT однозначно приведет к "зависанию" программы, т.к. прерывания процессор не дождется.

Команда DAA.

~~~~~~~~~~~~

Код - 27. По этой команде перестраивается содержимое аккумулятора. Оно переводится из двоичной формы в десятиричную, выраженную двоичной записью (BCD-арифметику).

6. ЗАКЛЮЧЕНИЕ.

На этом мы заканчиваем "Первые шаги в машинном коде".

Достаточно ли того, что Вы изучили, для того, чтобы начать программировать в машинных кодах или на языке АССЕМБЛЕРа?

По-видимому, нет. Хотя все необходимые сведения мы дали. Мы старались сделать изложение доступным и понятным даже для тех, кто только-только начал свое общение с компьютером, но Вам необходимы еще многие часы практики. Мы предлагаем Вам следующий подход в дальнейшей работе.

1. Во второй части этой книги "Практикум по программированию в машинных кодах" Вы найдете массу полезных примеров, разбор которых поможет Вам набраться опыта, а практические рекомендации помогут сэкономить дни самостоятельной работы.

2. Приобретите какую-либо программу ДИСАССЕМБЛЕР (конечно с инструкцией) и с ее помощью попробуйте просмотреть машинный код фирменных игровых программ. Анализируйте увиденное. Вносите изменения и смотрите, что из этого получается. Это спорт, и очень захватывающий. Вы сами не заметите, как начнет приходить опыт.

3. Приобретите программу АССЕМБЛЕР с инструкцией. Вы увидите, что АССЕМБЛЕР имеет немало возможностей для упрощения программирования в кодах. Он имеет несколько своих команд, называемых директивами АССЕМБЛЕРа и они помогут Вам в составлении хорошо читаемых, удобных программ.

После этого Вы сможете начать писать свои процедуры и программы в машинных кодах, но не старайтесь как можно больше сделать сами. Наоборот, максимум процедур старайтесь взять в готовом виде из других программ, конечно внося необходимые изменения. Ведь изобраетая новый автомобиль, конструктор не изобретает к нему новые болты и гайки.

На всех этапах работы Вам поможет также часть третья нашей книги - "Справочник по программированию в машинных кодах Z-80".

ЧАСТЬ II.

ПРАКТИКУМ ПО ПРОГРАММИРОВАНИЮ В МАШИННЫХ КОДАХ

ВВЕДЕНИЕ

Вы перешли ко второй части нашего учебного пособия, что безусловно свидетельствует о серьезности Ваших намерений изучить программирование в машинных кодах и на языке АССЕМБЛЕРа для бытовых персональных ЭВМ типа "ZX-Спектрум" и других, совместимых с системой СИНКЛЕР.

Вы познакомились с архитектурой процессора, с системой команд Z-80, мы дали Вам первичную информацию, позволяющую преодолеть психологический барьер при переходе к программированию в машинном коде. Работая с материалом второй части, Вы приобретете навыки практической работы и узнаете те приемы и "маленькие хитрости", которые широко используют опытные программисты, но перед которыми встают в тупик начинающие.

Приведем пример. Что Вы подумаете, если встретите в программе две команды, следующие одна за другой?

INC D - увеличить на единицу содержимое регистра D.

DEC D - уменьшить на единицу содержимое регистра D.

На первый взгляд, программист бесполезно истратил 2 байта памяти, потому что эти две команды взаимно противоположны и никак не изменяют содержимое регистра D. Это верно, но не совсем. Дело в том, что обе команды имеют еще и косвенный эффект, связанный с тем, что в соответствии с их результатом выставляются флаги регистра F. Поэтому результат действия этих команд эквивалентен как бы действию команды: "Проверить содержимое регистра D и, не изменяя его, установить флаги регистра F в соответствии с ним". Такой команды процессор Z-80 не имеет, вот почему приходится прибегать к обходному маневру.

Вот таким конкретным приемам мы и обучим тех читателей, у которых хватит бодрости прочитать и вторую часть книги.

В рамках глав второй части мы для упрощения подачи материала будем пользоваться следующими условными обозначениями:

N - целое число от 0 до 255 (байт);

NN - целое число от 0 до 65535 (слово);

A - аккумулятор (регистр A микропроцессора);
reg - регистр общего назначения (регистр B,C,D,E,H или L);
rp - регистровая пара (шестнадцатиразрядные регистры HL,BC,DE);
xy - индексный регистр (IX или IY);
ADDR - шестнадцатиричный адрес в поле памяти компьютера;
rph - старший регистр регистровой пары (B,D,H);
rpl - младший регистр регистровой пары (C,E,L);

В тех случаях, когда в процедурах будут участвовать несколько регистров, будут применяться обозначения reg1, reg2 или rp1, rp2, а также rph1, rph2, rpl1, rpl2.

На всех этапах работы Вам может быть полезна информация, приведенная в третьей части книги - "Справочник...". В частности, там указаны все косвенные воздействия команд процессора на содержимое флагового регистра. Вам также могут быть полезны те данные, которые имеются в справочнике о длительности исполнения команд процессора.

1. ОСОБЕННОСТИ ПРОЦЕССОРА Z-80

Мы полагаем, что приступая к изучению данной книги, Вы либо уже проработали "Первые шаги в машинных кодах Z-80", либо сами обладаете достаточным уровнем подготовки. Тем не менее, мы уделим несколько страниц краткому повторению материала 1-ой части, концентрированно рассмотрев основные особенности процессора.

Регистры процессора Z-80 очень несимметричны и программист должен тщательно планировать, какие данные и адреса в какие регистры направить.

1.1. Аккумулятор (регистр A) и регистровая пара HL обладают отличительными уникальными особенностями.

Аккумулятор - единственный однобайтный регистр, который может загружаться и выгружаться напрямую. Аккумулятор - единственный регистр, для которого можно ОДНОБАЙТНОЙ командой сделать сдвиг, инверсию, изменение знака. Его можно загрузить или

выгрузить косвенной адресацией путем указания на адрес, содержащийся в регистровых парах BC или DE. Для него могут выполняться операции IN и OUT с прямой адресацией.

HL - единственная регистровая пара, которая может служить в качестве косвенного указателя адреса для арифметических и логических операций или при загрузке/выгрузке регистров, отличных от аккумулятора.

HL - единственная регистровая пара, содержимое которой может быть напрямую передано в указатель стека SP или в программный счетчик PC.

Более того, регистр HL может служить в качестве аккумулятора двойной длины в операциях 16-разрядного сложения.

Благодаря наличию инструкции EX DE,HL, имеется возможность полного обмена содержимым между регистрами DE и HL.

Регистр HL почти всегда используется для хранения адресов, потому что имеются операции, позволяющие косвенное обращение к данным, находящимся в адресах, на которые указывает содержимое регистра HL, а также благодаря наличию таких операций, как LD SP,HL; JP (HL); EX (SP),HL; EX DE,HL. Если для операции необходимо наличие второго адреса, то его целесообразно хранить в регистровой паре DE благодаря наличию команды EX DE,HL.

1.2. Очень часто одни и те же регистры называют по-разному. Регистры A,B,C,D,E и L называют 8-битными. Регистровые пары BC (старший - B), DE (старший - D), HL (старший - H) называют 16-битными регистровыми парами. Термины "регистровая пара BC", "регистровая пара B", "регистр BC" означают одно и то же. Это же относится и к регистровым парам DE и HL. Равнозначны же и термины "регистровая пара", "регистр двойной длины", "16-разрядный регистр", "16-битный регистр".

1.3. Влияние операций на флаги в высшей степени неоднозначно. Вот некоторые необычные эффекты:

- логические операции OR, XOR и AND обнуляют флаг переноса;

- однобайтные операции ротации аккумулятора не влияют на флаги, кроме флага переноса;

- при работе с регистровыми парами загрузка данных в rp,

выгрузка данных в память, перемещение данных из rp в rp, увеличение содержимого пары на единицу (инкремент), уменьшение на единицу (декремент) не влияют на флаги вообще;

- операции 16-битного сложения влияют только на флаг переноса.

1.4. Флаг четности/переполнения выполняет двойную роль. В одних случаях он указывает на наличие четности или нечетности содержимого аккумулятора (понятие четности относится не к числу, содержащемуся в аккумуляторе, а к количеству включенных битов); в других случаях этот флаг указывает на наличие переполнения (в двоичной дополнительной арифметике).

1.5. Кроме прямой адрсации, Z-80 допускает также косвенную и индексную адресацию. Косвенной адресации через ячейки памяти нет. Этот недостаток компенсируется путем загрузки адреса в регистр HL. Таким образом, косвенная адресация - это двуступенчатый процесс. Адрес при косвенной адресации может содержаться и в регистрах BC и DE, но в этом случае косвенная адресация возможна только в операциях с аккумулятором.

1.6. Индексные регистры позволяют реализовать методы индексной адресации, когда базовый адрес содержится в индексном регистре, а после кода операции задается величина смещения адреса искомой ячейки относительно базового. Смещение S может быть только 8-битным (от 0 до 255). Основная цель индексной адресации - работа с массивами данных как с таблицами.

В более общей форме индексной адресации с 16-битным смещением необходимы операции сложения содержимого регистровых пар, при этом регистр HL выступает в качестве аккумулятора. Таким образом, в данном случае, индексная адресация требует несколько шагов. Смещение (16-битное) загружается в одну регистровую пару, базовый адрес - в другую. Одной из этих пар обязательно должна быть HL. Другая может быть любой, например, BC. Далее производится сложение ADD HL,BC, а затем сумма используется как косвенный адрес в операциях с регистром (HL).

1.7. Многие операции имеют побочные эффекты и при програм-

мировании этим активно пользуются. Так, в системе команд процессора нет команды на очистку аккумулятора. Для этой цели используют команды SUB A или XOR A.

Сброс флага переноса - AND A или OR A.

Логический сдвиг аккумулятора влево - ADD A,A.

Как AND A, так и OR A сбрасывают флаг переноса, а остальные флаги устанавливают в соответствии с содержимым аккумулятора, но запомните: загрузка регистра не влияет на состояние флагов.

1.8. Ветвление в программах может исполняться посредством команд условного и безусловного переходов. Для безусловного перехода существуют два метода - абсолютный переход по заданному адресу и относительный переход на заданное количество байтов вперед или назад. Им соответствуют мнемоники JP и JR.

Наборы команд для абсолютного и относительного условного переходов отличаются. Относительный переход выполняется только по состоянию флагов переноса и нуля, а абсолютный - по флагу переноса, нуля, знака и переполнения/четности.

Относительные переходы занимают меньше места в памяти (2 байта вместо 3), но исполняются медленнее (12 тактовых циклов вместо 10).

1.9. Операции инкремента и декремента ведут себя по-разному, в зависимости от того, для однобайтного или для двухбайтного операнда они применяются. Для однобайтного регистра они влияют на все флаги, кроме флага переноса. Для двухбайтного регистра они не влияют на флаги вообще. Таким образом, при использовании 16-битной пары, например BC, в качестве счетчика, единственный способ убедиться, что ее содержимое равно нулю - это выполнить операцию логическое "ИЛИ" между отдельными регистрами этой пары.

```
LD A,B
OR C
```

Только если и в B и в C содержатся нули, результат этой операции будет равен нулю. 16_ТИ РАЗРЯДНЫЕ РЕГИСТРЫ ПРЕДНАЗНАЧЕНЫ В ОСНОВНОМ ДЛЯ ОПЕРАЦИЙ С АДРЕСАМИ, А НЕ С ДАННЫМИ, отсюда такая сложность действий.

1.10. Система команд процессора Z-80 создавалась на основе системы команд процессора 8080. Те операции, которые являются дополнительными по отношению к процессору 8080, выполняются медленнее, чем аналогичные им операции того же метода адресации. Это операции с отдельными битами, команды арифметического сдвига и некоторые операции копирования. Они работают медленнее, т.к. перед ними стоит префикс, который распознается процессором и сообщает ему, что операция не принадлежит к набору 8080, а следующий байт является кодом операции.

1.11. Процессор Z-80 допускает проведение операций со стеком. На стек может помещаться только содержимое регистровых пар или индексных регистров. Для этих операций можно рассматривать регистры A и F как отдельную регистровую пару AF, состоящую из аккумулятора (старший байт) и набора флагов (младший байт). Команды CALL и RET помещают адрес на стек или снимают его со стека.

1.12. Процессор Z-80 имеет триггер управления прерываниями. Он либо разрешает, либо запрещает обработку прерываний. Состояние этого триггера можно проверить с помощью команды LD A,I или LD A,R. И та и другая команда передает состояние триггера прерываний во флаг переполнения/четности регистра F.

1.13. Для процессора Z-80 действуют следующие общепринятые соглашения:

- 16-битный адрес хранится в памяти так, что сначала записывается младший байт, а затем старший. Этот же порядок принят в процессорах 8080, 8085 и 6502. Противоположный порядок принят, например, в процессорах 6800 и 6809;

- в регистре SP (указатель стека) содержится адрес, по которому находится младший байт содержимого вершины стека. Такой же порядок принят для процессоров 8080, 8085, 6800. В процессорах 6502 принят другой порядок - указатель указывает на очередной свободный адрес;

- когда прерывание разрешено, триггер прерываний содержит 1 и когда запрещено - 0. Этот же порядок в процессорах 8080 и 8085, но противоположный - в процессорах 6502, 6800, 6809.

2. РАСШИРЕНИЕ СИСТЕМЫ КОМАНД Z-80

Процессор Z-80 является одним из самых насыщенных в мире по системе команд, которых насчитывается около 700. В то же время очевидно, что разработчики не могут сделать никакой процессор всеохватывающим. Планируя набор команд, разработчики принимают, во-первых, самые нобходимые команды и, во-вторых, учитывают возможность их комбинирования для реализации дополнительных возможностей.

В этом обширном разделе мы рассмотрим те "маленькие хитрости", которые позволяют программисту "конструировать" новые функциональные команды из имеющихся в стандартном набор Z-80.

В основу подборки этих приемов мы положили три принципа:

- дать очевидно полезные приемы и решения;
- дать эквиваленты командам, имеющимся в системах других процессоров, но отсутствующим в системе команд Z-80;
- указать на иные эквиваленты команд стандартного набора.

Такие расширяющие конструкции мы выделили в семь групп:

- арифметические;
- логические;
- передачи данных;
- ветвления;
- вызова;
- возврата;
- прочие.

Мы даже пойдем еще дальше и там, где это возможно, запишем мнемоники сконструированных нами "квазикоманд" процессора, т.е. мнемоники команд, которых нет на самом деле, но эффект действия которых эмулируется предлагаемой конструкцией. Чтобы отличать эти "квазимнемоники" от настоящих, мы будем печатать их с подчеркиванием. Например:

LD PC, (ADDR) - загрузить в программный счетчик адрес, ко-
~~~~~~~~~~~~~~ торый содержится в ячейках ADDR и ADDR+1.
~~~~~~~~~~~~~~

2.1. АРИФМЕТИЧЕСКИЕ КОНСТРУКЦИИ

2.1.1. Сложение без учета флага переноса.

1. Прибавить к аккумулятору содержимое ячейки памяти.

```
ADD A,(ADDR)        LD HL,ADDR     - указание на адрес
~~~~~~~~~~~         ADD A,(HL)     - сложение
```

2. Десятиричное прибавление числа к аккумулятору.

```
DADD A,N            ADD A,N        - сложение
~~~~~~~~            DAA            - перевод в десятиричную
                                     форму
```

3. Десятиричное прибавление содержимого регистра к аккумулятору.

```
DADD A,reg          ADD A,reg      - сложение
~~~~~~~~~~          DAA            - перевод в десятиричную
                                     форму
```

4. Прибавить 16-битное число к содержимому рег.пары HL.

```
ADD HL,NN           LD rp,NN       - загрузка числа в BC или DE.
~~~~~~~~~           ADD HL,rp      - сложение
```

5. Прибавить 16-битное число к содержимому индексного регистра.

```
ADD xy,NN           LD rp,NN       - загрузка числа
~~~~~~~~~           ADD xy,rp      - сложение
```

Регистровой парой может быть либо BC либо DE.

6. Прибавить к регистру HL содержимое адреса двух смежных ячеек памяти.

```
ADD HL,(ADDR)       LD rp,(ADDR)   - загрузка числа
~~~~~~~~~~~~        ADD HL,rp      - сложение
```

7. То же, но для индексного регистра.

```
ADD xy,(ADDR)       LD rp,(ADDR)   - загрузка числа
~~~~~~~~~~~~        ADD xy,rp      - сложение
```

8. Прибавить содержимое ячеек памяти ADDR1, ADDR1+1 к содержимому ячеек памяти ADDR2, ADDR2+1.

```
ADD (ADDR2),(ADDR1) LD HL,(ADDR2) - взять содержимое ADDR2
~~~~~~~~~~~~~~~~~~  LD DE,(ADDR1) - взять содержимое ADDR1
                    ADD HL,DE     - сложить их
                    LD (ADDR2),HL - поместить сумму в ADDR2
```

9. Прибавить 16-битное число к содержимому адреса.

```
ADD (ADDR),NN       LD HL,(ADDR)  - взять содержимое адреса
~~~~~~~~~~~~~       LD DE,NN      - взять число
                    ADD HL,DE     - сложить их
                    LD (ADDR),HL  - поместить сумму в адрес.
```

2.1.2. Сложение с учетом флага переноса.

10. Прибавить содержимое адреса к аккумулятору с учетом переноса.

```
ADC A,(ADDR)        LD HL,ADDR    - указание на адрес
~~~~~~~~~~~~        ADC A,(HL)    - сложение с учетом переноса
```

11. Прибавить к аккумулятору содержимое флага (переноса).

```
                    ADC A,0
```

12. Десятиричное прибавление числа к аккумулятору с учетом флага переноса.

```
DADC A,N            ADC A,N
~~~~~~~~            DAA
```

13. Десятиричное прибавление содержимого регистра к аккумулятору с учетом флага переноса.

```
DADC A,reg          ADC A,reg
~~~~~~~~~~          DAA
```

14. Прибавление 16-битного числа к HL с учетом переноса.

```
ADC HL,NN           LD rp,NN
~~~~~~~~~           ADC HL,rp
```

15. Прибавление к регистру HL содержимого адреса с учетом переноса

```
ADC HL,(ADDR)       LD rp,(ADDR)
~~~~~~~~~~~~        ADC HL,rp
```

2.1.3. Команды вычитания без учета двоичного займа.

16. Вычесть содержимое адреса из аккумулятора.

```
SUB (ADDR)          LD HL,ADDR
~~~~~~~~~~          SUB (HL)
```

17. Десятиричное вычитание числа из аккумулятора

```
DSUB N              SUB N
~~~~~~              DAA
```

18. Десятиричное вычитание содержимого регистра из аккумулятора.

```
DSUB reg            SUB reg
~~~~~~~~            DAA
```

19. Вычесть содержимое регистровой пары из HL.

```
SUB HL,rp           AND A        - сброс флага переноса
~~~~~~~~~           SBC HL,rp    - само вычитание
```

Необходимость во введении этой конструкции вызвана тем, что простого вычитания содержимого регистровой пары из HL без учета флага переноса в системе команд Z-80 нет. Приходится применять SBC, но предварительно обнулять флаг переноса.

2.1.4. Обратное вычитание.

20. Вычесть аккумулятор из числа, и результат поместить в аккумулятор.

```
SUB N,A             NEG      - изменение знака аккумулятора
~~~~~~~~            ADD A,N  - сложение (вместо вычитания)
   ИЛИ
                    LD reg,A - запомнили аккумулятор
                    LD A,N   - ввели число в аккумулятор
                    SUB reg  - вычитание
```

21. Вычесть аккумулятор из регистра и результат поместить в аккумулятор.

```
SUB reg,A           NEG
~~~~~~~~~           ADD A,reg
```

22. Десятиричное вычитание аккумулятора из числа.

```
DSUB N,A            LD reg,A  - запомнили аккумулятор
~~~~~~~~            LD A,N    - ввели число
                    SUB reg   - вычитание
                    DAA       - перевод в десятиричную форму
```

23. Десятиричное вычитание аккумулятора из регистра.

```
DSUB reg,A          LD reg1,A
~~~~~~~~~~          LD A,reg
                    SUB reg1
                    DAA
```

2.1.5. Вычитание с двоичным займом (с учетом флага переноса)

24. Вычесть содержимое адреса из аккумулятора.

```
SBC A,(ADDR)        LD HL,ADDR    - указали на адрес
~~~~~~~~~~~~        SBC A,(HL)    - вычитание с учетом флага C
```

25. Вычесть состояние флага C из аккумулятора.

```
                    SBC A,0
```

26.Десятиричное вычитание числа из аккумулятора с учетом C

```
DSBC A,N            SBC A,N
~~~~~~~~            DAA
```

27. Десятиричное вычитание содержимого регистра из аккумулятора с учетом флага переноса.

```
DSBCV A,reg         SBC A,reg
~~~~~~~~~~~~        DAA
```

28. Вычитание двойного числа из HL с учетом флага переноса.

```
SBC HL,NN           LD rp,NN
~~~~~~~~~           SBC HL,rp
```

2.1.6. Команды увеличения (инкремент).

29. Выполнить инкремент для содержимого заданного адреса.

```
INC (ADDR)          LD HL,ADDR
~~~~~~~~~~          INC (HL)
```

30. Инкремент аккумулятора с включением флага C, если в результате получается 0.

```
                    ADD A,1
```

O В отличие от команды INC команда ADD влияет на флаг переноса.

31. Десятиричный инкремент аккумулятора.

```
                    ADD A,1
                    DAA
```

32. Десятиричный инкремент регистра.

```
                    LD A,reg
                    ADD A,1
                    DAA
                    LD reg,A
```

Необходимость переброски через аккумулятор вызвана тем,

что команда DAA может применяться только к аккумулятору.

33. Инкремент содержимого двух смежных ячеек памяти.

```
              LD HL,(ADDR)
              INC HL
              LD (ADDR),HL
    ИЛИ
              LD HL,(ADDR)
              INC (HL)
              JR NZ,END
              INC HL
              INC (HL)
              DEC HL
      END:    NOP
```

ПРИМЕЧАНИЕ:

а) В команде JR NZ,S мы вместо числа дали метку END. В реальной программе здесь, конечно же, надо будет подставить число, указывающее на сколько байтов вперед или назад надо перейти. Отметим, что применение метки позволяет не утруждать себя вычислением величины этого перехода. Все АССЕМБЛЕРы понимают назначение меток и, если Вы пишете свою программу в АССЕМБЛЕРе, то можете сильно упростить себе жизнь. Если же Вы программируете в маш. кодах вручную, то придется выполнить подсчет.

б) Мы привели два варианта выполнения требуемой команды. На глаз видно, что второй вариант значительно длиннее, т.е. занимает больше места в памяти. Интересно оценить быстродействие первого и второго вариантов. Для этого Вам может помочь наш "Справочник..." в котором приведено время выполнения каждой команды процессора, измеренное в тактовых циклах

LD HL,(ADDR)	(16)	LD HL,ADDR	(10)	(10)
INC HL	(6)	INC (HL)	(11)	(11)
LD (ADDR),HL	(16)	JR NZ,END	(12)	(7)
	-----	INC HL	-	(6)
	38	INC (HL)	-	(11)
		DEC HL	-	(6)
			----------	----------
			33	51

Итак, первый вариант длится 38 тактовых циклов, а второй

может длиться 33 цикла или 51 цикл, в зависимости от того, произошло ли переполнение младшего байта после его приращения. Поскольку переполнение может произойти только в одном случае из 256, то наиболее вероятна ветвь, которая длится 33 цикла. Таким образом, мы имеем наглядный пример того, как программист может выбирать разные решения в зависимости от того, что ему важнее - экономия памяти или скорость работы.

2.1.7. Команды уменьшения (декремент).

34. Уменьшить содержимое заданного адреса.

```
DEC (ADDR)          LD HL,ADDR
~~~~~~~~~~          DEC (HL)
```

35. Декремент аккумулятора с включением флага C, если происходит двоичный заем.

```
                    SUB 1
```

36. Декремент аккумулятора с включением флага C, если двоичный заем не происходит.

```
                    ADD A,FF
```

37. Десятиричный декремент аккумулятора.

```
                    SUB 1
                    DAA
```

38. Десятиричный декремент регистра.

```
                    LD A,reg
                    SUB 1
                    DAA
                    LD reg,A
```

39. Декремент 16-битного числа, расположенного по адресу ADDR и ADDR+1.

```
                    LD HL,(ADDR)
                    DEC HL
                    LD (ADDR),HL
```

40. Декремент 16-битного регистра с включением флага Z (флага нуля), если в результате получается 0.

```
                   DEC rp
                   LD A,rpl       - проверка на 0 с
                   OR rph           выставлением флагов
```

2.1.8. Команды умножения.

41. Умножение аккумулятора на 2.

```
ADD A,A
```

42. Умножение аккумулятора на 3.

```
LD reg,A    - запомнили аккумулятор
ADD A,A     - умножили аккумулятор на 2
ADD A,reg   - прибавили к результату содер-
              жимое аккумулятора еще раз.
```

43. Умножение аккумулятора на 4.

```
ADD A,A
ADD A,A
```

Те же приемы могут быть применены и для умножения на другие малые целые числа.

44. Умножение содержимого регистра на 2.

```
SLA reg
```

45. Умножение содержимого регистра на 4.

```
SLA reg
SLA reg
```

Примечание: поскольку инструкция SLA - двухбайтная, все же быстрее перемещать содержимое регистра в аккумулятор и выполнять умножение через однобайтную команду ADD A,A. Вычисления выполняются почти в три раза быстрее.

46. Умножить на 2 содержимое регистра HL.

```
ADD HL,HL
```

47. Умножить на 3 содержимое регистра HL.

```
LD rph,H       - запомнили старший байт.
LD rpl,L       - запомнили младший байт
ADD HL,HL      - умножили на 2
ADD HL,rp      - умножили на 3
```

48. Умножить на 2 содержимое индексного регистра.

```
ADD xy,xy
```

49. Умножить на 2 16-битное число, находящееся в заданном адресе.

```
LD HL,ADDR     - указание на адрес
SLA (HL)       - умножили на 2 младший байт
INC HL         - перешли к старшему байту
RL (HL)        - умножили на 2 старший байт.
```

Обратите внимание на то, что при умножении старшего байта мы использовали команду RL, а не SLA. Это необходимо, чтобы "прихватить" и содержимое флага переноса, т.к. при умножении на 2 младшего байта мог ведь возникнуть и перенос единицы, которую надо учесть в старшем байте.

Второй вариант этой конструкции:

```
LD xy,ADDR
SLA (XY+0)
RL (XY+1)
```

2.1.9. Команды деления.

~~~~~~~~~~~~~~~~~~~~~~

50. Деление аккумулятора пополам (без знака).

```
SRL A
```

51. То же, но со знаком.

```
SRA A
```

52. Деление аккумулятора на 4 (без знака).

```
SRL A
SRL A
```

ИЛИ ТО ЖЕ САМОЕ, НО БЫСТРЕЕ

```
RRA
RRA
AND 3F
```

В результате ротации два младших бита перейдут в старшие. Теперь надо их очистить, что и делается маскированием их командой AND 0011 1111. Обратите внимание, что 0011 1111B = 3FH.

53. Деление на 2 16-битного содержимого адреса.

```
LD xy,ADDR
SRL (xy+1)      - деление старшего байта
RR (XY+0)       - деление младшего байта с
                  "прихватыванием" флага C.
```

54. Деление содержимого регистра на 2 (без знака).

```
SRL rph
RR rpl
```

55. То же, но со знаком.

```
SRA rph
RR rpl
```
~~~~~~~~~~~~~~~~~~~~~~

2.1.10. Команды сравнения.

~~~~~~~~~~~~~~~~~~~~~~~~~

56. Сравнить аккумулятор с числом. Включить в результате биты в тех позициях, где они отличаются.

```
                XOR N
```

57. Сравнить регистровую пару rp с HL. Если rp больше, включить флаг переноса, в противном случае - выключить.

```
CP rp,HL        AND A          -  сброс флага C
~~~~~~~~        SBC HL,rp
```

58. Сравнить HL с двухбайтным числом.

```
CP HL,NN        AND A
~~~~~~~~        LD rp,NN
                SBC HL,rp
```

59. Сравнить содержимое регистровой пары с двухбайтным числом, находящимся по заданному адресу.

```
CP rp,(ADDR)    AND A
~~~~~~~~~~~     LD HL,rp
                LD rp,(ADDR)
                SBC HL,rp
```

60. То же, но для индексного регистра.

```
CP xy,(ADDR)    PUSH xy        - переброска содержимого в HL
~~~~~~~~~~~~    POP HL           при посредстве стека
                AND A
                LD rp,(ADDR)
                SBC HL,rp
```

61. Сравнить указатель стека с 16-битным числом.

```
CP SP,NN        LD HL,0        - очистка HL
~~~~~~~~        ADD HL,SP      - переброска SP в HL
                LD rp,-NN
                ADD HL,rp
```

62. Сравнить указатель стека с 16-битным числом, находящимся в указанном адресе.

```
CP SP,(ADDR)    LD HL,0
~~~~~~~~~~~~    ADD HL,SP
                LD rp,(ADDR)
                AND A
                SBC HL,rp
```
~~~~~~~~~~~~

2.1.11. Изменение знака (в двоичной дополнительной форме).
~~~~~~

63. Изменение знака регистра.

```
NEG reg             SUB A         - обнуление аккумулятора
~~~~~~~             SUB reg       - формирование -reg в аккум-ре
                    LD reg A      - формирование -reg в регистре
ИЛИ
                    LD A,reg
                    NEG
                    LD reg,A
```

64. Изменение знака в заданном адресе.

```
NEG (ADDR)          SUB A
~~~~~~~~~~          LD HL,ADDR
                    SUB (HL)
                    LD (HL),A
ИЛИ
                    LD HL,ADDR
                    LD A,(HL)
                    NEG
                    LD (HL),A
```

65. Изменение знака регистровой пары.

```
NEG rp              LD A,rph
~~~~~~              CPL
                    LD rph,A      - инвертировали старший байт
                    LD A,rpl
                    CPL
                    LD rpl,A      - инвертировали младший байт
                    INC rp        - прибавили 1 для получения
                                    результата.
ИЛИ
                    LD HL,0
                    AND A
                    SBC hl,rp
```
```

66. Изменение знака 16-битного числа, находящегося в указанном адресе.

```
NEG (ADDR)          LD HL,0
~~~~~~~~~~~         LD rp,(ADDR)
                    AND A
                    SBC HL,rp
                    LD (ADDR),HL
```

67. Дополнение до 99. (Т.е. вычисление 99 -"A")

```
                    LD reg,A
                    LD A,99
                    SUB reg
```

Давать команду DAA здесь не нужно, поскольку если к началу операции в аккумуляторе находилось правильное BCD-число, то и 99 минус аккумулятор тоже будет правильным BCD-числом.

68. Дополнение до 100.

```
                    NEG
                    DAA
```

2.1.12. Команды преобразований.

69. Преобразовать содержимое аккумулятора в 16-битное число без знака и записать его в регистровой паре.

```
                    LD rpl,A
                    LD rph,0
```

70. То же, но со знаком.

Напомним, что в двоичной дополнительной форме, которая применяется для записи целых чисел со знаком, старший (седьмой, он же левый) бит является знаковым.

```
                    LD rpl,A
                    ADD A,A         - знаковый бит переносится во
                                      флаг переноса C.
                    SBC A,A         - в аккумуляторе остается ноль
                                      минус флаг C.
                    LD rph,A
```

71. Преобразование содержимого заданного адреса в 16-битную форму со знаком.

```
                    LD HL,ADDR      - указание на адрес
                    LD A,(HL)       - взяли число
```

```
ADD A,A       - перевод знакового бита в С
SBC A,A       - формирование знакового байта
INC HL        - указание на старший байт
LD (HL),A     - установка старшего байта
```

72. Приведение аккумулятора по младшему биту. Если младший бит равен нулю, то в аккумуляторе выставляется 0, а если он равен единице, то FF.

```
RRA       - младший бит переносится в С
SBC A,A   - формирование (0 минус флаг С)
```

73. Приведение аккумулятора по знаку. Если в аккумуляторе положительное число, то в нем выставляется 0, а если отрицательное, то FF.

```
ADD A,A       - знаковый бит переносится в С
SBC A,A
```

2.2. ЛОГИЧЕСКИЕ КОНСТРУКЦИИ

74. Сброс заданных битов в аккумуляторе - AND N.

Число N является маской. Оно содержит нули в тех битах, которые должны быть выключены в аккумуляторе и единицы в тех позициях, которые должны остаться без изменений. Например,

AND 1101 1011B - выключает 2-ой и 5-ый биты.

Поскольку команда RES выключает биты только по одному, то для получения такого же результата пришлось бы давать последовательность:

```
RES 2,A
RES 5,A
```

75. Проверка битов в аккумуляторе.

Задача состоит в том, чтобы выставить флаги так, как будто бы была выполнена команда AND для аккумулятора и для регистра или адреса, но содержимое аккумулятора при этом оставить без изменений.

```
LD reg,A      - запомнили аккумулятор
LD HL,ADDR    - указание на адрес
AND (HL)      - логическое "И"
LD A,reg      - восстановили аккумулятор
```

Команда LD на флаги не влияет.

76. Операция AND для флагового регистра и числа. Маскирование флагов.

```
AND F,N          PUSH AF      - переброска флаг. регистра
~~~~~~~          POP rp         через стек
                 LD A,N       - ввод маски
                 AND rpl      - маскирование
                 LD rpl,A     - сохранение результата в rpl
                 PUSH rp      - обратная переброска в
                 POP AF         регистр F через стек
```

2.2.2. Логические конструкции "ИЛИ".

77. Включение заданных битов в аккумуляторе.

OR N

Число N является маской. Оно содержит единицы в тех битах, которые должны быть включены в аккумуляторе и нули в тех позициях, которые должны остаться без изменений. Например:

OR 0010 0100B - включает 2-ой и 5-й биты.

Поскольку команда SET включает биты только по одному, то для получения такого же результата пришлось бы давать последовательность:

```
SET 2,A
SET 5,A
```

78. Проверка 16-разрядного регистра на 0.

```
LD A,rph
OR rpl
```

В результате операции будет 0 (включится флаг Z) в том и только в том случае, если и старший и младший байт проверяемого регистра равны нулю.

79. Операция OR для флагового регистра и числа. Маскирование флагов.

```
OR F,N           PUSH AF      - переброска флаг. регистра
~~~~~~           POP rp         через стек
                 LD A,N       - ввод маски
                 OR rpl       - маскирование
                 LD rpl,A     - сохранение результата в rpl
                 PUSH rp      - обратная переброска в
                 POP AF         регистр F через стек
```

2.2.3. Логические конструкции "ИСКЛЮЧАЮЩЕЕ ИЛИ".

80. Переключение заданных битов в аккумуляторе.

```
XOR N
```

Число N является маской. Оно содержит единицы в тех битах, которые должны быть переключены в аккумуляторе и нули в тех битах, которые должны остаться без изменений. Например,

XOR 0010 0100B - переключает 2-й и 5-й биты.

81. Инвертировать аккумулятор с установкой флагов.

```
XOR FF
```

Эта команда отличается от CPL тем, что в отличие от нее влияет на флаги.

82. Побитное сравнение. Сравнить аккумулятор с регистром и включить биты в тех позициях, которые отличаются.

```
XOR reg
```

2.2.4. Логические конструкции "НЕ".

84. Инвертировать содержимое заданного адреса.

```
CPL (ADDR)          LD HL,ADDR
~~~~~~~~~~          LD A,(HL)
                    CPL
                    LD (HL),A
```

85. Инвертировать младший бит содержимого регистра.

```
                    INC reg
```

ИЛИ

```
                    DEC reg
```

Правда, здесь надо иметь в виду, что эти команды могут изменить прочие биты в регистре.

86. Инвертировать младший бит заданного адреса.

```
                    LD HL,ADDR
                    INC (HL)
```

ИЛИ

```
                    LD HL,ADDR
                    DEC (HL)
```

87. Инвертировать младший полубайт в аккумуляторе.

```
                    XOR 0F
```

88. То же, но старший.

```
            XOR F0
```

89. Инвертировать содержимое регистровой пары.

```
CPL rp      LD HL,FFFF
~~~~~~      AND A
            SBC HL,rp
```

Повторно напоминаем, что поскольку для 16-разрядных регистров нет команды SUB, приходится использовать команду SBC, а она влечет за собой еще и вычитание содержимого флага переноса (флага C), поэтому этот флаг надо предварительно занулить (очистить), что делается командой AND A.

2.2.5. Конструкции сдвига.

90. Логический сдвиг аккумулятора влево.

```
            ADD A,A
```

91. Логический сдвиг влево содержимого HL.

```
SLL HL      ADD HL,HL
~~~~~~
```

92. Логический сдвиг влево индексного регистра.

```
SLL xy      ADD xy,xy
~~~~~~
```

93. Логический сдвиг вправо содержимого регистровой пары.

```
SRL rp      SRL rph
~~~~~~      RR rpl
```

Ключевым здесь является то, что младший байт здесь должен не сдвигаться, а ротировать, чтобы "подхватить" содержимое флага переноса, образовавшееся в результате сдвига старшего байта.

94. Арифметический сдвиг вправо содержимого регистровой пары.

```
SRA rp      SRA rph
~~~~~~      RR rpl
```

95. Логический сдвиг влево 16-разрядного числа, находящегося по заданному адресу.

```
SLL (ADDR)  LD HL,ADDR
~~~~~~~~~~  SLA (HL)
            INC HL
```

```
                RL (HL)
```

ИЛИ

```
                LD xy,ADDR
                SLA (xy+0)
                RL (xy+1)
```

При сдвиге влево сначала сдвигается младший байт, а затем ротируется старший.

96. Логический сдвиг вправо 16-разрядного числа, находящегося по заданному адресу.

```
SRL (ADDR)      LD HL,ADDR+1
~~~~~~~~~~      SRL (HL)
                DEC HL
                RR (HL)
```

ИЛИ

```
                LD xy,ADDR
                SRL (xy+1)
                RR (xy+0)
```

97. Обмен полубайтов в аккумуляторе.

```
                RLCA                RRCA
                RLCA        ИЛИ     RRCA
                RLCA                RRCA
                RLCA                RRCA
```

98. Нормализация аккумулятора. Аккумулятор сдвигается влево до тех пор, пока его старший байт не будет равен 1. Не сдвигать, если в аккумуляторе содержится 0.

```
                AND A           - проверка аккумулятора на 0
                JP M,END        - выход, если старший бит = 1
                JR Z,END        - выход, если акк-р равен нулю
    SHIFT:      ADD A,A         - сдвиг влево
                JP P,SHIFT      - повторить сдвиг, если стар-
                                  ший бит равен нулю
    END:        NOP
```

99. Нормализация содержимого регистровой пары HL.

```
                LD A,H          - проверка содержимого
                OR L              регистровой пары на ноль
                JR Z,END        - выход, если 0
    SHIFT:      ADD HL,HL       - сдвиг влево
```

```
           JR NC,SHIFT   - сдвигать, пока не произойдет перенос
           RR H          - затем вернуть вправо на одну
           RR L             позицию
 END       NOP
```

Примечание: ADD HL,HL влияет на флаг переноса, но не влияет на флаг знака и флаг нуля.

2.2.6. Конструкции ротации.

100. Ротация регистровой пары вправо.

Проблема состоит в том, чтобы младший бит младшего байта был перенесен в старший бит старшего байта.

```
RR rp              RRC rpl       - младший бит копируется в C
~~~~~              RL rpl        - байт восстанавливается, но
                                   флаг C остался
                   RR rph        - ротация старшего байта
                   RR rpl        - ротация младшего байта
```

101. Ротация регистровой пары влево.

```
RL rp              RLC rph
~~~~~              RR rph
                   RL rpl
                   RL rph
```

102. Ротация регистровой пары вправо через флаг переноса.

```
RRC rp             RR rph
~~~~~~             RR rpl
```

103. Ротация регистровой пары влево через флаг переноса,

```
RLC rp             RL rph
~~~~~~             RL rpl
```

104. Ротация вправо 16-разрядного числа, находящегося по указанному адресу, через флаг переноса.

```
RRC (ADDR)         LD HL,ADDR+1
~~~~~~~~~~         RR (HL)
                   DEC HL
                   RR (HL)
```

ИЛИ

```
                   LD xy,ADDR
                   RR (xy+1)
                   RR (xy+0)
```

105. Ротация влево 16-разрядного числа, находящегося по указанному адресу, через флаг переноса.

```
RLC (ADDR)          LD HL,ADDR
~~~~~~~~~~          RL (HL)
                    INC HL
                    RL (HL)
```

ИЛИ

```
                    LD xy,ADDR
                    RL (xy+0)
                    RL (xy+1)
```

2.2.7. Конструкции для проверки регистров и адресов.

~~~~~~

106. Проверка аккумулятора. Выставляются флаги в соответствии с содержимым, которое при этом не изменяется.

```
                    AND A      или      OR A
```

Оба варианта сбрасывают флаг переноса.

107. Проверка состояния регистра с выставлением флагов.

```
                    INC reg
                    DEC reg
```

108. Проверка состояния заданного адреса.

```
                    LD HL,ADDR
                    INC (HL)
                    DEC (HL)
```

109. Проверка содержимого регистра.

```
                    LD A,rph
                    OR rpl
```

110. Проверка содержимого индексного регистра.

```
                    PUSH xy
                    POP rp
                    LD A,rph
                    OR rpl
```

111. Проверка 16-битного числа, находящегося по заданному адресу.

```
                    LD HL,(ADDR)
                    LD A,H
                    OR L
```
~~~~~~

2.3. КОНСТРУКЦИИ ДЛЯ ПЕРЕДАЧИ ДАННЫХ

В этом разделе мы рассмотрим конструкции, выполняющие следующие манипуляции с данными: копирование, хранение, перемещение, обмен, ввод, вывод, удаление и установку.

2.3.1. Конструкции копирования и загрузки данных.

112. Копирование из адреса в регистр.

```
LD reg,(ADDR)      LD A,(ADDR)
~~~~~~~~~~~~~      LD reg,A
ИЛИ
                   LD HL,ADDR
                   LD reg,(HL)
```

113. Загрузка числа во флаговый регистр.

```
LD F,N             LD rpl,N
~~~~~~             PUSH HL
                   POP AF
```

114. Копировать данные из адреса во флаговый регистр.

```
LD F,(ADDR)        LD HL,(ADDR)
~~~~~~~~~~~        PUSH HL
                   POP AF
```

115. Скопировать в HL данные из адреса, содержащегося в HL.

```
LD HL,(HL)         LD A,(HL)
~~~~~~~~~~         INC HL
                   LD H,(HL)
                   LD L,A
```

116. Загрузить в BC или DE данные из адреса, содержащегося в HL.

```
LD rp,(HL)         LD rpl,(HL)
~~~~~~~~~~         INC HL
                   LD rph,(HL)
                   DEC HL
```

2.3.2. Конструкции, предназначенные для сохранения данных.

117. Копировать содержимое регистра в память.

```
LD (ADDR),reg       LD A,reg
~~~~~~~~~~~~~       LD (ADDR),A
ИЛИ
                    LD HL,ADDR
                    LD (HL),reg
```

118. Копировать содержимое флагового регистра в память.

```
LD (ADDR),F         PUSH AF
~~~~~~~~~~~         POP HL
                    LD (ADDR),HL
ИЛИ
                    PUSH AF
                    POP HL
                    LD A,L
                    LD (ADDR),A
```

119. Копировать содержимое регистровой пары в ячейку памяти, адрес которой находится в HL.

```
LD (HL),rp          LD (HL),rpl
~~~~~~~~~~          INC HL
                    LD (HL),rph
                    DEC HL
```

2.3.3. Конструкции перемещения данных.

120. Перенос аккумулятора во флаговый регистр.

```
LD F,A              LD rpl,A
~~~~~~              PUSH rp
                    POP AF
```

121. Перенос флагового регистра в аккумулятор.

```
LD A,F              PUSH AF
~~~~~~              POP rp
                    LD A,rpl
```

122. Перенос данных из одной регистровой пары в другую.

```
LD rp2,rp1          LD rp2l,rp1l
~~~~~~~~~~          LD rp2h,rp1h
```

123. Перенос указателя стека в HL.

```
LD HL,SP            LD HL,0
~~~~~~~~            ADD HL,SP
```

124. Перенос указателя стека в индексный регистр.

```
LD xy,SP          LD xy,0
~~~~~~~~          ADD xy,SP
```

125. Перенос содержимого регистровой пары в индексный регистр.

```
LD xy,rp          PUSH rp
~~~~~~~~          POP xy
```

126. Перенос содержимого индексного регистра в регистровую пару.

```
LD rp,xy          PUSH xy
~~~~~~~~          POP rp
```

127. а) Перенос из IX в IY.

```
LD IY,IX          PUSH IX
~~~~~~~~          POP IY
```

б) Перенос из IY в IX.

```
LD IX,IY          PUSH IY
~~~~~~~~          POP IX
```

128. Перенос данных из HL в программный счетчик.

```
LD PC,HL          JP (HL)
~~~~~~~~
```

129. Перенос данных из индексного регистра в программный счетчик.

```
LD PC,xy          JP (xy)
~~~~~~~~
```

130. Перенос данных из двух байтов памяти в программный счетчик (косвенный переход).

```
LD PC,(ADDR)      LD HL,(ADDR)
~~~~~~~~~~~~      JP (HL)
```

131. Заполнение области памяти данными, содержащимися в аккумуляторе. Адрес начала области - в HL. Длина области - в регистре B.

```
          FILL:   LD (HL),A     - копирование акк-ра в память
                  INC HL        - переход к следующему адресу
                  DJNZ FILL     - повторить, если B<>0
```

2.3.4. Команды обмена.

132. Обмен между регистрами через аккумулятор.

```
EX reg1, reg2      LD A,reg1
~~~~~~~~~~~~~      LD reg1, reg2
                   LD reg2,A
```

133. Другие пути обмена между регистрами.

а) между BC и HL:

```
EX BC,HL           PUSH BC
~~~~~~~~           EX HL,(SP)
                   POP BC
```

б) между произвольными регистровыми парами:

```
EX reg1, reg2      PUSH rp1,
~~~~~~~~~~~~~      PUSH rp2
                   POP rp1
                   POP rp2
```

134. Обмен между указателями стека и HL.

```
EX SP,HL           EX DE,HL  - сохранить HL в DE
~~~~~~~~           LD HL,0   - поместить указатель стека
                   ADD HL,SP   в регистр HL
                   EX DE,HL  - восстановить HL
                   LD SP,HL  - поместить HL в SP
                   EX DE,HL  - восстановить в HL значение SP
```

Эта процедура может применяться, если пользователь организует свой машинный стек, отличный от системного, например, при одновременной работе с двумя программами. Они "прозрачны" друг для друга.

135. Обмен индексного регистра с регистровой парой.

```
EX xy,rp           PUSH xy
~~~~~~~~           PUSH rp
                   POP xy
                   POP rp
```

136. Обмен индексных регистров.

```
EX IX,IY           PUSH IX
~~~~~~~~           PUSH IY
                   POP IX
                   POP IY
```

2.3.5. Конструкции удаления данных.

137. Очистка аккумулятора.

SUB A или XOR A или LD A,0

Третий вариант работает медленнее, чем два первых и занимает больше памяти, но зато не влияет на флаги. Применяйте тот вариант, который Вам больше подходит.

138. Очистка регистра.

```
LD reg,0
```

139. Очистка заданной ячейки памяти.

```
SUB A
LD (ADDR),A
```

ИЛИ

```
LD HL,ADDR
LD (HL),0
```

Второй вариант более медленный, но зато не затрагивает ни содержимое аккумулятора, ни флаги.

140. Очистка флага переноса.

```
AND A
```

ИЛИ

```
OR A
```

ИЛИ

```
SCF
CCF
```

2.3.6. Конструкции установления заданного значения.

141. Установить в аккумулятре FF. (Аналог LD A,FF).

```
SUB A
DEC A
```

142. Установить FF в заданном адресе.

```
LD (ADDR),FF
```

```
LD A,FF
LD (ADDR),A
```

ИЛИ

```
LD HL,ADDR
LD (HL),FF
```

2.4. КОНСТРУКЦИИ ВЕТВЛЕНИЯ

2.4.1. Конструкции безусловного ветвления.

143. Переход к адресу, находящемуся на вершине стека.

```
JP (SP)          RET
~~~~~~~
```

144. Переход к адресу, находящемуся в DE.

```
JP (DE)          EX DE,HL
~~~~~~~          JP (HL)
```

145. Переход к адресу, находящемуся в BC.

```
JP (BC)          LD H,B
~~~~~~~          LD L,C
                 JP (HL)
```

или

```
                 PUSH BC
                 RET
```

Второй вариант значительно медленнее первого, но зато не затрагивает регистра HL.

146. Переход к адресу, находящемуся в заданном адресе.

```
JP (ADDR)        LD HL,(ADDR)
~~~~~~~~~        JP (HL)
```

147. Многовариантное ветвление. Предполагается, что есть таблица адресов, в которой находятся адреса переходов. Адрес начала этой таблицы находится в HL, а индекс, указывающий на порядковый номер элемента в таблице, находится в аккумуляторе.

```
ADD A,A    - индекс надо удвоить, т.к. элементами таблицы
             являются адреса, а они занимают по два байта.
LD E,A     - перевод индекса в 16-битную
LD D,0       форму
ADD HL,DE  - определяется адрес элемента
LD E,(HL)  - вводится сам элемент,
INC HL         состоящий их
LD D,(HL)          двух байтов
EX DE,HL   - ввод адреса перехода
JP (HL)    - переход по адресу, взятому из таблицы.
```

2.4.2. Конструкции условного ветвления.

147. Переход, если в аккумуляторе 0.

```
AND A          - проверка аккумулятора
JR Z,ADDR      - переход, если 0
```

148. Переход, если в регистре 0.

```
INC reg
DEC reg        - проверка регистра
JR Z,ADDR      - переход, если 0
```

149. Переход, если в заданном адресе 0.

```
LD HL,ADDR     - указание на адрес
INC (HL)
DEC (HL)       - проверка на 0
JR Z,ADDR1     - переход, если 0
```

ИЛИ

```
LD A,(ADDR)
AND A
JR Z,ADDR1
```

150. Переход, если в регистровой паре 0.

```
LD A,rph
OR rpl         - проверка на 0
JR Z,ADDR      - переход, если 0
```

151. Переход, если в индексном регистре 0.

```
PUSH xy
POP reg
LD A,rph
OR rpl
JR Z,ADDR
```

152. Переход, если двухбайтный адрес содержит 0.

```
LD HL,(ADDR)
LD A,H
OR L
JR Z,ADDR1
```

153. Переход, если заданный бит регистра равен 0.

```
BIT N,reg
JR Z,ADDR
```

154. Переход, если бит 7 аккумулятора равен 0.

```
AND A
JP P,ADDR
```

ИЛИ

```
RLA
JR NC,ADDR
```

155. Переход, если бит 6 аккумулятора равен нулю.

```
ADD A,A
JP P,ADDR
```

156. Переход, если бит 0 аккумулятора равен нулю.

```
RRA
JP NC,ADDR
```

157. Переход, если заданный бит в адресе равен нулю.

```
LD HL,ADDR
BIT N,(HL)
JR Z,ADDR1
```

158. Переход, если прерывания отключены.

```
LD A,I        - по этой команде состояние
                триггера прерываний
                передается во флаг четности
JP PO,ADDR
```

По образу конструкций перехода по нулю строятся аналогичные конструкции перехода, если в аккумуляторе, регистре, адресе и т.п. не ноль.

159. Переход, если содержимое аккумулятора равно заданному числу.

```
CP N
JR Z,ADDR
```

160. Переход, если в регистре 1.

```
DEC reg
JR Z,ADDR
```

Эту же процедуру можно применять и для проверки ячейки памяти с адресацией через HL или через индексный регистр.

161. Переход, если в регистре FF.

```
INC reg
JR Z,ADDR
```

Эту же процедуру можно применять и для проверки ячейки памяти с адресацией через HL или через индексный регистр.

162. Переход, если содержимое аккумулятора равно содержимому регистра.

```
CP reg
JR Z,ADDR
```

163. Переход, если содержимое аккумулятора равно содержимому заданного адреса.

```
LD HL,ADDR
CP (HL)
JR Z,ADDR1
```

164. Переход, если в регистровой паре содержится число NN.

```
LD HL,NN
AND A
SBC HL,rp
JR Z,ADDR
```

165. Переход, если в адресе, на который указывает HL, содержится число, равное содержимому регистра.

```
AND A
SBC HL,rp
JR Z,ADDR
```

166. Переход, если в указателе стека содержится заданное число.

```
LD HL,NN
AND A
SBC HL,SP
JR Z,ADDR
```

167. Переход, если содержимое указателя стека равно содержимому регистра HL.

```
AND A
SBC HL,SP
JR Z,ADDR
```

168. Переход, если содержимое индексного регистра равно двухбайтному числу.

```
PUSH xy
POP rp
LD HL,NN
AND A
SBC HL,rp
JR Z,ADDR
```

Аналогично строятся конструкции перехода по условию "не равно".

169. Переход, если содержимое аккумулятора положительное.

```
AND A
JP P,ADDR
```

170. Переход, если содержимое регистровой пары положительное.

```
INC rph
DEC rph
JP P,ADDR
```

171. Переход, если содержимое регистра положительное.

```
INC reg
DEC reg
JP P,ADDR
```

172. Переход, если содержимое адреса положительное.

```
LD HL,ADDR
INC (HL)
DEC (HL)
JP P,ADDR1
```

ИЛИ

```
LD A,(ADDR)
AND A
JP P,ADDR
```

173. Переход, если содержимое индексного регистра положительное.

```
PUSH xy
POP AF
AND A
JP P,ADDR
```

174. Переход, если двухбайтное число, находящееся в заданном адресе положительное.

```
LD A,(ADDR+1) - достаточно проверить старший
AND A           байт
JP P,ADDR1
```

ИЛИ

```
LD HL,ADDR+1
BIT 7,HL      - информация о знаке
JR Z,ADDR       содержится в старшем бите
```

Аналогичными конструкциями выполняются и переходы по отрицательному числу.

175. Переход, если содержимое аккумулятора больше, чем заданное число (без знака). Особый случай здесь возникает, если операнды равны. В этом случае переход не совершается.

```
        CP N
        JR C,END
        JR NZ,ADDR
END:    NOP
```

176. Переход, если содержимое аккумулятора больше содержимого регистра (без знака).

```
        CP reg
        JR C,END
        JR NZ,ADDR
END:    NOP
```

ИЛИ

```
        LD reg1,A
        LD A,reg
        CP reg1
        JR NC,ADDR
```

ИЛИ

```
        INC reg
        CP reg
        JR NC,ADDR
```

177. Переход, если содержимое аккумулятора больше содержимого заданного адреса (без знака).

```
        LD HL,ADDR
        CP (HL)
        JR C,END
        JR NZ,ADDR1
END:    NOP
```

ИЛИ

```
        LD reg,A
        LD A,(ADDR)
        CP reg
        JR C,ADDR1
```

178. Переход, если содержимое регистра HL больше содержимого регистровой пары (без знака).

```
SCF
SBC HL,rp
JR NC,ADDR
```

179. Переход, если содержимое HL больше, чем заданное 16-битное число (без учета знака).

```
LD rp,-NN-1
ADD HL,rp
JR C,ADDR
```

180. Переход, если содержимое указателя стека больше, чем содержимое HL (без знака).

```
AND A
SBC HL,SP
JR C,ADDR
```

181. Переход, если содержимое указателя стека больше заданного 16-битного числа (без знака).

```
LD HL,-NN-1
ADD HL,SP
JR C,ADDR
```

182. Переход, если содержимое индексного регистра больше заданного 16-битного числа (без знака).

```
LD rp,-NN-1
ADD xy,rp
JR C,ADDR
```

183. Переход, если содержимое индексного регистра больше содержимого HL (без знака).

```
PUSH xy
POP rp
AND A
SBC HL,rp
JR C,ADDR
```

184. Переход, если содержимое аккумулятора не больше заданного числа (без знака).

```
CP N          - сравнение
JR C,ADDR     - переход, если "меньше"
JR Z,ADDR     - переход, если "равно"
```

Другой вариант:

```
CP N+1
JR C,ADDR     - переход, если "НЕ МЕНЬШЕ"
```

185. Переход, если содержимое аккумулятора не больше содержимого заданного регистра (без знака).

```
CP reg
JR C,ADDR
JR Z,ADDR
```

ИЛИ

```
LD reg1,A     - запомнили содержимое ак-ра
LD A,reg      - взяли содержимое регистра
CP reg1       - сравнение
JR NC,ADDR    - переход, если "не больше"
```

ИЛИ

```
INC reg
CP reg
JR C,ADDR
```

186. Переход, если содержимое аккумулятора не больше содержимого заданного адреса (без учета знака).

```
LD HL,ADDR
CP (HL)
JR C,ADDR
JR Z,ADDR
```

ИЛИ

```
LD reg,A
LD A,(ADDR)
CP reg
JR NC,ADDR
```

187. Переход, если содержимое HL не больше, чем содержимое заданной регистровой пары (без знака).

```
SCF
SBC HL,rp
JR C,ADDR
```

188. Переход, если содержимое HL не больше, чем заданное число двойной длины (без учета знака).

```
LD rp, -NN-1
ADD HL,rp
JR NC,ADDR
```

189. Переход, если содержимое указателя стека не больше, чем содержимое регистра HL (без учета знака).

```
AND A
SBC HL,SP
JR NC,ADDR
```

190. Переход, если содержимое указателя стека не больше, чем заданное двухбайтное число (без знака).

```
LD HL,-NN-1
ADD HL,SP
JR NC,ADDR
```

191. Переход, если содержимое индексного регистра не больше, чем заданное двухбайтное число.

```
LD rp,-NN-1
ADD xy,rp
JR NC,ADDR
```

192. Переход, если содержимое индексного регистра не больше, чем содержимое регистра HL (без знака).

```
PUSH xy
POP rp
AND A
SBC HL,rp
JR NC,ADDR
```

193. Переход, если содержимое аккумулятора меньше, чем заданное число (без учета знака).

```
CP N
JR C,ADDR
```

194. Переход, если содержимое аккумулятора меньше содержимого регистра (без знака).

```
CP reg
JR C,ADDR
```

195. Переход, если содержимое аккумулятора меньше содержимого заданного адреса (без знака).

```
LD HL,ADDR
CP (HL)
JR C,ADDR
```

196. Переход, если содержимое регистра HL меньше содержимого заданного регистра (без учета знака).

```
AND A
SBC HL,rp
JR C,ADDR
```

197. Переход, если содержимое регистра HL меньше заданного двухбайтного числа (без учета знака).

```
LD rp,-NN
ADD HL,rp
JR NC,ADDR
```

198. Переход, если указатель стека меньше HL (без знака).

```
SCF
SBC HL,SP
JR NC,ADDR
```

199. Переход, если указатель стека меньше двухбайтного числа (без учета знака).

```
LD HL,-NN
ADD HL,SP
JR NC,ADDR
```

200. Переход, если содержимое индексного регистра меньше заданного двухбайтного числа (без учета знака).

```
LD rp,-NN
ADD xy,rp
JR NC,ADDR
```

201. Переход, если содержимое индексного регистра меньше содержимого регистра HL (без знака).

```
PUSH xy
POP rp
SCF
SBC HL,rp
JR NC,ADDR
```

202. Переход, если содержимое аккумулятора не меньше заданного числа (без знака).

```
CP N
JR NC,ADDR
```

203. Переход, если аккумулятор не меньше заданного регистра (без знака).

```
CP reg
JR NC,ADDR
```

204. Переход, если аккумулятор не меньше содержимого заданного адреса (без знака).

```
LD HL,ADDR
CP (HL)
JR NC,ADDR1
```

205. Переход, если содержимое регистра HL не меньше содержимого заданной регистровой пары (без учета знака).

```
AND A
SBC HL,rp
JR NC,ADDR
```

206. Переход, если содержимое регистра HL не меньше заданного двухбайтного числа (без знака).

```
LD rp,-NN
ADD HL,rp
JR C,ADDR
```

207. Переход, если указатель стека не меньше HL (без знака).

```
SCF
SBC HL,SP
JR C,ADDR
```

208. Переход, если указатель стека не меньше, чем заданное двухбайтное число (без учета знака).

```
LD HL,-NN
ADD HL,SP
JR C,ADDR
```

209. Переход, если содержимое индексного регистра не меньше, чем заданное двухбайтное число (без знака).

```
LD rp,-NN
ADD xy,rp
JR C,ADDR
```

210. Переход, если содержимое индексного регистра не меньше, чем содержимое HL (без знака).

```
PUSH xy
POP rp
SCF
SBC HL,rp
JR C,ADDR
```

2.5. КОНСТРУКЦИИ ВЫЗОВА ПОДПРОГРАММ

2.5.1. Безусловный вызов.

Дополнительно к прямым командам вызова подпрограмм, реализованным в системе команд процессора, программисты часто применяют непрямой вызов подпрограмм с помощью промежуточной вызывающей программы. В этом случае вызов выполняется вызовом этой промежуточной программы, например, CALL TRANS. Вместо TRANS, конечно стоит ее двухбайтный адрес. Далее процедура TRANS передает управление вызываемой программе. Это выполняется уже не через CALL, а через JP. Возврат выполняется из вызываемой подпрограммы, в конце которой стоит команда RET.

Типовые конструкции процедуры TRANS, в зависимости от того, как передается адрес вызываемой программы, приведены ниже.

211. Адрес вызова передается через HL.

```
TRANS:   JP (HL)
```

212. Адрес - в индексном регистре.

```
TRANS:   JP (xy)
```

213. Адрес - в DE.

```
TRANS:   EX DE,HL
         JP (HL)
```

214. Адрес - в BC.

```
TRANS:   LD H,B
         LD L,C
         JP (HL)
```

ИЛИ

```
         PUSH BC
         RET
```

Второй вариант работает медленнее, зато не портит HL.

215. Адрес в адресах ADDR и ADDR+1.

```
TRANS:   LD HL,(ADDR)
         JP (HL)
```

2.5.2. Условный вызов.

Команды вызова подпрограмм по условию могут быть расширены

так же, как мы это сделали с командами условного перехода. Единственное изменение, которое надо сделать - это поменять команды перехода на команды вызова. Например, вместо JR NZ,ADDR надо поставить CALL NZ,ADDR или вместо JP P,ADDR - CALL P,ADDR.

2.6. КОНСТРУКЦИИ ВОЗВРАТА

2.6.1. Безусловный возврат.

Инструкция RET автоматически выполняет переход в адрес, который находится на вершине стека. Если же Ваш адрес возврата находится где-либо в другом месте (например в регистровой паре или, скажем в двух ячейках памяти), то передать туда управление можно исполнением перехода с косвенной адресацией.

2.6.2. Возврат по условию.

Возврат по условию для Z-80 выполняется конструкциями, аналогичными конструкциям условного перехода. Надо только заменить мнемонику JR или JP на RET и не давать адрес перехода.

```
Например,     RET NC    вместо   JP NC,NN
              RET N     вместо   JP N,NN
```

2.6.3. Возврат со смещением.

Возможен случай, когда Вам надо выполнить возврат не точно по адресу, находящемуся на вершине стека, а несколько отступя от него, например, чтобы пропустить какие-либо данные, параметры, списки и т.п. неисполнимые вещи.

```
POP DE        адрес возврата помещается в DE
LD HL,NN      в HL загружается величина смещения
ADD HL,DE     вычисляется новый адрес возврата
JP (HL)       переход по рассчитанному адресу
```

Возможен вариант, когда Вы не хотите возвращаться по адресу, содержащемуся на вершине стека, а планируете переход по

другому адресу, скажем NN. Тогда можете действовать так:

```
LD HL,NN     загрузка в HL нового адреса возврата
EX HL,(SP)   обмен между HL и указателем стека. Теперь на
             стеке содержится новый адрес возврат.
```

2.6.4. Возврат после прерывания.

Поскольку обычно работа процедуры обработки прерываний начинается с того, что запоминается на стеке содержимое всех регистров, то и возврат после прерывания должен производиться с восстановлением этих регистров, причем порядок восстановления строго противоположен порядку сохранения. Например:

```
ВХОД В ПРЕРЫВАНИЕ  ВЫХОД ИЗ ПРЕРЫВАНИЯ
    PUSH AF          POP IY
    PUSH BC          POP IX
    PUSH DE          POP HL
    PUSH HL          POP DE
    PUSH IX          POP BC
    PUSH IY          POP AF
                     EI      разрешение  прерывания
                     RETI    возврат
```

2.7. ПРОЧИЕ КОНСТРУКЦИИ

2.7.1. Пауза.

Процессор имеет команду NOP, которая ничего не делает, кроме как увеличивает на единицу содержимое программного счетчика (регистр PC).

Того же результата можно добиться и другим путем. Вот список команд, которые тоже не выполняют никакие действия:

```
LD A,A      LD B,B      LD C,C      LD D,D
LD E,E      LD H,H      LD L,L
```

2.7.2. Работа со стеком.

216. Сохранение на стеке одиночного регистра.

PUSH reg	PUSH rp	сохранение регистровой пары
~~	INC SP	отбрасывание младшего байта

217. Сохранение на стеке содержимого ячейки памяти.

PUSH (ADDR)	LD A,(ADDR)	поместить в аккумулятор
~~		содержимое ячейки памяти
	PUSH AF	сохранение содержимого пары AF
	INC SP	отбросить содержимое флагового регистра

218. Сохранение на стеке содержимого двух идущих подряд ячеек памяти ADDR, ADDR+1.

	LD HL,(ADDR)	содержимое заданного адреса помещается в регистр HL
	PUSH HL	сохранение на стеке

219. Сохранение на стеке состояния триггера прерываний IFF

	LD A,I	после такой операции состояние триггера прерываний заносится во флаг P/O микропроцессора
	PUSH AF	сохранение флагового регистра

220. Вызов со стека содержимого одиночного регистра (при условии, что он был сохранен там так, как указано в п.217).

	DEC SP	указатель стека переводится на одну позицию вверх
	POP rp	вызов содержимого стека в регистровую пару (содержимое младшего байта пары остается неопределенным)

221. Вызов со стека содержимого одной ячейки памяти.

	DEC SP	
	POP AF	
	LD (ADDR),A	в результате работы этой конструкции непредсказуемо изменение флагов регистра F

222. Вызов со стека содержимого двух ячеек памяти.

	POP HL	
	LD (ADDR),HL	

223. Восстановление режима прерываний таким, каким он был до сохранения на стеке.

```
                POP AF          восстановление флагового
                                регистра
                JP PE,ON        переход на включение
                DI              запрещение прерываний
                JR EXIT         переход на "выход"
        ON:     EI              разрешение прерываний
        EXIT:   NOP              выход на процедуры
```

2.7.3. Конструкции десятиричной арифметики в двоичном выражении (BCD-арифметики).

224. Проверка на то, что в аккумуляторе содержится полноценное десятиричное число.

```
                LD reg,A        исполнение копии аккумулятора
                ADD A,0
                DAA             перевод содержимого
                                аккумулятора в BCD-арифметику
                CMP reg         аккумулятор изменился?
                JR NZ,NN        если изменился, то в нем было
                                не десятиричное число и
                                выполняется переход на
                                заданный адрес NN
```

225. Десятиричное приращение аккумулятора.

```
                ADD A,1
                DAA
```

226. Десятиричное уменьшение аккумулятора.

```
                SUB A,1
                DAA
```

Можно применять и такую конструкцию:

```
                ADD A,99
                DAA
```

2.7.4. Работа с таблицами.

227. Таблица однобайтных величин. В регистровой паре HL содержится базовый адрес таблицы, а в аккумуляторе - порядковый номер элемента в этой таблице. Требуется поместить в аккумулятор содержимое данного элемента.

```
LD E,A
LD D,0          аккумулятор конвертируется в
                16-битный индекс
ADD HL,DE       определен адрес элемента
LD A,(HL)       помещаем заданный элемент в A
```

Эта процедура обычно используется, когда надо произвести какую-либо перекодировку данных.

228. Таблица двухбайтных элементов. Базовый адрес находится В HL. В аккумуляторе - порядковый номер элемента. Требуется найти этот элемент и поместить его в регистровую пару HL.

```
EX DE,HL        базовый адрес переносится в DE
LD L,A
LD H,0          преобразование индекса в
                2-байтный
ADD HL,HL       удвоение индекса (необходимо
                потому, что в таблице
                двухбайтные числа и адрес
                элемента относительно начала
                таблицы вдвое больше номера
                элемента)
ADD HL,DE       расчет адреса элемента
LD E,(HL)       взять младший байт
INC HL          перейти к старшему байту
                таблицы
LD D,(HL)       взять старший байт
EX DE,HL        передать результат в HL
```

Примечание: здесь удвоение применялось инструкцией ADD HL,HL, в то время как можно было бы сразу дать ADD A,A. Дело в том, что ADD A,A работает в дополнительной двоичной арифметике, и потому результат будет верен только для байтов от 0 до 127, а ADD HL,HL позволяет использовать любое однобайтное число.

3. ДИРЕКТИВЫ АССЕМБЛЕРА

Одним из важнейших элементов этой части книги являются примеры программ в машинных кодах и на языке АССЕМБЛЕРа. В распечатках этих программ нам придется употреблять так называемые ДИРЕКТИВЫ АССЕМБЛЕРа и сейчас, наверное, самое удобное время для того, чтобы дать представление о том, что это такое.

Мы рассмотрим следующие директивы: ORG, EQU, DEFB, DEFW, DEFM и END, но прежде чем начать их рассмотрение, надо твердо для себя понять:

1. Директивы АССЕМБЛЕРа не являются командами процессора Z80 и в этом смысле отношения к машинному коду Z80 не имеют.

2. АССЕМБЛЕР - это программа, которая переводит (транслирует) текст, написанный Вами в виде мнемоник в объектный код, являющийся машинным. И эти директивы АССЕМБЛЕРа - это некоторые команды ассемблирующей программе. Они не транслируются и в объектный код не войдут, но упростят Вам написание, и самое главное - чтение программы, записанной в мнемониках.

3. Программ-АССЕМБЛЕРов существует великое множество и каждая из них может иметь свои собственные директивы. Они могут иметь и одинаковые директивы, но предъявлять разные требования к их употреблению. Одним словом, конкретно способы использования директив АССЕМБЛЕРа Вам надо устанавливать по инструкции к ассемблирующей программе, которой Вы пользуетесь (напр. EDITAS, GENS 3, GENS 4, ZEUS и т.п.). И хотя стандартов не существует, тем не менее некоторые основополагающие понятия все же выделить можно, вот на них-то мы и остановимся.

3.1. Комментарии.

Мы начнем с самого простого - с комментариев. Они записываются после символа ";" (точка с запятой).

Вам, конечно понятно, что все, что является комментариями, АССЕМБЛЕРом в машинный код не компилируется - это ни к чему. Они служат только для того, чтобы Вам было удобнее разбираться с листингом, который составил кто-то другой или Вы сами, но давным-давно.

Например:

```
 10        60001      LD E,A     ; Загрузили в регистр E содер-
 20                               ; жимое аккумулятора.
 30        60002      DEC E      ; Уменьшили его на единицу.
             ..........................................
```

Как видите, строка может состоять только из комментария.

3.2. Метки.

Метки значительно упрощают написание программ в мнемониках АССЕМБЛЕРа. В операциях перехода JP, JR, DJNZ, вызова подпрограмм CALL Вы можете не указывать адрес, в который Вы хотите совершить переход, а вместо него подставить метку. С другой стороны, когда будете писать команды для этого адреса, подставите метку и там, Например:

```
  10    60001   BEGIN   LD B,04
  20    60003   AGAIN   INC HL
  30    60004           DEC DE
  40    60005           DJNZ, AGAIN
  ..............................
 250    60110           LD A,(HL)
 260    60111           CP 80H
 270    60113           JR NZ,BEGIN
 ..............................
```

Как видите, очень удобно. Сразу видно, что из строки 40 возврат осуществляется к метке AGAIN, если регистр B не достиг нуля. Из строки 270 возврат осуществляется к метке BEGIN.

Определенно имеет смысл выбирать для метки такое имя, которое соответствовало бы смыслу исполняемой операции - это облегчает чтение и понимание листинга программы.

При компиляции ассемблирующая программа сама подсчитает величины необходимых смещений в командах процессора и подставит их вместо меток. Так, например, в строке 40 вместо DJNZ AGAIN в объектный код пойдет DJNZ FCH, что то же самое.

3.3. Директива EQU.

В предыдущем примере мы использовали метки очень ограниченно. Дело в том, что и обращение по метке и сама метка находились в одной и той же процедуре. А как быть, если Вы хотите обратиться к метке, которая находится в другой процедуре, которую Вы написали и откомпилировали еще вчера, а как быть, если Вам надо сделать переход к процедуре ПЗУ и Вы при этом хотите воспользоваться меткой? В этом случае Вам поможет директива EQU. Она присваивает метке числовое значение. Конечно, при компиляции эта директива никак в машинный код не преобразовывается, но если по тексту программы есть ссылки на эту метку, то вместо нее будет подставлено значение, взятое из директивы EQU.

Например, Вам в Вашей программе неоднократно приходится вызывать процедуры ПЗУ, скажем CLEAR (1EACH=7852) и OUT-LINE (1856H=6230). Тогда в начале Вашей программы Вы задаете директивой EQU значения своим меткам, например назвав их CLEAR и OUT_L.

```
10  CLEAR       EQU  7852
20  OUT_L       EQU  6230
30  LABEL       EQU  60016
```

Далее вызываете эти процедуры или делаете разные переходы по метке.

```
30  60001       LD HL,(LABEL)
40  60004       LD BC, 0008
50  60007       LD DE,(0452)
60  60010       CALL CLEAR
70  60013       CALL OUT_L
80  60016       ..................................
.....................................................
```

Сразу должны Вас предупредить, что приведенные здесь примеры с точки зрения программной бессмысленны. Это только примеры того, как используются те или иные директивы АССЕМБЛЕРа и если Вам нужен в примерах реальный смысл, то Вы его получите чуть позже, в последующих главах, где мы будем разбирать практические приемы программирования.

3.4. Директивы DEFB, DEFW, DEFM.

Давайте еще раз взглянем на предыдущий пример. В строке 30 мы засылаем в регистровую пару HL то, что содержится в адресе, на который указывает метка LABEL, а она, согласно директиве EQU указывает на адрес 60016.

Итак, в ячейках 60016 и 60017 содержатся некоторые данные, которые впоследствии могут использоваться программой. Эти данные Вы можете заслать в ячейки сами перед компиляцией. И совсем не надо для этого привлекать машинный код. Первоначальные значения в ячейках памяти Вы можете выставить с помощью директив DEFB, DEFW и DEFM.

DEFB - DEFINE BYTE - задать байт.
DEFW - DEFINE WORD - задать "слово" ("слово" - это два последовательно расположенных байта. Обычно это адрес.)
DEFM - DEFINE MESSAGE - задать сообщение (это несколько подряд идущих байтов). Обычно ассемблирующие программы накладывают ограничение на то, сколько байтов можно задать одной директивой DEFM, скажем не более пяти. Но Вас это не должно волновать. Если Вы хотите задать длинное сообщение, то можете ставить подряд столько строк DEFM, сколько хотите.

Итак, DEFB задает один одиночный байт (0...255), DEFW - два подряд идущих байта (0...65535), а DEFM - группу подряд идущих байтов - текстовое сообщение, числовая таблица и т.п.

В нашем предыдущем примере, если мы хотим хранить в адресе 60016 и 60017 некоторое двухбайтное число, строку 80 следовало бы записать, например так:

```
                ..................................
      80  60016     DEFW 5C92H
      90  60018 ..................................
```

Предположим, Вы хотите начиная с адреса 60135 хранить слово "Spectrum".

Вы можете его задать по байтам:

```
60135      DEFB  53H      ;Код буквы "S"
60136      DEFB  70H      ;Код буквы "p"
60137      DEFB  65H      ; "e"
60138      DEFB  63H      ; "c"
60139      DEFB  74H      ; "t"
60140      DEFB  72H      ; "r"
60141      DEFB  75H      ; "u"
60142      DEFB  6DH      ; "m"
```

Вы можете его задать парами байтов:

```
60135      DEFW  5370H    ; "Sp"
60137      DEFW  6563H    ; "ec"
60139      DEFW  7472H    ; "tr"
60141      DEFW  756DH    ; "um"
```

Но проще и правильнее задать его как сообщение:

```
60135      DEFM  5370656374   ; "Spect"
60140      DEFM  72756D       ; "rum"
```

Есть особый случай при программировании на АССЕМБЛЕРе, когда текст программы тоже приходится вводить через DEFB или DEFM. Это случай, когда Вы пишете программу для встроенного калькулятора. Ведь ассемблирующая программа может перевести в машинный код мнемоники АССЕМБЛЕРа, но она ничего не знает о кодах калькулятора и не знает его мнемоник. Код калькулятора - это внутреннее "Синклеровское" дело, его интерпретацией занимаются программы, размещенные в ПЗУ и к процессору и к его командам код калькулятора не имеет никакого отношения. Посему ввести команды калькулятору в ассемблирующую программу Вам удастся только как последовательность независимых байтов, т.е. через DEFB или DEFM.

Мы с Вами в первой части книги употребляли мнемонические обозначения команд калькулятора, типа add, stk_data s_lt и т.п., и писали их с маленькой буквы в отличие от команд процессора. Но делали это ранее и будем делать впредь только ради понимания и удобства записи. Программа-АССЕМБЛЕР таких мнемоник не знает, их нет в ее словаре.

Итак, с помощью DEFB, DEFW и DEFM задают начальные значения программным переменным, вводят в программу таблицы,

сообщения и любые прочие последовательности данных, даже графику, а также такие кодовые последовательности, которые ассемблирующая программа не понимает, как команды АССЕМБЛЕРа.

3.5. Директивы ORG, END.

Нам осталось рассмотреть две самые тривиальные директивы. Директива ORG объявляет адрес, начиная с которого будет ассемблироваться программа. Она должна быть первой директивой в исходном тексте, хотя в принципе, перед ней могут быть комментарии.

Вы обратили внимание на то, что в вышеприведенных примерах мы слева писали столбец адресов, в которых будут размещаться те или иные команды. Так вот, этого при программировании на АСЕМБЛЕРе делать не надо. Достаточно в самом начале дать директиву

```
10              ORG  63000
```

и далее ассемблирующая программа сама рассчитает в какой ячейке памяти будет находиться та или иная команда. Это очень упрощает процесс программирования. А если Вы внесете изменения в готовый текст, АССЕМБЛЕР сам подправит все адреса.

Директива END отмечает конец программы. Если после него что-то еще и будет стоять, то АССЕМБЛЕР при компиляции это проигнорирует.

Вот пожалуй и все, что для начала стоит знать о директивах АССЕМБЛЕРа. Это не все директивы, какие могут встретиться в жизни, да и правила их использования для разных АССЕМБЛЕРов - разные, но по большому счету этот минимум удовлетворит 90 процентов Ваших потребностей в информации, а остальное Вы должны почерпнуть из инструкции к тому АССЕМБЛЕРу, с которым работаете.

4. РАЗБОР ПРОГРАММ В МАШИННЫХ КОДАХ.

Для закрепления практических навыков мы предлагаем Вашему вниманию подробный разбор некоторых реальных процедур.

Мы рассмотрели в качестве примеров ответы на конкретные вопросы, которые в своих письмах задают нам наши читатели. Как организуется вывод на бордюр экрана цветных полос? Часто это можно видеть при загрузке фирменных игровых программ (например, "BOMB JACK", "INTERNATIONAL KARATE"). Как осуществляется в машинных кодах управление от джойстика и клавиатуры, как можно вывести текст на экран и др.?

4.1. Вывод на бордюр цветных полос.

Из части 1 Вы должны знать, что управление бордюром производится по трем младшим битам порта 254 (FE HEX). Идея организации вывода разноцветных полос состит в том, чтобы выдавать по этому порту команды на изменение бордюра, причем делать это надо с достаточной частотой. Если частота сигналов на изменение цвета много меньше кадровой частоты телевизионного изображения, то Вы увидите только мигание и изменение цвета бордюра (как в режиме ожидания загрузки от магнитофона). Если эта частота слишком высока, то полосы на экране будут узкими и частыми, так что даже их цвет трудно разобрать.

Мы на примере промоделируем возможность получения цветных полос различной ширины.

Логика работы программы такая. Процессор выбирает какой-либо байт из области ПЗУ, проверяет в нем три младших бита, устанавливает новый цвет бордюра, равный значению этих битов (от 0 до 7) и далее выдерживает паузу, тем большую, чем больше величина поступившего байта. После этого выполняется прием следующего байта на ПЗУ и т.д.

При загрузке программ типа "INTERNATIONAL KARATE" то же самое происходит при приеме очередного байта из магнитофонного порта. Мы, к сожелению, лишены возможности рассмотреть здесь подробно как обрабатывается сигнал, поступивший от магнитофона, поскольку это выполняет пакет процедур, содержащихся в ПЗУ, что

выходит за рамки данной книги и рассмотрено подробно в статье "Секреты ПЗУ" в "ZX-РЕВЮ" N 4,5 за 1991 г. и в статье "Защита программ" в "ZX-РЕВЮ" N1,2 за 1993 г. Поэтому рассмотрим управление бордюром на модельном примере. Укажем только, что основная сложность при написании процедур нестандартной загрузки состоит в синхронизации работы процедуры с частотой поступления импульсов от магнитофонного порта, что достигается тщательным расчетом замедляющих циклов.

Процедура ассемблирована, начиная с адреса 26000 (6590HEX) и может запускаться RANDOMIZE USR 26000.

АДРЕС	МАШ.КОД	МНЕМОНИКА		КОММЕНТАРИЙ
6590	11FF3F	LD	DE,3FFF	В DE помещается длина области ПЗУ
6593	210000	LD	HL,0000	В HL - начало ПЗУ
6596	7E	LD	A,(HL)	В аккумулятор идет байт из ПЗУ
6597	E607	AND	07	Старшие 5 битов маскируются (гасятся), остаются младшие три
6599	D3FE	OUT	(FE),A	Акк-р выдается по 254-му порту
659B	7E	LD	A,(HL)	Повторно в акк-р загружается тот же байт из ПЗУ
659C	47	LD	B,A	и переносится в регистр B
659D	10FE	DJNZ	FE	Замедляющий цикл, повторяющийся тем дольше, чем больше поступивший байт.
659F	23	INC	HL	Переход на новый байт ПЗУ
65A0	1B	DEC	DE	Уменьшение счетчика байтов
65A1	7A	LD	A,D	Проверка на то, что в счетчике
65A2	B3	OR	E	(DE) достигнут 0
65A3	20F1	JR	NZ,F1	Если нет, то переход назад и повторение для очередного байта
65A5	C9	RET		Выход из процедуры

Несколько слов о том, как следует практиковаться с нашими процедурами.

Конечно, лучше всего набирать их в программе АССЕМБЛЕР (все равно в каком). Это позволяет избежать множества мелких ошибок при наборе процедуры. Набрав текст, надо дать команду на компиляцию и после устранения выявленных при компиляции неточ-

ностей, можно выгрузить машинный код и работать с ним.

Для небольших процедур (до 100...200 байтов) в принципе возможен набор процедуры последовательным введением байтов одного за другим из какого-либо АССЕМБЛЕРа, но учебная ценность такого подхода ниже, зато есть возможность в случае появления ошибки использовать отладочные возможности дисассемблера. Только не забудьте перед пробным запуском отгрузить введенный машинный код на ленту. Вероятность ошибки при вводе достаточно большая и обидно повторять ввод второй раз.

Для совсем маленьких процедур типа той, что мы привели выше, можно выполнить ввод и из БЕЙСИКа. При этом можно перевести шестнадцатиричный код в десятиричный и вводить его через FOR...READ...DATA...POKE...NEXT, а можно использовать какую-либо БЕЙСИК-программу шестнадцатиричного загрузчика. Шестнадцатиричным загрузчиком вводят иногда и очень большие блоки кодов (до нескольких килобайт), однако имейте в виду, что набор больших блоков кодов (более 256 байтов) очень сложен и крайне необходимо, чтобы вводимый блок имел контрольные суммы после каждой группы (из 8-ми или 16-ти) байтов и применяемый шестнадцатириный загрузчик должен выполнять проверку этих контрольных сумм.

Для тех, кто не хочет утруждать себя применением АССЕМБЛЕРа или ДИСАССЕМБЛЕРа, в порядке исключения дадим распечатку загрузчика данной процедуры из БЕЙСИКа.

```
10 DIM a(22)
20 FOR i=1 TO 22
30 READ a(i)
40 POKE (25999+i),a(i)
50 NEXT i
60 DATA 17,255,63,33
70 DATA 0,0,126,230
80 DATA 7,211,254,126
90 DATA 71,16,254,35
100 DATA 27,122,179,32
110 DATA 241,201
120 RANDOMIZE USR 26000
```

4.2 Вывод данных на экран из машинного кода.

В части 1 мы уже говорили о том, что для печати символов на экране используется команда процессора RST 010 (код D7 HEX). Она вызывает исполнение системной процедуры из ПЗУ, которая и выполнит печать нужного Вам символа. Правда, прежде, чем обращаться к RST010, Вы должны предусмотреть, чтобы код того символа, который Вы хотите напечатать, был помещен в аккумулятор. Коды всех символов, управляющих кодов, символов графики пользователя, токенов ключевых слов Вы можете найти в "Справочнике..." (ч.3, табл.1).

Для того, чтобы выполнить печать, надо предварительно решить три вопроса:

- что печатать (какой символ, найти его код);
- как печатать (каким цветом, на каком фоне);
- где печатать;

Вы знаете, что экран "Спектрума" имеет 24 строки по 32 позиции, причем нижние две строки являются системным окном для вывода сообщений от операционной системы и печать в этих строках требует специальных ухищрений, а лучше пока там не печатать.

Все эти вопросы решаются помещением в аккумулятор нужного кода и вызовом RST 010.

Рассмотрим конкретный пример. Допустим, Вы хотите напечатать в центре экрана фразу "ZX-Spectrum", причем выполнить ее желтым цветом по синему фону.

Это значит, что позиция начала печати должна быть AT 11,10 -двенадцатая строка сверху (с учетом нулевой) и одиннадцатая колонка слева.

Цветовые коды должны быть установлены: INK = 6 (желтые буквы); PAPER = 1 (синий фон).

Но прежде, чем начинать печать на экране, надо открыть канал экрана и очистить экран.

Канал открывается вызовом системной процедуры ПЗУ "CHAN_OPEN", находящейся по адресу 1601, а очистка экрана - процедурой "Clear LINES", находящейся по адресу 0E44.

```
АДРЕС МАШ.КОД   МНЕМОНИКА    КОММЕНТАРИЙ

6502  3E02      LD   A,02
6592  CD0116    CALL 1601     Открываем канал "S"
6595  0618      LD   B,18     Количество очищаемых строк (24DEC)
6597  CD440E    CALL 0E44     Очистка 24-х строк
659A  3E16      LD   A,16     В акк-р помещаем управл. код AT за
                              которым пойдут координаты позиции
                              печати
659C  D7        RST  10       Вывод кода AT
659D  3E0B      LD   A,0B     Установка номера строки (11)
659F  D7        RST  10       Ввод номера строки
65A0  3E0A      LD   A,0A     Установка номера столбца
65A2  D7        RST  10       Ввод номера столбца
65A3  3E11      LD   A,11     Установка кода PAPER
65A5  D7        RST  10       Ввод кода PAPER
65A6  3E10      LD   A,01     Установка цвета фона
65A8  D7        RST  10       Ввод цвета фона (синий = 1)
65A9  3E10      LD   A,01     Установка кода INK
65AB  D7        RST  10       Ввод кода INK
65AC  3E06      LD   A,06     Установка цвета символов
65AE  D7        RST  10       Ввод цвета символов (желтый = 6)
65AF  3E5A      LD   A,5A     Установка кода буквы Z
65B1  D7        RST  10       Печать буквы "Z"
65B2  3E58      LD   A,58     Установка кода буквы "X"
65B4  D7        RST  10       Печать буквы "X"
65B5  3E2D      LD   A,2D     Установка кода знака "-"
65B7  D7        RST  10       Печать знака "-"
65B8  3E73      LD   A,73     Установка кода буквы "s"
65BA  D7        RST  10       Печать знака "s"
65BB  3E70      LD   A,70     Установка кода буквы "p"
65BD  D7        RST  10       Печать буквы "p"
65BE  3E65      LD   A,65     Установка кода буквы "e"
65C0  D7        RST  10       Печать буквы "e"
65C1  3E63      LD   A,63     Установка кода буквы "c"
65C3  D7        RST  10       Печать буквы "c"
65C4  3E74      LD   A,74     Установка кода буквы "t"
```

```
АДРЕС МАШ.КОД    МНЕМОНИКА     КОММЕНТАРИЙ
65C6  D7         RST 10        печать буквы "t"
65C7  3E72       LD  A,72      Установка кода буквы "r"
65C9  D7         RST 10        Печать буквы "r"
65CA  3E75       LD  A,75      Установка кода буквы "u"
65CC  D7         RST 10        Печать буквы "u"
65CD  3E6D       LD  A,6D      Установка кода буквы "m"
65CF  D7         RST 10        Печать буквы "m"
65D0  C9         RET           Возврат
```

Запускается данная процедура RANDOMIZE USR 26000.

Теперь мы можем поэкспериментировать с кодами управления печатью. Вы, конечно, знаете, какую роль играет разделитель <,> (ЗАПЯТАЯ) в операторе БЕЙСИКа PRINT. Она вызывает печать того, что за ней стоит на другой половине экрана. Код этого управляющего символа 06. Попробуйте заменить по адресу 65B6 код 2D (знак минус) на код 06 и снова запустить процедуру RANDOMIZE USR 26000.

Теперь попробуйте поставить туда же код 0D (ENTER, он же CR - CARRIDGE RETURN = ВОЗВРАТ КАРЕТКИ, он же "КОНЕЦ СТРОКИ") и опять стартуйте процедуру.

Если все, что Вы сделали, выглядит для Вас понятным, но достаточно скучным, то отметим, что в ПЗУ "Спектрума" есть и более мощная процедура для печати. Она позволяет печатать сразу целые строки ("стринги"), называется PR-STRING и находится по адресу 203C. Это процедура более высокого уровня, чем RST 10 и при своей работе использует обращения к RST 10, как к подпрограмме.

Чтобы использовать эту процедуру, надо в регистр DE загрузить адрес, в котором начинается тот текст, котрый Вы хотите выдать на печать (с управляющими кодами AT, TAB, INK, PAPER и др.). Длина этой строки помещается в пару BC, а вызывается процедура CALL 203C.

Для справки мы даем здесь текст этой процедуры.

```
АДРЕС МАШ.КОД    МНЕМОНИКА     КОММЕНТАРИЙ
203C  78         LD   A,B      Это проверка BC на 0, т.е. напе-
203D  B1         OR   C        чатана вс строка или нет
```

```
АДРЕС МАШ.КОД   МНЕМОНИКА    КОММЕНТАРИЙ

203E  0B        DEC BC       Уменьшение счетчика на единицу.
203F  C8        RET Z        Если в счетчике был 0, т.е. вся
                             строка напечатана, то выход.
2040  1A        LD A,(DE)    Загрузка в акк-р очередного
                             символа из Вашей строки.
2041  13        INC DE       Переход к очередному символу.
2042  D7        RST 10       Печать символов.
2043  18F7      JR 203C      Переход для повтора.
```

Давайте с помощью этой процедуры распечатаем ту же фразу ZX-spectrum с теми же атрибутами в том же месте экрана.

Длина нашей строки = 11 печатных символов + 3 кода управления (AT, INK, PAPER) + 2 числа установки позиции начала печати + 2 числа установки цветовых атрибутов = 18 (12HEX).

```
АДРЕС МАШ.КОД   МНЕМОНИКА    КОММЕНТАРИЙ

6590  011200    LD BC,0012   В BC загружается длина строки.
6593  119A65    LD DE,659A   В DE - адрес начала строки.
6596  CD3C20    CALL 203C    Вызов процедуры печати строки.
6599  C9        RET          Возврат в вызывающую программу.
659A  16                     "AT"
659B  0B                     Строка 11.
659C  0A                     Столбец 10.
659D  11                     "PAPER"
659E  01                     Цвет синий.
659F  10                     "INK"
65A0  06                     Цвет желтый.
65A1  5A                     Z
65A2  58                     X
65A3  2D                     -
65A4  73                     s
65A5  70                     p
65A6  65                     e
65A7  63                     c
65A8  74                     t
```

```
АДРЕС МАШ.КОД   МНЕМОНИКА     КОММЕНТАРИЙ
65A9  72                      r
65AA  75                      u
65AB  6D                      m
```

4.3. Управление программой от Кемпстон-джойстика.

В части 1 мы уже писали о протоколе обмена Кемпстон-джойстика. Он опрашивается по внешнему порту 31(1F) командой IN 31. Принимаемый байт анализируется так:

Бит 0 - движение вправо Бит 3 - движение вверх
Бит 1 - движение влево Бит 4 - "огонь"
Бит 2 - движение вниз Биты 5...7 не используются

В качестве примера рассмотрим программу, которая будет рисовать на экране синие квадраты на белом поле, если кнопка не нажата. При нажатии этой кнопки производится выход из программы. Начальная позиция печати - AT (11,15) - в центре экрана.

Поскольку программа выглядит несколько громоздкой, то мы в тексте используем метки.

```
Адрес Метка Маш.код  Мнемоника      Комментарий
6590        3E02     LD   A,02
6592        CD0116   CALL 1601       Открываем канал для печати
                                     на экране
6595        0618     LD   B,18
6597        CD440E   CALL 0E44       Очистка экрана
659A        3E0F     LD   A,0F
659C        322B66   LD   (XPOS),A   Установка начальной колонки
                                     печати (15) в переменной
                                     XPOS, размещенной по адресу
                                     662B
659F        3E0B     LD   A,0B
65A1        322C66   LD   (YPOS),A   Установка начальной строки
                                     печати в YPOS
65A4        CD1266   CALL PRINT      Вызов процедуры печати
                                     символа
65A7  REP   DB1F     IN   A,(1F)     Прием байта от джойстика.
```

Адрес	Метка	Маш.код	Мнемоника		Комментарий
65A9		E61F	AND	1F	Выделение 5 младших битов
65AB		28FA	JR	Z,REP	Возврат, если джойстик не был тронут
65AD		DEFF	LD	C,FF	Установка параметров замедляющего цикла 1
65AF	HERE	00	NOP		
65B0		06FF	LD	B,FF	Установка параметра замедляющего цикла 2
65B2	PAUSE	10FE	DJNZ	PAUSE	Второй замедляющий цикл
65B4		0D	DEC	C	
65B5		20F8	JR	NZ,HERE	Первый замедляющий цикл
65B7		CB67	BIT	4,A	Проверка кнопки "огонь"
65B9		C0	RET	NZ	Выход в вызвающую програм-му, если нажата
65BA		CB47	BIT	0,A	Проверка контакта "вправо"
65BC		200C	JR	NZ,RIGHT	Переход на подпрограмму движения вправо
65DE		CB4F	BIT	1,A	Проверка контакта "влево"
65C0		201A	JR	NZ,LEFT	Переход на подпрограмму движения влево
65C2		CB57	BIT	2,A	Проверка контакта "вниз"
65C4		2028	JR	NZ,DOWN	Переход на подпрограмму движения вниз
65C6		CB5F	BIT	3,A	Проверка контакта "вверх"
65C8		2036	JR	NZ,UP	Переход на подпрограмму движения вверх
65CA	RIGHT	3A2B66	LD	A,(XPOS)	В акк-р идет последний столбец печати
65CD		3C	INC	A	и увеличивается на 1
65CE		FE1F	CP	1F	Проверка на достижение правого края
65D0		CAA765	JP	Z,REP	Если да,то возврат на опрос джойстика
65D3		322B66	LD	(XPOS),A	Если нет, то запомнить новый столбец
65D6		CD1266	CALL	PRINT	И вызвать подпрограмму печати

```
Адрес Метка Маш.код  Мнемоника      Комментарий
65D9        C3A765   JP   REP       Возврат на новый опрос
                                    джойстика
65DC  LEFT  3A2B66   LD   A,(XPOS)  Ввод текущего столбца
                                    печати
65DF        3D       DEC  A        Уменьшить номер столбца на 1
65E0        FE00     CP   00       Проверка на достижение
                                    левого края
65E2        CAA765   JP   Z,REP    Если да, то возврат на
                                    опрос джойстика
65E5        322B66   LD   (XPOS),A Если нет, запомнить новый
                                    столбец
65E8        CD1266   CALL PRINT    Вызов подпрограммы печати
65ED        C3A765   JP   REP      Возврат на опрос джойстика
65EE  DOWN  3A2C66   LD   A,(YPOS)  В акк-р идет последняя
                                    строка печати
65F1        3C       INC  A        и увеличивается на 1
65F2        FE16     CP   16       Проверка на нижний край
65F4        CAA765   JP   Z,REP    Если да, то возврат на
                                    опрос джойстика
65F7        322C66   LD   (YPOS),A Если нет, то запомнить
                                    новую строку и вызвать
65FA        CD1266   CALL PRINT    подпрограмму печати
65FD        C3A765   JP   REP      Возврат на новый опрос
                                    джойстика
6600  UP    3A2C66   LD   A,(YPOS)  Ввод текущей строки печати
6603        3D       DEC  A        Уменьшить номер строки на 1
6604        FE00     CP   00       Проверка на достижение
                                    верхнего края
6606        CAA765   JP   Z,REP    Если да, то возврат на
                                    опрос джойстика
6609        322C66   LD   (YPOS),A Если нет, запомнить новую
                                    строку

660C        CD1266   CALL PRINT    Вызов подпрограммы печати
660F        C3A765   JP   REP      Возврат на опрос джойстика
6612  PRINT 06FF     LD   B,FF     Установка параметра
                                    замедляющего цикла 3
6614  WAIT  10FE     DJNZ WAIT     Третий замедляющий цикл
```

```
6616          3E16      LD    A,16       Управляющий код AT
6618          D7        RST   10
6619          3A2C66    LD    A,(YPOS)   Позиция печати по вертикали
661C          D7        RST   10
661D          3A2B66    LD    A,(XPOS)   Позиция печати по горизонт.
6620          D7        RST   10
6621          3E10      LD    A,10       Управляющий код INK
6623          D7        RST   10
6624          3E01      LD    A,01       Цвет символа - синий
6626          D7        RST   10
6627          3E8F      LD    A,8F       Символ ▉ (код 143)
6629          D7        RST   10
662A          C9        RET              Возврат в вызывающую
                                         программу
662B  XPOS    00        DEFB  00         Координата X
662C  YPOS    00        DEFB  00         Координата Y
```

4.4. Управление программой от клавиатуры.

Это, без сомнения, тоже задача повседневной потребности. Сразу укажем на то, что существуют два различных метода опроса клавиатуры - один связан с приемом и анализом данных, поступающих от внешнего порта, а другой - с использованием системных переменных KSTATE или LASTK. Мы здесь рассмотрим оба метода, т.к. оба достаточно часто применяются в программах.

Рассмотрим задачу, аналогичную предыдущей. Допустим, нам надо, чтобы на экране рисовались синие квадраты при нажатии следующих клавиш:

"P" - вправо "Q" - вверх

"O" - влево "A" - вниз

"0" - выход из программы

Для усложнения задачи введем еще клавишу "ПРОБЕЛ" в качестве переключателя режимов "РИСОВАНИЕ/СТИРАНИЕ".

МЕТОД 1

Опрос клавиатуры производится по внешнему порту 254 (FE HEX). Но, поскольку адрес порта клавиатуры является двухбайтным

числом, то FE - является младшим байтом адреса порта, а старший должен быть установлен предварительно в аккумуляторе.

Анализируются 5 младших битов. При этом если клавиша не нажата, то бит включен (в отличие от опроса Кемпстон-джойстика). Поэтому, если ни одна клавиша не нажата, то должно бы выдаваться число 255, но оно не выдается, а выдается 191. Дело в том, что по 6-му биту анализируется состояние порта входа от магнитофона и если сигнал не поступает, то 6-ой бит выключен и поэтому Вы получаете 191, а не 255 (255-64 = 191).

Адрес	Метка	Маш.код	Мнемоника	Комментарий
6590		3E02	LD A,02	
6592		CD0116	CALL 1601	Открываем канал для печати на экране
6595		0618	LD B,18	
6597		CD440E	CALL 0E44	Очистка экрана
659A		3E0F	LD A,0F	
659C		325766	LD (XPOS),A	Установка начальной колонки печати в переменной XPOS
659F		3E0B	LD A,0B	
65A1		325666	LD (YPOS),A	Установка начальной строки печати в YPOS
65A4		3E8F	LD A,8F	Аккумулятор указывает на символ ▉ (143)
65A6		325A66	LD (SIM),A	Код символа отправляется в переменную SYM.
65A9		CD4766	CALL PRINT	Вызов процедуры печати символа
65AC	REP	0E4F	LD C,4F	Установка параметра замедляющего цикла 1. Он взят меньше, чем в программе для джойстика, поэтому рисование здесь пойдет быстрее.
65AE	HERE	06FF	LD B,FF	Установка параметра замедляющего цикла 2.
65B0	PAUSE	10FE	DJNZ PAUSE	Второй замедляющий цикл
65B2		0D	DEC C	
65B3		20F9	JR NZ,HERE	Первый замедляющий цикл

Адрес	Метка	Маш.код	Мнемоника		Комментарий
65B5		3EEF	LD	A,EF	Указание на 5-ый полуряд
65B7		DBFE	IN	A,(FE)	Опрос 5-го полуряда
65B9		CB47	BIT	0,A	Проверка клавиши "0"
65BB		C8	RET	Z	Выход в БЕЙСИК, если нажата
65BC		3E7F	LD	A,7F	Указание на 8-ой полуряд
65BE		DBFE	IN	A,(FE)	Опрос 8-го полуряда
65C0		CB47	BIT	0,A	Проверка клавиши "ПРОБЕЛ"
65C2		CC2C66	CALL	Z,TOGGL	Вызов подпрограммы на пере- ключение печатаемого симво- ла для обеспечения стирания
65C5		3EDF	LD	A,DF	Указание на 6-й полуряд
65C7		DBFE	IN	A,(FE)	Опрос 6-го полуряда
65C9		CB47	BIT	0,A	Проверка клавиши "P"
65CB		2817	JR	Z,RIGHT	Вправо, если нажата
65CD		CB4F	BIT	1,A	Проверка клавиши "O"
65CF		2825	JR	Z,LEFT	Влево, если нажата
65D1		3EFD	LD	A,FD	Указание на 2-й полуряд
65D3		DBFE	IN	A,(FE)	Опрос 2-го полуряда
65D5		CB47	BIT	0,A	Проверка клавиши "A"
65D7		282F	JR	Z,DOWN	Вниз, если нажата
65D9		3EFB	LD	A,FB	Указание на 3-й полуряд
65DB		DBFE	IN	A,(FE)	Опрос 3-го полуряда
65DD		CB47	BIT	0,A	Проверка клавиши "Q"
65DF		2839	JR	Z,UP	Вверх, если нажата
65E1		C3AC65	JP	RET	Если ни одна клавиша не нажата, то сначала
65E4	RIGHT	3A5766	LD	A,(XPOS)	в акк-р идет последний столбец печати и
65E7		3C	INC	A	увеличивается на 1
65E8		FE1F	CP	1F	Проверка на достижение правого края
65EA		CAAC65	JP	Z,REP	Если да, то возврат на опрос клавиатуры
65ED		325766	LD	(XPOS),A	Если нет, то запомнить новый столбец и вызвать
65F0		CD4766	CALL	PRINT	подпрограмму печати

Адрес	Метка	Маш.код	Мнемоника		Комментарий
65F3		C3AC65	JP	REP	Возврат на новый опрос клавиатуры
65F6	LEFT	3A5766	LD	A,(XPOS)	Ввод текущего столбца печати
65F9		3D	DEC	A	Уменьшить его номер на 1
65FA		FE00	CP	00	Проверка на достижение левого края
65FC		CAAC65	JP	Z,REP	Если да, то возврат на опрос клавиатуры
65FF		325766	LD	(XPOS),A	Если нет, запомнить новый столбец
6602		CD4766	CALL	PRINT	Вызов подпрограммы печати
6605		C3AC65	JP	REP	Возврат на опрос клавиатуры
6608	DOWN	3A5666	LD	A,(YPOS)	В акк-р идет последняя
660B		3C	INC	A	строка печати и увеличивается на 1
660C		FE16	CP	16	Проверка на достижение нижнего края
660E		CAAC65	JP	Z,REP	Если да, то возврат на опрос клавиатуры
6611		325666	LD	A,(YPOS)	Если нет, то запомнить
6614		CD4766	CALL	PRINT	новую строку и вызвать подпрограмму печати
6617		C3AC65	JP	REP	Возврат на новый опрос клавиатуры
661A	UP	3A5666	LD	A,(YPOS)	Ввод текущей строки печати
661D		3D	DEC	A	Уменьшить номер строки на 1
661E		FE00	CP	00	Проверка на достижение верхнего края
6620		CAAC65	JP	Z,REP	Если да, то возврат на опрос клавиатуры
6623		325666	LD	A,(YPOS)	Если нет, то запомнить новую строку
6626		CD4766	CALL	PRINT	Вызов подпрограммы печати
6629		C3AC65	JP	REP	Возврат на опрос клавиатуры
662C	TOGGL	3E7F	LD	A,7F	Вновь проверяем нажатие клавиши "ПРОБЕЛ", т.к. она

```
Адрес Метка Маш.код  Мнемоника      Комментарий
662E        DBFE     IN   A,(FE)    теперь должна быть отпущена
6630        CB47     BIT  0,A       иначе режимы будут
                                    переключаться многократно и
                                    выход будет неопределенным
6632        28F8     JR   Z,TOGGL   Возврат, если клавиша еще
                                    не отпущена
6634        3A5A66   LD   A,(SYM)   Чтобы переключить режим,
                                    надо сначала определить,
6637        FE8F     CP   8F        какой символ установлен -
6639        2806     JR   Z,NEXT    синий квадрат (8F) или
                                    белый (80)
663B        3E8F     LD   A,8F      Поскольку был не синий, то
663D        325A66   LD   (SYM),A   устанавливаем в переменную
                                    SYM код синего квадрата
6640        C9       RET            Возврат
6641  NEXT  3E80     LD   A,80      И, наоборот, раз был синий,
6643        325A66   LD   (SYM),A   то теперь устанавливаем
                                    белый квадрат ("СТИРАНИЕ")
6646        C9       RET            Возврат
6647  PRINT 06FF     LD   B,FF      Организация третьего
                                    замедляющего цикла
6649  WAIT  10FE     DJNZ WAIT      Третий замедляющий цикл
664B        115566   LD   DE,TEXT   Регистр DE указывает на
                                    печатный текст
664E        010600   LD   BC,0006   Длина текста - 6 символов
6651        CD3C20   CALL 203C      Вызов процедуры печати
                                    строки из ПЗУ
6654        C9       RET            Возврат в вызывающую
                                    процедуру
6655  TEXT  16       DEFB 16        Символ управления печатью
                                    AT
6656  YPOS  00       DEFB 00        Позиция печати по горизонт.
6657  XPOS  00       DEFB 00        Позиция печати по вертик.
6658        10       DEFB 10        Управляющий символ INK
6659        01       DEFB 01        Цвет символа - синий
665A  SYM   00       DEFB 00        Код печатаемого символа.
```

МЕТОД 2

Можно проводить управление программой от клавиатуры и не опрашивая внешние порты, связанные с ней, а используя некоторые системные переменные "Спектрума". Поскольку программа при этом очень похожа на предыдущую, то мы ее текст приводить не будем, но некоторые существенные особенности имеются.

1. Использование системной переменной LAST K.

Эта системная переменная находится по адресу 23560. В ней запоминается код символа (токена), находящегося на последней нажатой клавише. Причем, здесь есть различия, в зависимости от того, в каком клавиатурном регистре эта клавиша была нажата. Например, после нажатия клавиши "P" там будет помещен код 70HEX (символ "p"), при нажатии CAPS SHIFT "P" - код 50HEX (символ "P"), а при нажатии SYMB SHIFT "P" - код 22HEX (символ").

Итак, первой особенностью является то, что одна и та же клавиша при использовании ее в разных режимах дает разные коды. поскольку в программе с равной вероятностью клавиатура может быть в режиме CAPS LOCK и в обычном режиме, то при определении перехода надо проверять оба кода. Например, для клавиши "P" - переход на процедуру обработки движения вправо:

```
.................
CP        70
JR        Z,RIGHT
CP        50
JR        Z,RIGHT
.................
```

2. Вторая особенность состоит в том, что системная переменная "помнит" факт нажатия последней клавиши. Если при опросе клавиатуры через внешний порт мы имели сигнал только пока клавиша нажата, то здесь код сидит в переменной до тех пор, пока не будет нажата какая-либо другая клавиша, и тогда код сменится другим. Это означает, например для предыдущей программы, что стоит Вам только один раз коснуться клавиши "P", как рисование синей полосы будет идти безостановочно, пока не будет достигнут край экрана или пока не произойдет нажатие другой клавиши. Но такое управление тоже часто применяется в программах, и об этом

надо знать.

3. Если этот недостаток является существенным, то можно воспользоваться системной переменной KSTATE. Она занимает 8 байтов, начиная с адреса 23552 по 23559. Строго говоря, это не просто переменная, а это небольшая область памяти, в которой процедуры ПЗУ, отвечающие за опрос клавиатуры, организуют свои временные хранилища информации. Наиболее интересен для нас адрес клавиши, причем он хоть и "запоминается" там, но ненадолго, в отличие от переменной LASTK. Здесь Вы можете прочитать код последней нажатой клавиши, причем он хоть и "запоминается" там, но ненадолго. Он хранится пять циклов прерываний, а поскольку в "Спектруме" прерывания происходят каждую 1/50 секунды, то код хранится в этой ячейке примерно 0.1 секунды.

4. И, наконец, последнее замечание. Для того, чтобы системные процедуры ПЗУ могли производить опрос клавиатуры и устанавливать системные переменные LASTK и KSTATE, необходимо, чтобы маскируемое прерывание было разрешено. Надо убедиться в этом, а лучше принудительно перед опросом клавиатуры дать команду EI (код FB HEX).

4.5. Проверка оперативной памяти компьютера.

Приведенная ниже программа также является полезной и поучительной. Логика ее работы такая. Перед вызовом этой процедуры в регистровой паре HL должен быть установлен начальный адрес проверяемой области ОЗУ, а в паре BC - длина этой области. Проверка проводится в пять этапов. В каждую ячейку записывается проверочный байт, а затем проверяется, как он туда записался. Если вскрывается ошибка, то программа прерывается с включением флага CARRY, а содержимое регистра HL указывает на адрес ошибочной ячейки. Если проверка прошла нормально, то проверяемая область памяти после выхода будет содержать нули, а если нет, то в аккумуляторе будет содержаться то число, которое было записано в ячейку, но не воспроизвелось.

Мы ассемблировали эту программу с адреса 26000, но Вы, конечно, можете это переделать.

На первом проходе в память записываются нули и проверяются.

На втором проходе записывается FF. На третьем - AA (10101010B). На четвертом - 55 HEX (01010101B). Пятый проход - проверка с "плавающим" битом. Сначала помещается число 80 HEX (1000 000B), а затем включенный бит перемещается вправо, пока не пройдет полный оборот и происходит переход к проверке следующего адреса.

Адрес	Метка	Маш.код	Мнемоника		Комментарий
6590		7A	LD	A,D	Проверка заданной области ОЗУ на 0
6591		B3	OR	E	
6592		C8	RET	Z	Выход, если так
6593		42	LD	B,D	В BC - размер области
6594		4B	LD	C,E	
6595	TEST1	97	SUB	A	Это то же, что и LD A,0
6596		CDC065	CALL	FILL	Вызов процедуры заполнения области байтом
6599		D8	RET	C	Выход, если найдена ошибка (флаг C включен)
659A	TEST2	3EFF	LD	A,FF	
659C		CDC065	CALL	FILL	
659F		D8	RET	C	
65A0	TEST3	3EAA	LD	A,AA	
65A2		CDC065	CALL	FILL	
65A5		D8	RET	C	
65A6	TEST4	3E55	LD	A,55	
65A8		CDC065	CALL	FILL	
65AB		D8	RET	C	
65AC	TEST5	3E80	LD	A,80	В акк-р загружен байт 10000000
65AE	REP	77	LD	(HL),A	Этот байт загружен в проверяемый адрес
65AAF		BE	CP	(HL)	И тут же проверяется
65B0		37	SCF		Включается флаг C на случай выхода по ошибке
65B1		C0	RET	NZ	Выход, если ошибка
65B2		0F	RRCA		Сдвиг включенного бита вправо

Адрес	Метка	Маш.код	Мнемоника		Комментарий
65B3		FE80	CP	80	Сравнение нового байта с 1000 0000, т.е. сделан полный оборот или нет?
65B5		20F7	JR	NZ,REP	Если нет, то возврат и опять проверка
65B7		3600	LD	(HL),00	Если да, то сначала обнулить проверяемую ячейку
65B9		23	INC	HL	и перейти к проверке следующей, уменьшив счетчик
65BA		0B	DEC	BC	проверяемых ячеек ОЗУ
65BB		78	LD	A,B	Проверка на конец
65BC		B1	OR	C	проверяемой области
65BD		20ED	JR	NZ,TEST5	Переход к проверке следующей ячейки, если область не исчерпана
65BF		C9	RET		Выход, если исчерпана
65C0	FILL	E5	PUSH	HL	
65C1		C5	PUSH	BC	Запомнили на стеке HL и BC
65C2		5F	LD	E,A	Временно освободили A для других дел
65C3		77	LD	(HL),A	Засылка в проверяемую ячейку проверочного числа
65C4		0B	DEC	BC	Уменьшение счетчика проверяемых ячеек
65C5		78	LD	A,B	Проверка на то, что область
65C6		B1	OR	C	исчерпана
65C7		7B	LD	A,E	Восстановили аккумулятор
65C8		2805	JR	Z,COMP	Переход на проверку, если длина проверяемой области была всего 1 байт
65CA		54	LD	D,H	Подготовка к применению
65CB		5D	LD	E,L	команды для быстрой переброски байта LDIR
65CC		13	INC	DE	Адрес приемника на единицу больше источника

```
65CD        EDB0     LDIR           Переброска байта
65CF  COMP  C1       POP  BC       Восстановление со стека
65D0        E1       POP  HL       регистровых пар
65D1        E5       PUSH HL       без изменение содержания
65D2        C5       PUSH BC       стека
65D3  AGAIN EDA1     CPI            Сравнение
65D5        2007     JR   NZ,ERROR  Если сравнение не прошло,
                                    то переход на подпрограмму
                                    обработки ошибки
65D7        EAD365   JP   PE,AGAIN  Напоминаем, что команда CPI
                                    обнуляет флаг четности, ес-
                                    ли исчерпан список сравне-
                                    ния, т.е. если BC = 0
65DA        C1       POP  BC       Восстановление регистров BC
65DB        E1       POP  HL       и HL перед выходом
65DC        B7       OR   A        Выключение флага C,
                                    поскольку ошибок нет
65DD        C9       RET            Выход
65DE  ERROR C1       POP  BC       Восстановление регистров
65DF        D1       POP  DE       перед выходом
65E0        37       SCF            Включение флага "C",
                                    поскольку была ошибка
65E1        C9       RET            Выход
```

4.6. Практические приемы работы с калькулятором.

~~~~~~

В части 1 мы достаточно подробно осветили принципы работы встроенного калькулятора "Спектрума", а в "Справочнике" (часть 3) дали полную сводку команд калькулятора с указанием особенностей их применения. Система команд калькулятора достаточно проста, но вместе с тем при практическом программировании начинающий пользователь упирается в проблему. А как организовать взаимосвязь трех основных инструментов "Спектрума" - БЕЙСИКа, машинного кода и калькулятора?

Если связь между БЕЙСИКом и машинным кодом осуществляется достаточно просто через выделение области программных переменных, к которым БЕЙСИК обращается через PEEK, POKE, а машинный
~~~~~~

код через LD..., то с калькулятором могут появиться сложности.

Вы уже должны знать, что вся работа калькулятора строится вокруг специальной области памяти - стека калькулятора. Если у Вас есть инструмент для того, чтобы из БЕЙСИКа помещать переменные на стек калькулятора (не путать с машинным стеком процессора), то и проблем для Вас не будет. Логика работы такая:

- поместить данные из БЕЙСИКа на стек калькулятора;
- перейти в машинный код;
- перейти в калькулятор;
- обработать данные;
- выйти из калькулятора;
- войти в БЕЙСИК

Но данные на стеке калькулятора хранятся и обрабатываются в пятибайтной (интегральной) форме, в этом и состоит проблема. Можно, конечно, организовать стек калькулятора в нужной Вам области памяти, затем произвести вручную пересчет передаваемых Вами данных в интегральную форму, а потом через POKE заполнить стек нужным Вам числом, но это очень утомительно. Существуют хитрые приемы и здесь мы считаем своим долгом с Вами поделиться специальной техникой.

Рассмотрим простейшую БЕЙСИК-строку: LET P = Q + USR addr. Вы, конечно, понимаете, что по такой команде будет вызвано исполнение программы в машинных кодах, которая записана, начиная с адреса addr. А уж что будет дальше - это вопрос, который зависит от того, что написано в этой машинокодовой программе. Может быть после нее Вы в БЕЙСИК и не вернетесь, и до вычисления Q + ... дело и не дойдет. Зато вместе с запуском машинного кода вышеуказанная БЕЙСИК-строка делает еще одно дело. Она помещает содержимое переменной Q на вершину стека калькулятора!!! И, конечно, же делает это в пятибайтной форме. Если теперь Вы организуете машинный код с адреса addr так, чтобы он переходил в калькулятор и производил обработку данных, помещенных на стек, то проблема решена.

Рассмотрим конкретный пример. Допустим, Вам надо вычислить выражение P = SQR(Q+1). Допустим, мы расположим наш машинный код, начиная с адреса 30000.

Дайте команду LET P = Q + USR 30000. Q поместится на вершину стека калькулятора, и запустится машинный код, начинающий-

ся с адреса 30000.

АДРЕС	КОД	МНЕМОНИКА	СТЕК	КОММЕНТАРИЙ
30000	EF	RST 28	Q	Включение калькулятора, Q уже находится на стеке
30001	A1	stk_one	Q,1	Команда калькулятора, помещающая на стек единицу
30002	0F	add	Q+1	Команда калькулятора "сложение"
30003	28	sqr	SQR(Q+1)	Извлечение квадратного корня
30004	38	end calc		Выключение калькулятора
30005	C9	RET		Возврат в БЕЙСИК.

А как быть, если надо передать последовательность данных? А если надо передать символьный стринг?

В этом случае можно воспользоваться оператором БЕЙСИКа DEF FN <имя> (). Он служит для объявления функции пользователя, которая в дальнейшем может вызываться на исполнение командой FN <имя> (). В круглых скобках может стоять список параметров функции пользователя, отделенных друг от друга запятыми. Задание и вызов функции пользователя выглядят по-разному в зависимости от того, являются ли они числовыми или стринговыми. Например, они могут выглядеть так:

DEF FN A(K,L,M,C$,D$...) - числовая функция

DEF FN A$(K,L,M,C$,D$...) - символьная функция

Аналогично выглядят и обращения к ним:

FN A(K,L,...); FN A$(K,L,...)

Чтобы двигаться дальше, нам необходимо знать, как хранятся в БЕЙСИКе параметры функций пользователя. Для этого выделяется специальная область памяти, на которую указывает системная переменная DEFADD, имеющая длину два байта и адреса 23563, 23564. Найти адрес, с которого начинается размещение параметров функции пользователя, несложно:

PRINT PEEK 23563 + 256*PEEK 23564

Каждый параметр числовой функции пользователя занимает 8 байтов, а стринговой - девять байтов, и хранятся они там в следующем формате:

Для числовой функции:

Байт 1 - Имя функции (один символ);

Байт 2 - код 0E;

Байт 3...7 - значение параметра в пятибайтной форме;

Байт 8 - код 29, если этот параметр последний, если нет - код 2C;

Для символьной функции:

Байт 1 - Имя функции (один символ);

Байт 2 - код 24;

Байт 3 - код 0E;

Байт 4...8 - значение параметра в пятибайтной форме;

Байт 9 - код 29, если этот параметр последний, если нет - код 2C.

Теперь рассмотрим, как список параметров функции пользователя может быть использован для передачи на вершину стека калькулятора данных в пятибайтной форме. Для этого могут быть использованы конструкции:

DEF FN A(.....) = USR addr - для работы с числами;

DEF FN A$(.....) = P$ AND USR addr - для работы со стрингами.

И теперь Ваша задача организовать, начиная с адреса addr такую процедуру в машинных кодах, которая прочитает передаваемое в качестве параметра функции пользователя число в пятибайтной форме, поместит его на стек и перейдет к переброске следующего числа, если данный параметр не является последним. Например, такая процедура может выглядеть так:

```
2A0B5C       LD   HL,(DEFADD)   указание на адрес параметров
                                 функции пользователя DEF FN
23      REP  INC  HL            теперь HL указывает на 2-й
                                 байт, который 0E или 24
7E           LD   A,(HL)        проверка этого байта
FE0E         CP   0E
2801         JR   Z,NUMB        Вперед, если функция числовая
23           INC  HL            Если символьная, то лишний байт
                                 надо пропустить
23      NUMB INC  HL            Теперь HL указывает на
                                 пятибайтную форму
```

CDB433	CALL 33B4	Вызов процедуры ПЗУ STACK-NUM, которая отправляет пятибайтное число на стек калькулятора
7E	LD A,(HL)	
FE2C	CP 2C	Проверка на то, что этот параметр последний
28F0	JR Z,REP	Возврат на повтор, если не так
C9	RET	Выход в БЕЙСИК.

Можно передавать также и переменные из БЕЙСИКа на стек калькулятора. Первое, что надо сделать - это поместить имя переменной в качестве стринга на стек калькулятора, а затем использовать команду калькулятора "val" для числовой переменной или "val$" для символьной. Однако надо помнить, что обе эти команды являются зависимыми от регистра B процессора, т.е. еще находясь в машинном коде, до команды RST 28, надо в регистр B поместить 1D для команды "val" или 18 для команды "val$".

Чтобы поместить имя переменной на вершину стека в качестве стринга, можно поместить туда в упакованной форме код первого символа, затем конвертировать его в символ командой калькулятора chr$, затем тоже для второго символа, слить их воедино командой s_add и т.д. в зависимости от длины имени переменной. Рассмотрим два примера - для числовой переменной и для строковой.

Передача числовой переменной.

МАШ.КОД	МНЕМОНИКА	КОММЕНТАРИЙ
061D	LD B,1D	Предварительная настройка регистра B с целью последующего вызова команды калькулятора val
EF	RST 28	Включение калькулятора
3440B00041	stk_data 41	Поместить на стек код буквы A(41 HEX)
2F	chr$	Преобразование кода 41 в стринг "A"
3B	execute_b	Выполнение команды, код которой записан в регистре B, т.е. преобразование переменной A в ее числовое значение
38	end_calc	Выключение калькулятора

Передача строковой переменной.

Пусть имя переменной B$.

МАШ.КОД	МНЕМОНИКА	КОММЕНТАРИЙ
0618	LD B,18	Предварительная настройка регистра B с целью последующего вызова команды калькулятора val$
EF	RST 28	Включение калькулятора
3440B00042	stk_data 42	Поместить на стек код буквы B(42 HEX)
2F	chr$	Преобразование кода 42 в стринг "B"
3440B00024	stk_data 24	Помещение на стек кода знака $
2F	chr$	Преобразование кода 24 в стринг "$"
17	s_add	Слияние стрингов в стринг "B$"
3B	execute_B	Преобразование имени стринга "B$" в его значение.
38	end_calc	Выключение калькулятора

4.7 Примеры использования прерываний 2-го рода.

В первой части этой книги мы рассмотрели теоретические аспекты использования прерываний 2-го типа в пользовательских программах. Здесь мы рассмотрим несколько конкретных примеров.

Так как полное понимание концепции использования прерываний очень важно, ниже приведены примеры программ обработки прерываний и включения режима "IM 2", написанные на АССЕМБЛЕРе с комментариями. Желательно предварительно поработать с ними, прежде чем начать создавать свои программы.

Вообще, для работы с "IM 2" необходимо создание двух программ сразу : включающей прерывания "IM 2" и, собственно, программы обработки прерываний. После запуска первой программы, она может быть удалена из памяти.

Если Вы будете пользоваться каким-либо АССЕМБЛЕРОМ, допускающим двойную установку адреса трансляции "ORG", например, это разрешает программа "ZEUS" фирмы "CRYSTAL COMPUTING", то Вы можете набирать программы так, как это представлено в листинге программ, иначе Вам придется разбивать программы на две части и транслировать их по отдельности с нового адреса "ORG".

1. Эта программа демонстрирует смену типов прерываний с "IM 1" на "IM 2" с вызовом указателя из ПЗУ (см. ч.III, гл.5). После запуска программы в правом верхнем углу появится мигающий черно-белый атрибут, свидетельствующий о том, что прерывания переключены, и до тех пор, пока включен "IM 2", этот атрибут будет постоянно включен. При этом вы сможете работать в БЕЙСИКЕ или в других программах (если, конечно, они не перекрывают область памяти, занимаемую программой обработки прерываний "IM 2"), как и в обычном режиме.

```
 10          ORG 40000     ; адрес трансляции и запуска про-
 20                        ; граммы смены прерываний.
 30          DI
 40          LD A,33       ; выбираем адрес точки прерываний
 50          LD I,A        ; равным 33485 и загружаем в "I"
                           ; новый вектор.
 60          IM 2          ; смена "IM 1" на "IM 2".
 70          EI
 80          RET

 90          ORG 33485     ; программа обработки "IM 2".
100          DI
110          PUSH HL
120          LD HL,581FH   ; адрес экрана для атрибута.
130          LD (HL),C7H   ; загружаем по адресу код атрибута.
140          POP HL
150          JP 0038H      ; переход для сканирования
                           ; клавиатуры.
```

2. Это программные часы, работающие и выводящие время на экран по прерыванию. В этой программе имеется своя процедура для печати в экранную область. Вообще, если Вы хотите что-либо печатать на экране во время прерываний, то лучшим способом является введение своей подпрограммы печати, т.к. при использовании программ печати из ПЗУ Вы можете запутаться с сохранением открытого канала печати.

Программа часов использует данные в двоично-десятичной форме, именно в такой форме Вам необходимо перед запуском часов задать время, записав в ячейки :

63667 - часы,
63668 - минуты,
63669 - секунды.

Для задания времени из БЕЙСИКА можно пользоваться следующим способом. Например, Вы задаете время : 12 часов 24 минуты 30 секунд:

Тогда в 63667 запишем 1*16+2=18,
63668 - 2*16+4=36,
63669 - 3*16+0=48.

Т.е. десятки умножаем на 16 и прибавляем младший разряд.

```
 10  HOUR   EQU 63667    ; точка хранения часов.
 20  MIN    EQU 63668    ; -/-/-/-/- минут.
 30  SEC    EQU 63669    ; -/-/-/-/- секунд.
 40  COR    EQU 63666    ; -/-/-/-/- переменной коррекции.
 50         ORG 65040    ;адрес трансляции и запуска программы
 60                      ;смены прерываний.
 70         DI
 80         PUSH HL
 90         LD A,FDH     ; адрес указателя FDFFH - 65023.
100         LD I,A
110         LD HL,FDFFH  ; заполняем указатель адресом старта
120         LD (HL),F7H  ; программы обработки "IM 2" -
                         ; F7F7H - 63479
130         INC HL
140         LD (HL),F7H  ;
150         POP HL
160         IM 2
170         EI
180         RET
190         ORG 63479    ; адрес программы часов по "IM 2".
200         PUSH IX
210         PUSH AF
220         PUSH BC
230         PUSH DE
240         PUSH HL
250         DI
260         LD A,(COR)   ; проверка на необходимость изменения
270         DEC A        ; счетчика времени.
```

```
280         LD (COR),A
290         JP NZ,WRITE  ; если не ноль, то сразу на печать,
300         LD A,32H     ; иначе восстановим переменную коррекции и
310         LD (COR),A   ; и ведем подсчет времени.
320         LD A,(SEC)   ; подпрограмма подсчета секунд.
330         AND A
340         ADC A,01
350         DAA
360         LD (SEC),A
370         CP 60H       ; если не 60,
380         JP NZ,WRITE  ; то переход на печать,
390         XOR A        ; иначе обнуление секунд и подсчет минут.
400         LD (SEC),A
410         LD A,(MIN)   ; подпрограмма подсчета минут.
420         AND A
430         ADC A,01
440         DAA
450         LD (MIN),A
460         CP 60H
470         JP NZ,WRITE
480         XOR A
490         LD (MIN),A
500         LD A,(HOUR)  ; подпрограмма подсчета часов.
510         AND A
520         ADC A,01
530         DAA
540         LD (HOUR),A
550         CP 24H       ; проверка на 24-х часовой цикл.
560         JP NZ,WRITE
570         XOR A
580         LD (HOUR),A
590  WRITE  LD IX,4018H  ; программа вывода данных на экран,
600                      ; в IX содержится адрес начала печати.
610         LD A,(HOUR)  ; печатаем часы.
620         CALL PRFP
630         LD A,0AH     ; печатаем двоеточие.
640         CALL PRCH
650         LD A,(MIN)   ; печатаем минуты.
```

```
 660         CALL PRFP
 670         LD A,0AH
 680         CALL PRCH
 690         LD A,(SEC)    ; печатаем секунды.
 700         CALL PRFP
 710         LD HL,5818H   ; заполняем место печати времени
 720         LD B,08       ; мигающими черно-белыми атрибутами.
 730  FLAG   LD (HL),C7H
 740         INC HL
 750         DJNZ FLAG
 760         POP HL
 770         POP DE
 780         POP BC
 790         POP AF
 800         POP IX
 810         JP 0038H      ; переход для сканирования клавиатуры.
 820  PRFP   PUSH AF       ; подпрограмма печати чисел.
 830         SRL A         ; вычисляем код старшей цифры.
 840         SRL A
 850         SRL A
 860         SRL A
 870         CALL PRCH     ; печатаем старшую цифру.
 880         POP AF
 890         AND 0FH       ; вычисляем код младшей цифры.
 900         CALL PRCH     ; печатаем младшую цифру.
 910         RET
 920  PRCH   PUSH IX       ; подпрограмма печати литеры.
 930         LD HL,(5C36H); загружаем адрес знакогенератора.
 940         LD DE,0180H   ; производим необходимое смещение.
 950         ADD HL,DE
 960         EX DE,HL
 970         LD L,A        ; вычисляем по коду литеры ее адрес
 980         LD H,00       ; в таблице знакогенератора.
 990         ADD HL,HL
1000         ADD HL,HL
1010         ADD HL,HL
1020         ADD HL,DE
1030         LD DE,0100H
```

```
1040         LD B,08
1050  PSET   LD A,(HL)    ; выводим на экран литеру.
1060         XOR FFH
1070         LD (IX+00),A
1080         INC HL
1090         ADD IX,DE
1100         DJNZ PSET
1110         POP IX
1120         INC IX
1130         RET
```

Вы, вероятно заметили, что в подпрограмме "PRCH" есть смещение в знакогенераторе до литеры "0", т.е. все коды литер смещены на 48 байтов по сравнению с таблицей ASCII. Возможно, это неправильно, но упрощает подпрограммы печати чисел и литер.

Предлагаемые программы можно переместить, если рассчитать новые значения указателей и изменить содержимое регистра "I".

5. КАНАЛЫ И ПОТОКИ

Для многих начинающих пользователей "Спектрума" такие понятия, как каналы и потоки могут зву чать, как непонятные жаргонные обозначения, но на самом деле за ними скрывается интересная концепция, которая позволит Вам взять от компьютера то, что другими способами взять не так просто.

Работая в БЕЙСИКе, Вы можете и не задумываться о потоках и каналах, а вот программируя в машинных кодах, без них не обойтись.

Можете представить себе, что канал - это некоторое техническое устройство, используемое для ввода/вывода информации. Надо, правда,оговориться, что канал - не всегда техническое устройство. Каналом, например, может быть файл на диске или, скажем, в памяти Вашего компьютера. В файл ведь тоже можно заносить информацию и можно ее оттуда считывать.

Проще всего представить концепцию каналов и потоков на примере морского побережья с многочисленными заливами и бухтами. Со стороны суши в них впадают многочисленные ручьи и реки. Так вот, эти заливы и бухты - это каналы, а те ручьи и реки,

которые в них впадают - это потоки, подключенные к каналам.

5.1 Стандартные каналы.

Стандартными каналами "Спектрума" для вывода информации являются каналы "K" - нижние две строки экрана (системное окно), "S" - главная часть экрана и "P" - стандартный "ZX-принтер".

К этим каналам стандартно подключены потоки:

- поток #0 - к каналу "K";
- поток #1 - тоже подключен к каналу "K";
- поток #2 - подключен к каналу "S";
- поток #3 - к каналу "P".

Таким образом, оказываются идентичными следующие команды ввода/вывода:

PRINT #0 "Hello"; A$ - то же самое, что и INPUT "Hello"; A$

PRINT "Hello" - то же самое, что и PRINT #2 "Hello"

LPRINT "Hello" - то же самое, что и PRINT #3 "Hello"

Номер, стоящий после знака # в вышеприведенных примерах, является номером потока. Поскольку эти потоки подключены стандартно и переподключены быть не могут, мы программируем на БЕЙСИКе и используем операторы INPUT, PRINT, LPRINT без указания номера потока.

В "Спектруме" каждый канал имеет имя, которое выражено одной буквой алфавита. Так, канал "S" - это экран, потому, что по-английски слово экран звучит, как "Screen".

Оператор OPEN# служит для того, чтобы подключить поток к каналу. Так, Вы можете в БЕЙСИКе дать команду OPEN#6,"S" и тем самым подключите шестой поток к экрану и тогда команда PRINT #6 будет печатать текст на экране точно так же, как это делает обычная команда PRINT.

Аналогично, Вы можете представить, что клавиатура - это устройство, предназначенное для ввода информации и потому это тоже канал. Он имеет имя "K" (Keyboard - клавиатура). К нему можно подключить поток точно так же, как мы это делали с экраном. Например, OPEN #n,"K" - подключает к каналу клавиатуры поток номер n.

Всего Вы можете иметь не более шестнадцати потоков с

номерами от 0 до 15. Потока с номером 16 не существует и, если Вы попробуете его использовать, то получите сообщение об ошибке.

Мы уже сказали о том, что потоки от 0 до 3 являются стандартными и организованы без нашего участия. Они стандартно подключены к стандартным каналам.

Каналы "S" и "P" предназначены только для вывода информации, поэтому например PRINT #2 или PRINT #3 - возможны, а INPUT #2 или INPUT #3 - невозможны.

В отличие от них, канал "K" может использоваться и для ввода и для вывода.

INPUT #0 - самая обычная команда INPUT, а PRINT #0 - выполняет печать Вашего сообщения в нижние две строки экрана, которые выполняют роль "системного окна". Это те самые строки, в которых появляется информация при работе команды INPUT.

5.2 Прочие каналы.

После подключения дополнительной периферии, располагающей своим ПЗУ, к стандартным каналам могут добавляться дополнительные. Так, например, подключение ИНТЕРФЕЙСа-1 (ZX-INTERFACE 1) создает для пользователя несколько новых каналов:

- "M" - канал микродрайва;
- "N" - канал локальной сети;
- "T" - канал принтера для печати программ (коды выше 165 интерпретируются, как токены ключевых слов "Спектрума").
- "B" - канал принтера для печати данных (все коды интерпретируются по своему значению).

Дисковые интерфейсы могут добавлять дополнительно свои каналы, например, канал "D" и т.п. Но у Вас есть возможность и для создания собственных каналов. Вы можете создавать их в оперативной памяти и эффективно использовать. Такие каналы мы будем называть пользовательскими, но для того, чтобы научиться их создавать, нам надо ознакомиться с ОБЛАСТЬЮ ИНФОРМАЦИИ О КАНАЛАХ.

5.3 Область информации о каналах.

Вся концепция каналов и потоков базируется на существовании в оперативной памяти компьютера области, называемой "Областью информации о каналах". Эта область лежит непосредственно перед БЕЙСИК-областью, чуть ниже ее. То есть, она лежит между системными переменными и БЕЙСИКом. Начинается она с адреса, на который указывает системная переменная CHANS, расположенная по адресу 5C4FH (23631) и заканчивается байтом, в котором стоит маркер 80H. Далее уже идет Ваша БЕЙСИК-программа, на начало которой, как известно, указывает системная переменная PROG, расположенная по адресу 5C53H (23635).

Для того, чтобы создать свой канал, Вам практически надо переорганизовать данные в этой области, может быть и раздвинуть эту область, переместив вверх маркер и поменяв значение PROG, а также разместить в памяти две процедуры. Одну - для обеспечения ввода в Ваш канал (INPUT) и вторую - для обеспечения вывода - (PRINT).

Каждый канал должен иметь блок информации о канале - CHANNEL INFORMATION BLOCK (CIB). Он и располагается в области информации о каналах. В этом блоке содержится вся информация, необходимая для того, чтобы канал мог функционировать. Стандартные каналы "K", "S", "P" имеют каждый по пятибайтному блоку, но это исключение. Все остальные каналы, в том числе и те, которые создадите Вы, должны иметь в этом блоке не менее, чем по 11 байтов.

Блок CIB для стандартного канала.

Байты 0 и 1 - адрес процедуры вывода (PRINT #);
Байты 2 и 3 - адрес процедуры ввода (INPUT #);
Байт 4 - имя канала (код одной буквы - "K", "S" и т.п.).

Блок CIB для канала, создаваемого стандартной периферией.

Если при подключении стандартных периферийных устройств создаются новые каналы, то блок информации имеет длину в 11 байтов и более. Обычно адресуются к данным, имеющимся в этом

блоке путем индексной адресации, поместив в регистр IX базовый адрес начала блока. Так, например, при подключении ИНТЕРФЕЙСа-1 блоки дополнительных каналов имеют следующий формат:

IX+0 (2 байта) - адрес процедуры обработки ошибок 0008.
IX+2 (2 байта) - адрес процедуры обработки ошибок 0008.
IX+4 (1 байт) - имя канала.
IX+5 (2 байта) - адрес процедуры PRINT#.
IX+7 (2 байта) - адрес процедуры INPUT#.
IX+9 (2 байта) - длина данного блока (не менее 000BH).
IX+0B (длина любая) - любая дополнительная информация.

Блок CIB для канала, создаваемого пользователем.
~~~~~~

IX+0 (2 байта) - адрес процедуры вывода (PRINT#).
IX+2 (2 байта) - адрес процедуры ввода (INPUT#).
IX+4 (1 байт) - имя канала.
IX+5 (2 байта) - число "1234" - оно свидетельствует о том, что канал - пользовательский.
IX+7 (2 байта) - адрес закрывающей процедуры CLOSE#.
IX+9 (2 байта) - длина данного блока (не менее 000BH).
IX+0B (длина любая) - любая дополнительная информация.

5.4 Подключение потоков.

Теперь, когда мы с Вами разобрались с каналами и поняли, что это не так сложно, что это всего лишь еще один способ организации памяти компьютера и взаимодействия процедур друг с другом, вернемся к потокам.

Среди системных переменных компьютера, есть переменная STRMS. Ее адрес - 5C10H (23568). Ее назначение - указание на адреса каналов, подключенных к потокам. Длина этой переменной - 38 байтов и, если говорить откровенно, то никакая это не переменная, а самая настоящая указательная таблица, в которой каждому потоку отдано по 2 байта для того, чтобы хранить в них адрес канала, к которому подключен данный поток.

Внимательный читатель, конечно заметил, что 16 потоков по 2 байта на поток составляет 32 байта, а системная переменная
~~~~~~

STRMS имеет почему-то 38 байтов. Дело в том, что есть еще три системных потока, занимающиеся своими "внутренними" делами при работе ПЗУ. Это "минус третий" поток (FD), "минус второй" (FE) и "минус первый" (FF). Таким образом, таблицу STRMS можно представить, как 19 двухбайтных системных переменных:

```
STRMS_FD   5C10 (23568)
STRMS_FE   5C12 (23570)
.....................
STRMS_0F   5C36 (23606)
```

Поток FD подключен к каналу "K" и не должен переподключаться. Аналогично поток FE подключен к каналу "S". Интересен поток FF, который подключен к "внутреннему" каналу "R" (мы о нем не упоминали, поскольку маловероятно, чтобы им пришлось кому-либо пользоваться), который занимается организацией динамического копирования информации из одних областей памяти в другие, производя при этом "раздвигание" информации для вставки новой в середину имеющейся.

Итак, с помощью таблицы переменных STRMS выполняется привязка потоков к каналам. Если какая-либо переменная из набора STRMS содержит 0000, то это означает, что к данному потоку не подключен ни один канал, иначе коворя, канал закрыт. Если же там не ноль, значит канал открыт и этот поток подключен к этому каналу. Фактически же поток подключен к тому каналу, информационный блок CIB которого находится по адресу, на который указывает системная переменная CHANS плюс величина, содержащаяся в STRMS для данного потока минус единица:

```
(CHANS) + (STRMS_n) -1
```

5.5.Практические приемы работы с каналами и потоками.

Итак, мы знаем, что информация о каналах хранится в ОЗУ, в ОБЛАСТИ ИНФОРМАЦИИ О КАНАЛАХ в виде блоков информации о канале. Конечно, при включении компьютера там ничего пока нет, а информация эта содержится только в ПЗУ и при инициализации системы копируется в ОЗУ. Такое двойное хранение информации сделано в качестве шага навстречу пользователю, ведь в ОЗУ он может менять информацию так, как ему необходимо. Рассмотрим, например, как выглядит в ОЗУ блок информации по каналу "S" (он

занимает, как мы уже установили, 5 байтов).

Адрес	Содержимое	Комментарий
23739	244	9*256+244 = 2548 - адрес процедуры
23740	9	обслуживания канала при выводе.
23741	196	21*256+196 = 5572 - адрес процедуры
23742	21	обслуживания канала при вводе.
23743	83	Код литеры "S"

Во многих фирменных программах для того, чтобы не искажать изображение на экране, полученное после загрузки фирменной экранной заставки, меняют адрес процедуры обслуживания канала "S" при выводе. Это делают подачей команд:

```
POKE 23739,82
POKE 23740,0
```

При этом загрузка блока в компьютер будет выполняться нормально, но на экран надписи program...., bytes.... и т.п. выводиться не будут.

Так происходит потому, что в ПЗУ по адресу 82 (0052H) записана команда RET (код C9H) и обращение туда в момент вывода информации на экран приводит просто к возврату и все. В ПЗУ имеются сотни адресов, в которых записана команда RET и использовать здесь можно было бы любую.

Использование же для этих целей команды CLOSE#2 - неприемлимо.

После загрузки всех блоков программы адрес процедуры вывода восстанавливается.

```
POKE 23739,244
POKE 23740,0
```

Этот нехитрый прием пользователи могут применять и в своих целях.

Другое, не менее интересное применение концепции каналов и потоков для практических целей - для защиты своей программы от несанкционированного вмешательства. Это делается изменением информации о канале "R", который используется системой компьютера при работе внутреннего редактора. Достаточно записать:

```
POKE 23744,124
POKE 23745,0
```

- и редактирование Вашей программы станет невозможным.

Аналогичные эффекты можно получить, манипулируя не с данными в области информации о каналах, а с данными в системной переменной STRMS (23568). Так как каналы связаны с определенными потоками, то переподключая потоки к другим каналам, получим новые эффекты. Например, чтобы запретить вывод на экран информации о программе при загрузке ее с магнитофона, достаточно записать в ячейку 23570 через команду POKE вместо числа 6 число 16, переподключив поток-2, связанный с каналом "S" на канал "P", обеспечивающий вывод на "ZX-принтер".

Переподключать стандартные каналы к потокам можно и из БЕЙСИКа с помощью символа "хэш" - "#". Попробуйте ввести программу и запустить ее:

```
10 PRINT #0; "Текст в нижней части экрана": PAUSE 0
20 LPRINT #2; "Текст в верхней части экрана": PAUSE 0
30 LPRINT #3: "Текст на принтере": PAUSE 0
40 LIST #1
```

Вы убедитесь, что знакомые Вам операторы БЕЙСИКа работают несколько необычно.

Кроме системных переменных CHANS (23631/2) и STRMS (23568), хранящих информацию о каналах и потоках, существует еще системная переменная CURCHL (5C51H - 23633), в которой записан адрес активизированного в данный момент канала. Именно значение этой переменной и используется командой вывода на печать RST 10H.

При работе со стандартными каналами компьютера, Вам может потребоваться еще и информация о некоторых флаговых системных переменных: FLAGS (5C3BH = 23611), FLAGS2 (5C6AH=23658) и TVFLAG(5C3CH=23612).

Нулевой бит переменной TVFLAG информирует систему о том, используется ли верхняя часть экрана (бит равен 0) или нижняя (бит равен 1).

Первый бит переменной FLAGS указывает на то, должен ли символ выводиться на принтер (бит равен 1) или на экран (бит равен нулю).

Четвертый бит системной переменной FLAGS2 определяет, является ли канал "K" рабочим (бит включен) или нерабочим (бит выключен) в текущий момент.

Выше мы говорили о том, что информация о каналах в компьютера записана дважды - в ПЗУ и затем скопирована в ОЗУ. Этим можно воспользоваться, например для того, чтобы вообще убрать из ОЗУ информацию о каналах и оставить ее только в ПЗУ. А за счет освободившегося при этом места можно увеличить БЕЙСИК-область компьютера на 21 байт. Это, конечно, немного, но зато интересен сам подход:

```
10 POKE 23635, PEEK 23631
20 POKE 23636, PEEK 23632
30 POKE 23631, 175
40 POKE 23632, 21
```

Здесь в строках 10 и 20 меняется значение системной переменной PROG, указывающей на начало БЕЙСИК-программы. Теперь она будет начинаться не с адреса 23755, а с адреса 23734. В строках

же 30 и 40 в системную переменную CHANS записывается начальный адрес области информации о каналах, находящейся в ПЗУ.

Весьма полезной, особенно для создателей обучающих программ, может быть следующая процедура, которая несколько меняет действие команды PRINT. При использовании этой процедуры по команде PRINT информация на экран будет выводиться с временной паузой между отдельными символами и с генерацией короткого звукового сигнала после вывода каждого символа. Причем величиной паузы можно регулировать какой-то обучающий момент программы, например в программах, развивающих чтение или скорочтение. Эффект имитации телетайпа, кроме того, вносит разнообразие в работу программы (вспомните, например, игровую программу Black Hawk).

```
10    BEEP     EQU 949
20    CURCHL   EQU 63633
30    TIME     EQU 65300
40             ORG 65301
50    PRINT    DEFW 2548
60    START    PUSH AF
70             LD A,I
80             JR PO,NOPAUSE
```

```
 90             LD A,(TIME)
100             LD B,A
110    PAUSE    HALT
120             DJNZ,PAUSE
130    NOPAUSE  POP AF
140             CP 33
150             JR C,NOBEEP
160             PUSH AF
170             PUSH IX
180             LD DE,50
190             LD HL,100
200             CALL BEEP
210             POP IX
220             POP AF
230    NOBEEP   LD HL,(PRINT)
240             CALL 111,
250             LD HL,(CURCHL)
260             LD BC,START
270             LD E,(HL)
280             LD (HL),C
290             INC HL
300             LD D,(HL)
310             LD (HL),B
320             LD A,B
330             CP D
340             JR NZ,CHNG
350             LD A,C
360             CP E
370             RET Z
380             LD (PRINT),DE
390             RET
```

Запуск этой процедуры может показаться несколько необычным. Для ее вызова не надо давать ни традиционную команду RANDOMIZE USR, ни похожие на нее PRINT USR, RESTORE USR и т.п. Для того, чтобы эта процедура выполнялась, достаточно записать в области информации о каналах в ячейках, отведенных для хранения адреса вызова процедуры, обслуживающей канал "S",

адрес начала этой процедуры. Стандартно там записан адрес 2548, а мы с помощью POKE запишем туда адрес 65303.

```
POKE 23739, 23
POKE 23740,255
```

Теперь по команде процессора RST 10H (вывод символа на экран) будет вызываться не процедура ПЗУ, а созданная нами процедура из адреса 65303. Временная пауза между выводимыми символами задается путем записи с помощью команды POKE соответствующего числа от 1 до 10 в ячейку 65300.

Пояснения к процедуре:

В строках 50...80 выполняется проверка на разрешение прерываний. Если прерывания запрещены, то выполняется переход на метку NOPAUSE, иначе программа может зависнуть на команде HALT в строке 110.

В строках 90...120 организуется цикл, длина которого берется из ячейки 65300 и устанавливается в регистре A, а затем B. В этом цикле находится команда HALT, которая и обеспечивает необходимую задержку по времени при печати символов.

В строках 130...150 проверяется код вводимого символа. Если его код менее 33, то это не печатаемый символ, а управляющий код или пробел и при его выдаче не надо давать звуковой сигнал. Поэтому выполняется переход к NOBEEP, обходя выдачу звукового сигнала.

Поскольку обработка управляющих кодов происходит с использованием других процедур, то прямой вызов CALL 2548 может привести к ошибке или потере управляющего кода. Поэтому выполняется CALL 111. По адресу 111 в ПЗУ записана команда JP(HL). Таким образом мы реализуем несуществующую в АССЕМБЛЕРе процессора Z-80 команду CALL (HL) и выполняем переход по адресу, записанному в регистровой паре HL (строка 230).

Внутри процедур обработки управляющих кодов имеются команды, которые меняют содержимое байтов области информации о каналах. Поэтому в строках 250...310 выполняется проверка записанного для канала "S" адреса процедуры вывода и восстановление адреса 65303, а затем восстановление, при необходимости, переменной PRINT (строки 320...390).

Данная процедура прекрасно работает только до момента, пока не будет выполнена команда CLS. Эта команда восстанавливает

в области информации о каналах стандартные адреса процедур вывода. Чтобы вернуть прежнее действие команде PRINT, необходимо опять же с помощью команды POKE записать в ячейки 23739/40 адрес 65303. И так поступать после каждой команды CLS или нажатия клавиши ENTER.

Естественно, это не совсем удобно, поэтому лучшим решением будет открытие нового пользовательского канала, адреса процедур ввода-вывода которого будут оставаться незыблемыми все время, пока включен компьютер. Для этого необходимо расширить область информации о каналах на 5 байтов, в которых записать адреса соответствующих процедур ввода и вывода и имя нового канала. Выполняется это с помощью процедуры ПЗУ MAKE_ROOM, находящейся по адресу 5717 (1655H). При этом перед вызовом этой процедуры в регистровую пару HL необходимо записать начальный адрес резервируемой области, а в регистровую пару BC - количество байтов. При использовании этой процедуры автоматически изменяются системные переменные, определяющие адреса отдельных блоков БЕЙСИК-области. Ниже представлена программа, позволяющая создать нужный нам канал:

```
10 FOR n=23296 TO 23304: READ a: POKE n,a: NEXT n
20 DATA 1,5,0,33,202,92,195,85,22
30 RANDOMIZE USR 23296
40 RESTORE 60
50 FOR n=23754 TO 23758: READ b: POKE n,b: NEXT n
60 DATA 23,255,196,21,83
70 STOP
```

После выполнения этой программы Вам остается только подключить новый канал к потоку_2 командой POKE 23578,21. Теперь можно не заботиться о необходимости восстановления адреса созданной процедуры в области данных о каналах.

6. ПРАКТИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПРОСМОТРУ МАШИННОГО КОДА ФИРМЕННЫХ ИГРОВЫХ ПРОГРАММ.

Приведенные в предыдущей главе примеры, конечно могут помочь в приобретении практических навыков программирования в ма-

шинных кодах, но в рамках одной книги, сколь бы объемной она ни была, невозможно охватить все приемы, используемые лучшими программистами мира. К тому же, техника программирования не стоит на месте и непрерывно развивается. Огромный материал для анализа Вам могут предоставить готовые игровые программы, и мы готовы в этой главе открыть некоторые приемы, которые позволят Вам сохранить массу времени при работе.

Сразу оговоримся, что рекомендации, которые мы здесь приведем, не являются строго однозначными и окончательными. Это только путеводитель. Возникли они как результат наших поисков и экспериментов, может быть Вы найдете другие, лучшие. Во всяком случае, с набором опыта Вы будете расширять и набор приемов. Может быть, Вы сами сможете поделиться своими приемами с нашими читателями.

Предположим, что Вы взяли игровую программу (незащищенную от просмотра) и с помощью, допустим, программы-копировщика выделили из нее машинокодовый блок. У разных программ он бывает разным, но допустим его размер примерно 40К.

Что будет, если Вы захотите только просмотреть его, например с помощью программы ДИСАССЕМБЛЕРа, скажем MONITOR или MONS. Если каждому байту Вы уделите хотя бы по 3 секунды своего внимания, то на просмотр всего блока у Вас уйдет более 30 часов напряженнейшей работы. В то же время, есть приемы, которые позволяют в течение 30...40 минут вчерне разобраться с устройством программы и дальнейшую работу производить более целенаправленно.

Самое лучшее, что можно сделать - это распечатать код программы на принтере. Уже за то время, пока он будет печататься, Вы сможете во многом разобраться. На худой конец напишите программу, которая будет распечатывать текст Вашего блока на экране. Главное требование, чтобы печать производилась в следующем формате строки: адрес, а за ним восемь байтов. Очень важно, чтобы их было именно восемь, это поможет при беглом просмотре кода многое открыть.

Давайте рассмотрим варианты того, что Вы при этом можете увидеть.

1. Если в тексте относительно часто встречаются коды CD (CALL NN) и FE (CP N), то есть большая вероятность, что перед

Вами главный логический блок программы, осуществляющий увязку всех структур в единое целое. Он обычно невелик (до 1К) и может быть запрятан глубоко в тело программы.

Пример фрагмента программы "BOMB JACK".

```
49952   F6  E6  CD  38  EC  CD  10  ED
49960   CD  A0  ED  CD  EB  D7  CD  5B
49968   CC  CD  8F  D9  CD  A8  D6  3A
```

2. Если перед Вами сравнительно однородные значения кодов, но все же относительно чаще появляются коды 20, 21, 22, 2A, 32, 3A, 3E, C9, то здесь скорее всего исполняемый машинный код, а именно - рабочие процедуры программы.

Пример фрагмента:

```
55656   C8  C9  11  00  40  06  BF  ED
55664   53  E1  FD  21  E3  FD  14  7A
55672   E6  07  20  12  7B  C6  20  5F
```

3. Если в выводимых строках явно преобладают близкие по значению коды, лежащие в диапазоне от 30 до 7A, то с большой вероятностью можно сказать, что в этих областях памяти хранятся тексты сообщений, запросов к пользователю, экранных меню и т.п.

Пример фрагмента.

```
61616   6E  20  77  69  74  68  20  74
61624   68  65  20  42  65  73  74  10
61632   03  10  42  6F  6D  62  65  72
```

4. Если в какой-то области памяти Вам встречаются относительно частые повторения нулей, то логично предположить, что эта область занята графикой, но это предположение следует серьезно проверять. Дело в том, что как правило, графические изображения хранятся в программах в компрессированном виде, т.е. многократные повторения одинаковых байтов из них удалены. Вернее другое предположение, что эта область предназначается программой для хранения программных переменных и, поскольку программа еще не начинала свою работу, то эта зарезервированная память заполнена пока нулями.

Возможен, конечно, вариант, что программист при стыковке всех блоков программы в единое целое просто оставил эту область в качестве "мусора".

Пример фрагмента:

```
63968   27  00  00  00  00  00  0A  00
63976   0A  00  10  27  00  00  00  00
63984   00  0A  00  0A  00  10  27  00
```

5. Участки, занятые графикой (экранами), отыскать несложно. Как правило, они занимают очень обширные области памяти (многие килобайты). Поскольку алгоритм, которым компрессируются и декомпрессируются изображения, Вам неизвестен, то здесь возможны варианты.

5.1. Если экран компрессирован по принципу замены группы повторяющихся байтов на их величину и количество повторений, а как правило, так оно и есть, то эти области сразу видны по "контрасту" следующих друг за другом байтов, причем еще раз отметим: эти области должны быть обширны.

Фрагмент:

```
34888   F8  02  F8  03  00  FF  01  FC
34896   03  78  00  EF  FC  01  FE  00
34904   10  C6  87  30  CF  00  39  00
34912   FF  00  FF  00  FF  00  FF  00
```

5.2. Бывает, что экраны хранятся в виде табулированных стрингов, т.е. в строго заданном формате (скажем в стринге длиной 256 байтов каждый бит несет информацию о том, какой символ, где и как должен помещаться на экран). Это бывает в программах с многочисленными экранами и псевдографикой. Например, MANIC MINER, JET SET WILLY. В этом случае Вам поможет тот факт, что распечатку Вы ведете по восемь кодов в строке. Возникает ситуация, когда при просмотре строк по вертикали Вы можете уловить закономерности повторения одинаковых байтов.

По этому же принципу можно уловить и области, где в программах размещаются графические изображения спрайтов (движущихся объектов, персонажей и т.п.)

Фрагмент программы:

```
49424   78  87  38  78  38  38  78  38
49432   30  08  78  81  38  04  78  81
49440   38  05  78  81  38  04  78  81
49448   38  08  78  81  38  04  78  81
```

6. Распечатка по восемь кодов в одну строку может Вам помочь определить еще одну важную область в программе - где про-

грамма хранит шаблоны для нестандартного генератора шрифта. Часто перед Вами стоит задача переделать шрифт в программе на русский. Если программа использует стандартный шрифт "Спектрума", то Вам достаточно где-то загрузить русский шрифт и поместить указание на него в системные переменные (мы об этом писали в нашей книге "Большие возможности Вашего Спектрума"). Если же программа применяет свой нестандартный генератор шрифта, то это ничего не даст, т.к. она все равно выставит системную переменную CHARS так, чтобы она указывала на программный генератор шрифта. Поэтому Ваша задача - отыскать, где в программе находятся шаблоны символов шрифта и переделать их.

При распечатке по восемь кодов в строке, Вы явно увидите в одной из позиций столбец нулей, когда будут распечатываться шаблоны. Пример фрагмента программы:

```
.................00.....................
36984   20  00  00  3C  6E  CE  D6  E6
36992   EC  78  00  3C  7E  18  18  18
37000   3C  6E  00  3C  7E  06  0C  10
37008   3E  7C  00  3E  7C  08  1C  06
.................00.....................
```

Вы, наверное, догадались, почему так происходит. Дело в том, что шаблон символа записывается восемью байтами, по одному байту на каждую строку шаблона сверху вниз. Поскольку верхняя строка почти всегда пустая (чтобы обеспечить минимальный интервал между строчками при печати), то каждый восьмой байт оказывается нулевым.

Дополнительным указанием на то, что здесь хранятся шаблоны символов, служит тот факт, что все байты в приведенном фрагменте - четные. И это тоже не случайно. Чтобы между буквами при печати был минимальный зазор, надо, чтобы правый крайний столбец шаблона был пустым. А правой позиции каждой строки шаблона соответствует нулевой бит. Если в байте нулевой бит выключен, то число четное, а если включен, то нечетное.

Итак, пользуясь нашими рекомендациями, Вы уже примерно через полчаса сможете грубо проанализировать структуру Вашей программы. Конечно, программы друг на друга не похожи. Конечно,

эти рекомендации являются относительно условными, но если Вы будете вести систематический целенаправленный поиск, а не тыкаться куда попало, то уже через два-три дня работы сможете выработать для себя свои критерии, свои методы и свои приемы. Наша же задача - дать первичную информацию и сэкономить Ваше время.

Обратите внимание на то, что все, о чем мы здесь говорили, относится к незащищенным блокам машинокодовых программ. Если блок защищен от просмотра, то сначала внимательно читайте загрузчик, там на каждом шагу могут быть подвохи, обманы, ложные коды.

Например, если в БЕЙСИКе Вы встретите запись:

LET a = 15: GO TO a

- это совсем не значит, что здесь выполняется переход к 15-й строке. Вас, может быть, обманывают. Дело в том, что формат БЕЙСИК-строки таков, что сначала число записывается в виде стринга, а затем в пятибайтной форме. Если Вы просмотрите БЕЙСИК-строку байт за байтом командой PEEK, то увидите, что каждое число, кроме номеров строк, повторено дважды. Так вот, первая запись говорит о том, что будет показано на экране, а вторая (интегральная) - что реально будет исполнено. Конечно, нормально эти записи соответствуют друг другу, но программист легко мог сделать, что на экране изображается 15, а в расчет идет что угодно другое. Не уловив такого ложного перехода в БЕЙСИК-строке, Вы потеряете нить и попадете в подготовленную ловушку.

Итак, все надо проверять, в том числе и БЕЙСИК-строки, желательно с помощью ДИСАССЕМБЛЕРа.

Все вышеприведенные фрагменты мы взяли из широкораспространенной программы BOMB JACK. Мы даже указывали адреса, где что содержится. Но не спешите заглядывать в машинный код этой программы. Вы ничего подобного там не увидите. Тот код, который содержится на ленте, не соответствует тому, который работает. Те адреса, куда загружается эта программа, не соответствуют тем, в которых она находится во время работы. Нестандартный загрузчик (он же, кстати, выдает на бордюр красивые цветные полосы при загрузке) выполняет раскодирование информации, поступающей от магнитофона и переброску всей программы в новое место

после окончания загрузки. Защита не самая серьезная, но достаточная, чтобы испортить настроение начинающему.

Приведенные выше фрагменты взяты нами после того, как все ступени предохранения были сняты.

Разобравшись грубо со структурой программы, Вы можете начать ее анализировать. Отыщите главный логический блок. В нем Вы найдете массу обращений к подпрограммам:

```
.........
CALL  NN
CALL  NN
CALL  NN
.........
```

Применяя дисассемблирующую программу, поставьте вместо CALL NN точку прерывания (BREAK POINT) и стартуйте проверяемую программу прямо из дисассемблера (JP ADDR). Программа начнет работать, дойдет до точки прерывания, остановится и вызовет дисассемблер. Снимите эту точку прерывания и поставьте другую в следующем CALL. Опять стартуйте проверяемую программу. Теперь она пройдет немного дальше, а Вы поймете, что делала та процедура, которую Вы перед этим отключали. Продолжайте в том же духе. Уже через два-три часа Вы будете знать назначение основных процедурных блоков исследуемой программы и сможете приступить к их детальному анализу.

Как правило, Вы начинаете разбор готовой програмы с какой-то конкретной целью. В зависимости от того, какова Ваша цель, Вы можете применять какие-либо иные тактические приемы.

Если Вы хотите модернизировать программу потому, что количество попыток на ее прохождение Вам недостаточно, то Вы можете применять такую стратегию.

Допустим, Вам дано всего четыре попытки. Очевидно, где-то в программе есть переменная, и ей отведен адрес, в котором хранится количество оставшихся попыток. Найти этот адрес - задача непростая, но ведь как-то в начале работы там было установлено число 4. С большой вероятностью можно полагать, что оно устанавливалось сначала в аккумуляторе, а потом пересылалось в адрес. Если это так, то где-то в программе должна быть команда LD A,4 (машинный код 3E04). Напишите несложную программу на БЕЙСИКе, которая просмотрит всю память исследуемой программы и вы-

даст Вам адреса, в которых она встретила сочетание кодов 3E и 04. Поверьте, их будет очень немного. Теперь с помощью POKE дайте в найденный адрес вместо 04 скажем 05 или 25. Посмотрите, что получилось. Если Вы работаете с дисассемблером, то он уже имеет команды на поиск заданного стринга и вся задача упрощается.

Если Вы ставите перед собой задачу адаптации программы на русский язык, то Вам надо прежде всего две вещи:

- найти, где в программе находится шрифт;

- найти, где в программе находятся тексты сообщений, меню и т.п.

Шрифт можно искать по какому-либо символу, например, по восклицательному знаку. Поскольку как бы шрифт ни изображался, восклицательный знак в основной своей части представляет вертикальную палочку, то в записи его шаблона должны присутствовать подряд по крайней мере три одинаковых байта.

Напишите программу на поиск по памяти трех одинаковых байтов, отличных от нуля, запустите ее и проверьте те места, на которые она Вам укажет.

Программу на поиск текстов сообщений также несложно написать. Логика ее работы может быть такой. Найти все случаи, когда подряд следуют пять байтов, каждый из которых больше 40 (28 HEX), но меньше 122 (7A HEX).

Задач перед Вами может стоять сколько угодно - от желания постичь премудрости машинного программирования до организации опроса по внешнему порту монетоприемника с целью выяснения пикантного вопроса "А не забыл ли заигравшийся клиент опустить очередные двадцать копеек?", и все, конечно, осветить невозможно. Думайте, ставьте тесты, анализируйте результаты и, насколько возможно, автоматизируйте свою работу. Поручайте компьютеру самому производить поиски, сравнения, ставить эксперименты.

Мы не даем тексты несложных поисковых программ, так как полагаем, что Вы легко их напишете сами, но на всякий случай, если у Вас нет под рукой удобного ДИСАССЕМБЛЕРа, даем листинг программы шестнадцатиричного вывода по восемь кодов в строке.

HEXOUT

~~~~~~

```
10 INPUT "begin",beg
20 INPUT "end",end
30 DIM a$(8,3),DIM d(8)
40 FOR i=beg TO end STEP 8
50 LET b$=""
60 FOR k=1 TO 8
70 LET d(k) = PEEK(i+k-1)
80 GO SUB
90 LET b$=b$+a$(k)
100 NEXT k
110 GO SUB 1000
120 PRINT w$;b$
130 PRINT
140 NEXT i
500 LET h=INT(D(k)/16)
510 LET l=d(k)-h*16
520 LET h$=CHR$(h+48+7*(h>9))
530 LET l$=CHR$(l+48+7*(l>9))
540 LET a$(k)=h$+l$+"_"
550 RETURN
1000 LET a1=INT(i/16/16/16)
1010 LET rez=i-a1*16*16*16
1020 LET a2=INT(rez/16/16)
1030 LET rez=rez-a2*16*16
1040 LET a3=INT(rez/16)
1050 LET a4=rez-a3*16
1060 LET m$=CHR$(a1+48+7*(a1>9))
1060 LET m$=CHR$(a1+48+7*(a1>9))
1070 LET n$=CHR$(a2+48+7*(a1>9))
1080 LET p$=CHR$(a3+48+7*(a1>9))
1090 LET q$=CHR$(a4+48+7*(a1>9))
1100 LET w$=a$ + n$ + p$ + q$ + "__"
1110 RETURN
```
~~~~~~

7. ОБЗОР ТИПИЧНЫХ ОШИБОК, ВОЗНИКАЮЩИХ ПРИ ПРОГРАММИРОВАНИИ В МАШИННЫХ КОДАХ.

Классификация ошибок.

1. Перестановка операндов.
2. Неправильное использование флагов.
3. Путаница в использовании регистров и регистровых пар.
4. Путаница в работе с адресами и с данными.
5. Неправильная обработка массивов.
6. Неучет косвенных эффектов.
7. Неправильное задание начальных условий.
8. Ошибки, связанные с неправильной организацией программ.

7.1. Перестановка операндов.

Здесь наиболее характерны три типа ошибок. Чтобы избежать их, надо твердо помнить три правила.

1. В командах копирования регистров, например LD D,reg содержимое регистра REG копируется в регистр D, а не наоборот. Это одна из самых частых ошибок при программировании в машинных кодах.

2. Часто забывают, что 16-битные данные хранятся в памяти в следующем порядке: сначала младший байт, а потом старший. Особенно часто ошибка возникает в операциях загрузки регистров и в операциях между стеком и регистровыми парами.

3. Команда сравнения CP во время работы вычитает свой операнд из аккумулятора, а не наоборот. Флаги выставляются так, как будто было выполнено вычитание A-N.

7.2. Неправильное использование флагов.

1. Команды загрузки регистров LD не влияют на флаги. Чтобы установить флаги для последующей проверки, надо применять операции AND, DEC, INC, OR, XOR.

2. Часто путают состояние флага нуля (Z). Если после операции сравнения операнды оказываются равными, т.е. разность между ними равна нулю, то флаг включается, т.е. устанавливается

1. Возникает мнемонический конфуз "если результат операции - ноль, то флаг нуля - не ноль". Это приводит к неправильному применению JR NZ, N и JR Z,N.

3. В операциях сравнения CP флаг C указывает, какой операнд больше. Если аккумулятор больше, то флаг выключен, если больше операнд, то включен, но если они равны, то флаг выключается. Получается, что по этому флагу проверяется альтернатива для аккумулятора: БОЛЬШЕ ИЛИ РАВЕН/МЕНЬШЕ.

А если Вам надо проверить альтернативу БОЛЬШЕ/МЕНЬШЕ ИЛИ РАВЕН? Тогда перед операцией сравнения Вы должны добавить единицу к операнду или вычесть единицу из аккумулятора.

4. Программисты хорошо помнят, что в арифметических операциях с 16-разрядными регистрами DEC и INC не влияют на флаги вообще, а ADD влияет только на флаг переноса, но забывают, что ADC и SBC влияют на все флаги.

7.3. Путаница с регистрами и регистровыми парами.

Здесь ошибки возникают, когда пытаются применить для 8-разрядных операндов команды, предназначенные для 16-разрядных и наоборот. К счастью, ошибки этой группы возникают только при программировании в ассемблирующей программе, а она при компиляции сама найдет эти ошибки и укажет на них. Тем не менее, надо помнить несколько простых правил и руководствоваться ими:

ADC, ADD, DEC, INC, LD, SBC могут использоваться как с 8-битными операндами, так и с 16-битными регистровыми парами, причем ADD, DEC, INC и LD могут также применяться и в работе с индексными регистрами.

AND, OR, SUB, XOR применимы только к 8-битным операндам.

EX, POP, PUSH - применимы только к 16-битным операндам.

Часто пользуются косвенной адресацией через регистровую пару HL. Например, LD A,(HL); LD B,(HL)... и т.п. Но иногда делают и вполне законную, но не очень корректную операцию LD L,(HL), после которой нарушается содержимое HL. То же относится и к операции LD H,(HL).

7.4. Путаница в работе с адресами и данными.

Это самая распространенная ошибка при программировании в машинных кодах или на языке АССЕМБЛЕРа. Так, LD при выполнении копирования данных из регистра в память или из памяти в регистр всегда требует указания адреса. Этот адрес должен стоять в скобках. Короче говоря, все, что не стоит в скобках, АССЕМБЛЕР воспринимает как данные, а все, что стоит в скобках - как адреса. Если Вы забудете поставить скобки, то результат может быть совершенно непредсказуемым.

Часто возникает следующая ошибка. Допустим, Вы имеете набор процедур для выполнения каких-либо операций. Допустим, где-то в памяти Вами организована таблица адресов входа в эти процедуры. Правильная последовательность действий при этом такова: взять адрес из таблицы, поместив, допустим в HL, - здесь он фигурирует как данные; а далее стартовать процедуру с того адреса, на который указывает HL - здесь он уже выступает как адрес. На практике же часто пытаются стартовать процедуру переходом в таблицу, где содержится настоящий адрес старта. Это адрес старта, но это не место старта. Место старта находится там, куда указывает адрес старта.

Запомните также простое правило: DJNZ, JP, JR и CALL всегда требуют после себя указания адреса (или величины смещения, что в принципе одно и то же).

7.5. Неправильная обработка массивов.

Наиболее типичными ошибками здесь являются:

- выполнение лишнего цикла вычислений;
- недовыполнение цикла вычислений.

Обратите серьезное внимание на широкоизвестный факт, что если массив начинается с адреса BASE и кончается в адресе BASE+N, то этот массив имеет N+1 элемент. В этом и кроется основная причина ошибок. Забывают использовать либо первый, либо последний элемент. С другой стороны, если Ваш массив имеет N элементов, то он должен закончиться в адресе BASE+N-1. Часто же продолжают вычисления до адреса BASE+N, т.е. "прихватывают" лишний элемент.

7.6. Неучет косвенных эффектов.

Вот список косвенных эффектов, которые необходимо учитывать при программировании. Невнимание к этим особенностям приводит к ошибкам.

1. Все логические операции кроме CPL сбрасывают флаг CARRY.

2. В операциях LD rp,(ADDR); LD (ADDR),rp; LD xy,(ADDR); LD (ADDR),xy участвует не только указанный адрес, но и адрес, следующий за ним.

3. Операции POP, PUSH, CALL, RET, RET, RETN, RST влияют на указатель стека.

4. Операции CALL и RST засылают адрес на стек.

5. Операция DJNZ относится не к аккумулятору, как большинство других операций, имя регистра в которых не указывается, а к регистру B.

6. Команды блочного поиска, сравнения, перемещения байтов оказывают косвенное влияние на содержимое регистровых пар BC, DE, HL.

7. В командах LDD, LDI, CPD, CPDR, CPI, CPIR флаг P/O используется не по назначению, а служит для указания на то, что счетчик байтов в BC уменьшился до нуля.

8. Команды ротации в BCD-арифметике RRD, RLD вовлекают в ротацию содержимое регистра HL.

7.7. Неправильное задание начальных условий.

1. Не думайте, что если Вы не используете какие-либо ячейки памяти, то там должен быть 0. В начале работы программы, при инициализации желательно выставлять в рабочей области ОЗУ начальные значения.

2. При начале работы программы установите в регистрах и во флагах начальные значения, чтобы избежать неопределенностей.

3. Прежде, чем давать команду косвенной адресации, потрудитесь выставить адрес в регистре, через который производится адресация. Не полагайтесь на то, что он уже давно там есть.

7.8. Ошибки, связанные с неправильной организацией программ.
~~~~~~

1. Иногда программисты забывают сохранить результат. Выполнив расчеты и получив в итоге результат в аккумуляторе, тут же засылают в аккумулятор новые данные для других вычислений.

2. Внимательно смотрите за всеми ветвлениями в программе. Бывает, что управление передается процедурам инициализации программы, хотя это вовсе и не нужно.

3. В сложных программах нередко оказывается запутанным порядок исполнения процедур. Если Вы плохо разбираетесь в структуре собственной программы, может произойти исполнение блоков, которые исполнять не надо. Не забывайте ставить вокруг них обходные пути.
~~~~~~

ЧАСТЬ III

СПРАВОЧНИК ПО ПРОГРАММИРОВАНИЮ В МАШИННЫХ КОДАХ.

ВВЕДЕНИЕ

Основное назначение данного справочника - дать удобное подручное средство поддержки повседневной практической работы. Он может быть полезен как при работе с первой, так и со второй частью данной книги, но еще больше он Вам пригодится, когда Вы перейдете к самостоятельному разбору фирменных программ и составлению собственных процедур.

Весьма полезным, на наш взгляд, является двойное представление команд машинного кода. В табл.1 они приведены, отсортированными по кодам, а в табл. 2 - структурированы по группам в соответствии со своим целевым назначением. Для тех, кто самостоятельно пишет программы в машинных кодах, безусловно будут необходимы сведения о косвенном влиянии операций на флаги регистра F и справочные данные по длительности команд, измеряемой в тактах процессора; они приведены в табл.2.

Наши предложения по русификации компьютера, приведенные в таблицах шестой главы, не следует считать истиной в последней инстанции, но поскольку этот вопрос является одинаково важным как для производителей программного обеспечения, так и для его потребителей, мы надеемся, что наши рекомендации будут приняты, тем более, что многие ими уже давно пользуются.

1. СИСТЕМА КОМАНД КОМПЬЮТЕРОВ ТИПА "ZX-СПЕКТРУМ"

Таблица 1.

Код		Символ	А С С Е М Б Л Е Р		
Десят.	Шестнад.	в БЕЙСИКЕ	-	После CB	После ED
0	00	не используется	NOP	RLC B	
1	01	не используется	LD BC,NN	RLC C	
2	02	не используется	LD (BC),A	RLC D	
3	03	не используется	INC BC	RLC E	
4	04	не используется	INC B	RLC H	
5	05	не используется	DEC B	RLC L	
6	06	PRINT запятая*	LD B,N	RLC (HL)	
7	07	EDIT	RLCA	RLC A	
8	08	курсор влево	EX AF,AF	RRC A	
9	09	курсор вправо	ADD HL,BC	RRC B	
10	0A	курсор вниз	LD A,(BC)	RRC D	
11	0B	курсор вверх	DEC BC	RRC E	
12	0C	DELETE	INC C	RRC H	
13	0D	ENTER	DEC C	RRC L	
14	0E	число	LD C,N	RRC (HL)	
15	0F	не используется	RRCA	RRC A	
16	10	управление INK	DJNZ S	RL B	
17	11	управление PAPER	LD DE,NN	RL C	
18	12	управление FLASH	LD (DE),A	RL D	
19	13	управл. BRIGHT	INC DE	RL E	
20	14	управл. INVERSE	INC D	RL H	
21	15	управление OVER	DEC D	RL L	
22	16	управление AT	LD D,N	RL (HL)	
23	17	управление TAB	RLA	RL A	
24	18	не используется	JR S	RR B	
25	19	не используется	ADD HL,DE	RR C	
26	1A	не используется	LD A,(DE)	RR D	
27	1B	не используется	DEC DE	RR E	
28	1C	не используется	INC E	RR H	
29	1D	не используется	DEC E	RR L	
30	1E	не используется	LD E,N	RR (HL)	
31	1F	не используется	RRA	RR A	
32	20	пробел	JR NZ,S	SLA B	
33	21	!	LD HL,NN	SLA C	
34	22	"	LD (NN),HL	SLA D	
35	23	#	INC HL	SLA E	
36	24	$	INC H	SLA H	

*Примечание: Символы 6...23 в БЕЙСИКе выполняют роль управляющих кодов для печати на экране.

Продолжение табл.1.

Код		Символ	А С С Е М Б Л Е Р		
Десят.	Шестнад.	в БЕЙСИКЕ	-	После CB	После ED
37	25	%	DEC H	SLA L	
38	26	&	LD H,N	SLA (HL)	
39	27	'	DAA	SLA A	
40	28	(	JR Z,S	SRA B	
41	29	)	ADD HL,HL	SRA C	
42	2A	*	LD HL,(NN)	SRA D	
43	2B	+	DEC HL	SRA E	
44	2C	,	INC L	SRA H	
45	2D	_	DEC L	SRA L	
46	2E	.	LD L,N	SRA HL	
47	2F	/	CPL	SRA A	
48	30	0	JR NC,S		
49	31	1	LD SP,NN		
50	32	2	LD (NN),A		
51	33	3	INC SP		
52	34	4	INC (HL)		
53	35	5	DEC (HL)		
54	36	6	LD (HL),N		
55	37	7	SCF		
56	38	8	JR C,S	SRL B	
57	39	9	ADD HL,SP	SRL C	
58	3A	:	LD A,(NN)	SRL D	
59	3B	;	DEC SP	SRL E	
60	3C	<	INC A	SRL H	
61	3D	=	DEC A	SRL L	
62	3E	>	LD A,N	SRL (HL)	
63	3F	?	CCF	SRL A	
64	40	@	LD B,B	BIT 0,B	IN B,(C)
65	41	A	LD B,C	BIT 0,C	OUT (C),B
66	42	B	LD B,D	BIT 0,D	SBC HL,BC
67	43	C	LD B,E	BIT 0,E	LD (NN),BC
68	44	D	LD B,H	BIT 0,H	NEG
69	45	E	LD B,L	BIT 0,L	RETN
70	46	F	LD B,(HL)	BIT 0,(HL)	IM 0
71	47	G	LD B,A	BIT 0,A	LD I,A
72	48	H	LD C,B	BIT 1,B	IN C,(C)
73	49	I	LD C,C	BIT 1,C	OUT (C),C
74	4A	J	LD C,D	BIT 1,D	ADC HL,BC
75	4B	K	LD C,E	BIT 1,E	LD BC,(NN)
76	4C	L	LD C,H	BIT 1,H	
77	4D	M	LD C,L	BIT 1,L	RETI
78	4E	N	LD C,(HL)	BIT 1,(HL)	

Продолжение табл.1.

Код		Символ	А С С Е М Б Л Е Р		
Десят.	Шестнад.	в БЕЙСИКЕ	-	После CB	После ED
79	4F	O	LD C,A	BIT 1,A	LD R,A
80	50	P	LD D,B	BIT 2,B	IN D,(C)
81	51	Q	LD D,C	BIT 2,C	OUT (C),D
82	52	R	LD D,D	BIT 2,D	SBC HL,DE
83	53	S	LD D,E	BIT 2,E	LD (NN),DE
84	54	T	LD D,H	BIT 2,H	
85	55	U	LD D,L	BIT 2,L	
86	56	V	LD D,(HL)	BIT 2,(HL)	IM 1
87	57	W	LD D A	BIT 2,A	LD A,I
88	58	X	LD E,B	BIT 3,B	IN E,(C)
89	59	Y	LD E,C	BIT 3,C	OUT (C),E
90	5A	Z	LD E,D	BIT 3,D	ADC HL,DE
91	5B	[	LD E,E	BIT 3,E	LD DE,(NN)
92	5C	/	LD E,H	BIT 3,H	
93	5D	]	LD E,L	BIT 3,L	
94	5E	...	LD E,(HL)	BIT 3,(HL)	IM 2
95	5F	_	LD E,A	BIT 3,A	LD A,R BC
96	60	...	LD H,B	BIT 4,B	IN H,(C)
97	61	a	LD H,C	BIT 4,C	OUT (C),H
98	62	b	LD H,D	BIT 4,D	SBC HL,HL
99	63	c	LD H,E	BIT 4,E	LD (NN),HL
100	64	d	LD H,H	BIT 4,H	
101	65	e	LD H,L	BIT 4,L	
102	66	f	LD H,(HL)	BIT 4,(HL)	
103	67	g	LD H,A	BIT 4,A	RRD
104	68	h	LD L,B	BIT 5,B	IN L,(C)
105	69	i	LD L,C	BIT 5,C	OUT (C),L
106	6A	j	LD L,D	BIT 5,D	ADC HL,HL
107	6B	k	LD L,EHL)	BIT 5,E	LD HL,(NN)
108	6C	l	LD L,H	BIT 5,H	
109	6D	m	LD L,L	BIT 5,L	
110	6E	n	LD L,(HL)	BIT 5,(HL)	
111	6F	o	LD L,A	BIT 5,A	RLD
112	70	p	LD (HL),B	BIT 6,B	IN F,(C)
113	71	q	LD (HL),C	BIT 6,C	
114	72	r	LD (HL),D	BIT 6,D	SBC HL,SP
115	73	s	LD (HL),E	BIT 6,E	LD (NN),SP
116	74	t	LD (HL),H	BIT 6,H	
117	75	u	LD (HL),L	BIT 6,L	
118	76	v	HALT	BIT 6,(HL)	
119	77	w	LD (HL),A	BIT 6,A	
120	78	x	LD A,B	BIT 7,B	IN A,(C)

Продолжение табл.1.

Код		Символ	А С С Е М Б Л Е Р		
Десят.	Шестнад.	в БЕЙСИКЕ	-	После CB	После ED
121	79	y	LD A,C	BIT 7,C	OUT (C),A
122	7A	z	LD A,D	BIT 7,D	ADC HL,SP
123	7B	{	LD A,E	BIT 7,E	LD (SP),NN
124	7C	...	LD A,H	BIT 7,H	
125	7D	}	LD A,L	BIT 7,L	
126	7E	...	LD A,(HL)	BIT 7,(HL)	
127	7F	...	LD A,A	BIT 7,A	
128	80	...	ADD A,B	RES 0,B	
129	81	...	ADD A,C	RES 0,C	
130	82	...	ADD A,D	RES 0,D	
131	83	...	ADD A,E	RES 0,E	
132	84	...	ADD A,H	RES 0,H	
133	85	...	ADD A,L	RES 0,L	
134	86	...	ADD A,(HL)	RES 0,(HL)	
135	87	...	ADD A,A	RES 0,A	
136	88	...	ADC A,B	RES 1,B	
137	89	...	ADC A,C	RES 1,C	
138	8A	...	ADC A,D	RES 1,D	
139	8B	...	ADC A,E	RES 1,E	
140	8C	...	ADC A,H	RES 1,H	
141	8D	...	ADC A,L	RES 1,L	
142	8E	...	ADC A,(HL)	RES 1,(HL)	
143	8F	...	ADC A,A,D	RES 1,A	
144	90	граф. (A)	SUB B	RES 2,B	
145	91	граф. (B)	SUB C	RES 2,C	
146	92	граф. (C)	SUB D	RES 2,D	
147	93	граф. (D)	SUB E	RES 2,E	
148	94	граф. (E)	SUB H	RES 2,H	
149	95	граф. (F)	SUB L	RES 2,L	
150	96	граф. (G)	SUB (HL)	RES 2,(HL)	
151	97	граф. (H)	SUB A	RES 2,A	
152	98	граф. (I)	SBC A,B	RES 3,B	
153	99	граф. (J)	SBC A,C	RES 3,C	
154	9A	граф. (K)	SBC A,D	RES 3,D	
155	9B	граф. (L)	SBC A,E	RES 3,E	
156	9C	граф. (M)	SBC A,H	RES 3,H	
157	9D	граф. (N)	SBC A,L	RES 3,L	
158	9E	граф. (O)	SBC A,(HL)	RES 3,(HL)	
159	9F	граф. (P)	SBC A,A	RES 3,A	
160	A0	граф. (Q)	AND B	RES 4,B	
161	A1	граф. (R)	AND C	RES 4,C	
162	A2	граф. (S)	AND D	RES 4,D	

Продолжение табл.1.

Код		Символ	А С С Е М Б Л Е Р		
Десят.	Шестнад.	в БЕЙСИК	-	После CB	После ED
163	A3	графическое(T)	AND E	RES 4,E	
164	A4	графическое(U)	AND H	RES 4,H	
165	A5	RND	AND L	RES 4,L	
166	A6	INKEY$	AND (HL)	RES 4,(HL)	
167	A7	PI	AND A	RES 4,A	
168	A8	FN	XOR B	RES 5,B	LDD
169	A9	POINT	XOR C	RES 5,C	CPD
170	AA	SCREEN$	XOR D	RES 5,D	IND
171	AB	ATTR	XOR E	RES 5,E	OUTD
172	AC	AT	XOR H	RES 5,H	
173	AD	TAB	XOR L	RES 5,L	
174	AE	VAL$	XOR (HL)	RES 5,(HL)	
175	AF	CODE	XOR A	RES 5,A	
176	B0	VAL	OR B	RES 6,B	LDIR
177	B1	LEN	OR C	RES 6,C	CPIR
178	B2	SIN	OR D	RES 6,D	INIR
179	B3	COS	OR E	RES 6,E	OTIR
180	B4	TAN	OR H	RES 6,H	
181	B5	ASN	OR L	RES 6,L	
182	B6	ACS	OR (HL)	RES 6,(HL)	
183	B7	ATN	OR A	RES 6,A	
184	B8	LN	CP B	RES 7,B	LDDR
185	B9	EXP	CP C	RES 7,C	CPDR
186	BA	INT	CP D	RES 7,D	INDR
187	BB	SQR	CP E	RES 7,E	OTDR
188	BC	SGN	CP H	RES 7,H	
189	BD	ABS	CP L	RES 7,L	
190	BE	PEEK	CP (HL)	RES 7,(HL)	
191	BF	IN	CP A	RES 7,A	
192	C0	USR	RET NZ	SET 0,B	
193	C1	STR$	POP BC	SET 0,C	
194	C2	CHR$	JP NZ,NN	SET 0,D	
195	C3	NOT	JP NN	SET 0,E	
196	C4	BIN	CALL NZ,NN	SET 0,H	
197	C5	OR	PUSH BC	SET 0,L	
198	C6	AND	ADD A,N	SET 0,(HL)	
199	C7	<==	RST 0	SET 0,A	
200	C8	>==	RET Z	SET 1,B	
201	C9	<>	RET	SET 1,C	
202	CA	LINE	JP Z,NN	SET 1,D	
203	CB	THEN		SET 1,E	
204	CC	TO	CALL Z,NN	SET 1,H	

Продолжение табл.1.

Код		Символ	А С С Е М Б Л Е Р		
Десят.	Шестнад.	в БЕЙСИК	-	После CB	После ED
205	CD	STEP	CALL NN	SET 1,L	
206	CE	DEF FN	ADC A,N	SET 1,(HL)	
207	CF	CAT	RST 8	SET 1,A	
208	D0	FORMAT	RET NC	SET 2,B	
209	D1	MOVE	POP DE	SET 2,C	
210	D2	ERASE	JP NC,NN	SET 2,D	
211	D3	OPEN#	OUT (N),A	SET 2,E	
212	D4	CLOSE#	CALL NC,NN	SET 2,H	
213	D5	MERGE	PUSH DE	SET 2,L	
214	D6	VERIFY	SUB N	SET 2,(HL)	
215	D7	BEEP	RST 10	SET 2,A	
216	D8	CIRCLE	RET C	SET 3,B	
217	D9	INK	EXX	SET 3,C	
218	DA	PAPER	JP C,NN	SET 3,D	
219	DB	FLASH	IN A,(N)	SET 3,E	
220	DC	BRIGHT	CALL C,NN	SET 3,H	
221	DD	INVERSE	Предшествует операциям, в которых вместо регистра HL участвует IX.	SET 3,L	
222	DE	OVER	SBC A,N	SET 3,(HL)	
223	DF	OUT	RST 18	SET 3,A	
224	E0	LPRINT	RET PO	SET 4,B	
225	E1	LLIST	POP HL	SET 4,C	
226	E2	STOP	JP PO,NN	SET 4,D	
227	E3	READ	EX(SP),HL	SET 4,E	
228	E4	DATA	CALL PO,NN	SET 4,H	
229	E5	RESTORE	PUSH HL	SET 4,L	
230	E6	NEW	AND N	SET 4,(HL)	
231	E7	BORDER	RST 20	SET 4,A	
232	E8	CONTINUE	RET PE	SET 5,B	
233	E9	DIM	JP (HL)	SET 5,C	
234	EA	REM	JP PE,NN	SET 5,D	
235	EB	FOR	EX DE,HL	SET 5,E	
236	EC	GO TO	CALL PE,NN	SET 5,H	
237	ED	GO SUB		SET 5,L	
238	EE	INPUT	XOR N	SET 5,(HL)	
239	EF	LOAD	RST 28	SET 5,A	
240	F0	LIST	RET P	SET 6,B	
241	F1	LET	POP AF	SET 6,C	
242	F2	PAUSE	JP P,NN	SET 6,D	
243	F3	NEXT	DI	SET 6,E	
244	F4	POKE	CALL P,NN	SET 6,H	

Продолжение табл.1.

Код		Символ	А С С Е М Б Л Е Р		
Десят.	Шестнад.	в БЕЙСИК	-	После CB	После ED
245	F5	PRINT	PUSH AF	SET 6,L	
246	F6	PLOT	OR N	SET 6,(HL)	
247	F7	RUN	RST 30	SET 6,A	
248	F8	SAVE	RET M	SET 7,B	
249	F9	RANDOMIZE	LD SP,HL	SET 7,C	
250	FA	IF	JP M,NN	SET 7,D	
251	FB	CLS	EI	SET 7,E	
252	FC	DRAW	CALL M,NN	SET 7,H	
253	FD	CLEAR	Предшествует операциям с регистром IY.	SET 7,L	
254	FE	RETURN	CP N	SET 7,(HL)	
255	FF	COPY	RST 38	SET 7,A	

2. СИСТЕМА КОМАНД ПРОЦЕССОРА Z-80

Условные обозначения.
- флаги не изменяются; * флаг устанавливается в соответствии с результатом операции; 1 флаг включается; 0 флаг выключается; ? флаг неопределен; P флаг переполнения/четности работает как флаг четности; V флаг переполнения/четности работает как флаг переполнения.

Команды загрузки числа в регистр. Таблица 2.1.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
LD A,N	3E N	-	-	-	-	7	
LD B,N	06 N	-	-	-	-	7	
LD C,N	0E N	-	-	-	-	7	
LD D,N	16 N	-	-	-	-	7	
LD E,N	1E N	-	-	-	-	7	
LD H,N	26 N	-	-	-	-	7	
LD L,N	2E N	-	-	-	-	7	

Команды загрузки числа в регистровую пару. Таблица 2.2.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
LD HL,NN	21 N N	-	-	-	-	10	
LD DE,NN	11 N N	-	-	-	-	10	
LD BC,NN	01 N N	-	-	-	-	10	
LD IX,NN	DD 21 N N	-	-	-	-	14	
LD IY,NN	FD 21 N N	-	-	-	-	14	
LD SP,NN	31 N N	-	-	-	-	10	

Таблица 2.3.

Команды копирования содержимого одиночных регистров.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
LD A,A	7F	-	-	-	-	4	
LD A,H	7C	-	-	-	-	4	
LD A,L	7D	-	-	-	-	4	
LD A,B	78	-	-	-	-	4	
LD A,C	79	-	-	-	-	4	
LD A,D	7A	-	-	-	-	4	
LD A,E	7B	-	-	-	-	4	
LD H,A	67	-	-	-	-	4	
LD H,H	64	-	-	-	-	4	
LD H,L	65	-	-	-	-	4	
LD H,B	60	-	-	-	-	4	
LD H,C	61	-	-	-	-	4	
LD H,D	62	-	-	-	-	4	
LD H,E	63	-	-	-	-	4	
LD L,A	6F	-	-	-	-	4	
LD L,H	6C	-	-	-	-	4	
LD L,L	6D	-	-	-	-	4	
LD L,B	68	-	-	-	-	4	
LD L,C	69	-	-	-	-	4	
LD L,D	6A	-	-	-	-	4	
LD L,E	6B	-	-	-	-	4	
LD B,A	47	-	-	-	-	4	
LD B,H	44	-	-	-	-	4	
LD B,L	45	-	-	-	-	4	
LD B,B	40	-	-	-	-	4	

Продолжение таблицы 2.3.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
LD B,C	41	-	-	-	-	4	
LD B,D	42	-	-	-	-	4	
LD B,E	43	-	-	-	-	4	
LD C,A	4F	-	-	-	-	4	
LD C,H	4C	-	-	-	-	4	
LD C,L	4D	-	-	-	-	4	
LD C,B	48	-	-	-	-	4	
LD C,C	49	-	-	-	-	4	
LD C,D	4A	-	-	-	-	4	
LD C,E	4B	-	-	-	-	4	
LD D,A	57	-	-	-	-	4	
LD D,H	54	-	-	-	-	4	
LD D,L	55	-	-	-	-	4	
LD D,B	50	-	-	-	-	4	
LD D,C	51	-	-	-	-	4	
LD D,D	52	-	-	-	-	4	
LD D,E	53	-	-	-	-	4	
LD E,A	5F	-	-	-	-	4	
LD E,H	5C	-	-	-	-	4	
LD E,L	5D	-	-	-	-	4	
LD E,B	58	-	-	-	-	4	
LD E,C	59	-	-	-	-	4	
LD E,D	5A	-	-	-	-	4	
LD E,E	5B	-	-	-	-	4	
LD A,I	ED 57	-	*	*	*	9	Во флаг P/V копируется состояние триггера прерывания IFF.
LD I,A	ED 47	-	-	-	-	9	
LD A,R	ED 5F	-	*	*	*	9	Во флаг P/V копируется состояние триггера прерывания IFF.
LD R,A	ED 4F	-	-	-	-	9	

Копирование содержимого регистровых пар. Таблица 2.4.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
LD SP,HL	F9	-	-	-	-	6	
LD SP,IX	DD F9	-	-	-	-	10	
LD SP,IY	FD F9	-	-	-	-	10	

Загрузка регистров из памяти прямой адресацией. Таблица 2.5.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
LD A,(NN)	3A N N	-	-	-	-	13	
LD SP,(NN)	ED 7B N N	-	-	-	-	20	
LD HL,(NN)	2A N N	-	-	-	-	16	
	ED 6B N N	-	-	-	-	20	
LD BC,(NN)	ED 4B N N	-	-	-	-	20	
LD DE,(NN)	DD 5B N N	-	-	-	-	20	
LD IX,(NN)	DD 2A N N	-	-	-	-	20	
LD IY,(NN)	FD 2A N N	-	-	-	-	20	

Загрузка регистров из памяти косвенной адресацией.

Таблица 2.6.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
LD A,(HL)	7E	-	-	-	-	7	
LD A,(BC)	0A	-	-	-	-	7	
LD A,(DE)	1A	-	-	-	-	7	
LD H,(HL)	66	-	-	-	-	7	
LD L,(HL)	6E	-	-	-	-	7	
LD B,(HL)	46	-	-	-	-	7	
LD C,(HL)	4E	-	-	-	-	7	
LD D,(HL)	56	-	-	-	-	7	
LD E,(HL)	5E	-	-	-	-	7	

Загрузка регистров из памяти индексной адресацией. Табл. 2.7.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
LD A,(IX+S)	DD 7E S	-	-	-	-	19	
LD H,(IX+S)	DD 66 S	-	-	-	-	19	
LD L,(IX+S)	DD 6E S	-	-	-	-	19	
LD B,(IX+S)	DD 46 S	-	-	-	-	19	
LD C,(IX+S)	DD 4E S	-	-	-	-	19	
LD D,(IX+S)	DD 56 S	-	-	-	-	19	
LD E,(IX+S)	DD 5E S	-	-	-	-	19	
LD A,(IY+S)	FD 7E S	-	-	-	-	19	
LD H,(IY+S)	FD 66 S	-	-	-	-	19	
LD L,(IY+S)	FD 6E S	-	-	-	-	19	
LD B,(IY+S)	FD 46 S	-	-	-	-	19	
LD C,(IY+S)	FD 4E S	-	-	-	-	19	
LD D,(IY+S)	FD 56 S	-	-	-	-	19	
LD E,(IY+S)	FD 5E S	-	-	-	-	19	

Команды обмена. Таблица 2.8.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
EX DE,HL	EB	-	-	-	-	4	
EXX	D9	-	-	-	-	4	
EX AF,A'F'	08	-	-	-	-	4	

Запись в память прямой адресацией. Таблица 2.9.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
LD (NN),A	32 N N	-	-	-	-	13	
LD (NN),SP	ED 73 N N	-	-	-	-	20	
LD (NN),HL	22 N N	-	-	-	-	16	
	ED 63 N N	-	-	-	-	20	
LD (NN),BC	ED 43 N N	-	-	-	-	20	
LD (NN),DE	ED 53 N N	-	-	-	-	20	
LD (NN),IX	DD 22 N N	-	-	-	-	20	
LD (NN),IY	FD 22 N N	-	-	-	-	20	

Запись в память косвенной адресацией.

Таблица 2.10

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
LD (HL),A	77	-	-	-	-	7	
LD (BC),A	02	-	-	-	-	7	
LD (DE),A	12	-	-	-	-	7	
LD (HL),H	74	-	-	-	-	7	
LD (HL),L	75	-	-	-	-	7	
LD (HL),B	70	-	-	-	-	7	
LD (HL),C	71	-	-	-	-	7	
LD (HL),D	72	-	-	-	-	7	
LD (HL),E	73	-	-	-	-	7	
LD (HL),N	36 N	-	-	-	-	10	

Запись в память индексной адресацией.

Таблица 2.11.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
LD (IX+S),A	DD 77 S	-	-	-	-	19	
LD (IX+S),H	DD 74 S	-	-	-	-	19	
LD (IX+S),L	DD 75 S	-	-	-	-	19	
LD (IX+S),B	DD 70 S	-	-	-	-	19	
LD (IX+S),C	DD 71 S	-	-	-	-	19	
LD (IX+S),D	DD 72 S	-	-	-	-	19	
LD (IX+S),E	DD 73 S	-	-	-	-	19	
LD (IX+S),N	DD 36 S N	-	-	-	-	19	
LD (IY+S),A	FD 77 S	-	-	-	-	19	
LD (IY+S),H	FD 74 S	-	-	-	-	19	
LD (IY+S),L	FD 75 S	-	-	-	-	19	
LD (IY+S),B	FD 70 S	-	-	-	-	19	
LD (IY+S),C	FD 71 S	-	-	-	-	19	
LD (IY+S),D	FD 72 S	-	-	-	-	19	
LD (IY+S),E	FD 73 S	-	-	-	-	19	
LD (IY+S),N	FD 36 S N	-	-	-	-	19	

Команды простого сложения. Таблица 2.12.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
ADD A,N	C6 N	*	*	V	*	7	
ADD A,A	87	*	*	V	*	4	
ADD A,H	84	*	*	V	*	4	
ADD A,L	85	*	*	V	*	4	
ADD A,B	80	*	*	V	*	4	
ADD A,C	81	*	*	V	*	4	
ADD A,D	82	*	*	V	*	4	
ADD A,E	83	*	*	V	*	4	
ADD A,(HL)	86	*	*	V	*	7	
ADD A,(IX+S)	DD 86 S	*	*	V	*	19	
ADD A,(IY+S)	FD 86 S	*	*	V	*	19	
ADD HL,HL	29	*	-	-	-	11	
ADD HL,BC	09	*	-	-	-	11	
ADD HL,DE	19	*	-	-	-	11	
ADD HL,SP	39	*	-	-	-	11	
ADD IX,IX	DD 29	*	-	-	-	15	
ADD IX,BC	DD 09	*	-	-	-	15	
ADD IX,DE	DD 19	*	-	-	-	15	
ADD IX,SP	DD 39	*	-	-	-	15	
ADD IY,IY	FD 29	*	-	-	-	15	
ADD IY,BC	FD 09	*	-	-	-	15	
ADD IY,DE	FD 19	*	-	-	-	15	
ADD IY,SP	FD 39	*	-	-	-	15	

Команды приращения (инкремент). Таблица 2.13.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
INC A	3C	-	*	V	*	4	
INC H	24	-	*	V	*	4	
INC L	2C	-	*	V	*	4	
INC B	04	-	*	V	*	4	
INC D	14	-	*	V	*	4	
INC E	1C	-	*	V	*	4	
INC (HL)	34	-	*	V	*	11	
INC (IX+S)	DD 34 S	-	*	V	*	23	
INC (IY+S)	FD 34 S	-	*	V	*	23	

Продолжение таблицы 2.13

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
INC HL	23	-	-	-	-	6	
INC BC	03	-	-	-	-	6	
INC DE	13	-	-	-	-	6	
INC SP	33	-	-	-	-	6	
INC IX	DD 23	-	-	-	-	10	
INC IY	FD 23	-	-	-	-	10	

Команды сложения с учетом переноса. Таблица 2.14.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
ADC A,N	CE N	*	*	V	*	7	
ADC A,A	8F	*	*	V	*	4	
ADC A,H	8C	*	*	V	*	4	
ADC A,L	8D	*	*	V	*	4	
ADC A,B	88	*	*	V	*	4	
ADC A,C	89	*	*	V	*	4	
ADC A,D	8A	*	*	V	*	4	
ADC A,E	8B	*	*	V	*	4	
ADC A,(HL)	8E	*	*	V	*	7	
ADC A,(IX+S)	DD 8E S	*	*	V	*	19	
ADC A,(IY+S)	FD 8E S	*	*	V	*	19	
ADC HL,HL	ED 6A	*	-	-	-	15	
ADC HL,BC	ED 4A	*	-	-	-	15	
ADC HL,DE	ED 5A	*	-	-	-	15	
ADD HL,SP	ED 7A	*	-	-	-	11	

Команды простого вычитания. Таблица 2.15.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
SUB N	D6 N	*	*	V	*	7	
SUB A	97	*	*	V	*	4	
SUB H	94	*	*	V	*	4	
SUB L	95	*	*	V	*	4	

Продолжение таблицы 2.15.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
SUB B	90	*	*	V	*	4	
SUB C	91	*	*	V	*	4	
SUB D	92	*	*	V	*	4	
SUB E	93	*	*	V	*	4	
SUB (HL)	96	*	*	V	*	7	
SUB (IX+S)	DD 96 S	*	*	V	*	19	
SUB (IY+S)	FD 96 S	*	*	V	*	19	

Команды уменьшения (декремент). Таблица 2.16.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
DEC A	3D	-	*	V	*	4	
DEC H	25	-	*	V	*	4	
DEC L	2D	-	*	V	*	4	
DEC B	05	-	*	V	*	4	
DEC E	1D	-	*	V	*	7	
DEC (HL)	35	-	*	V	*	11	
DEC (IX+S)	DD 35 S	-	*	V	*	23	
DEC (IY+S)	FD 35 S	-	*	V	*	23	
DEC HL	2B	-	-	-	-	6	
DEC BC	0B	-	-	-	-	6	
DEC DE	1B	-	-	-	-	6	
DEC SP	3B	-	-	-	-	6	
DEC IX	DD 2B	-	-	-	-	10	
DEC IY	FD 2B	-	-	-	-	10	

Команды вычитания с учетом переноса. Таблица 2.17.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
SBC A,N	DE N	*	*	V	*	7	
SBC A,A	9F	*	*	V	*	4	
SBC A,H	9C	*	*	V	*	4	

Продолжение таблицы 2.17

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
SBC A,L	9D	*	*	V	*	4	
SBC A,B	98	*	*	V	*	4	
SBC A,C	99	*	*	V	*	4	
SBC A,D	9A	*	*	V	*	4	
SBC A,E	9B	*	*	V	*	4	
SBC A,(HL)	9E	*	*	V	*	7	
SBC A,(IX+S)	DD 9E S	*	*	V	*	19	
SBC A,(IY+S)	FD 9E S	*	*	V	*	19	
SBC HL,HL	ED 62	*	*	V	*	15	
SBC HL,BC	ED 42	*	*	V	*	15	
SBC HL,DE	ED 52	*	*	V	*	15	
SBC HL,SP	ED 72	*	*	V	*	15	

Команды сравнения. Таблица 2.18.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
CP N	FE N	*	*	V	*	7	
CP A	BF	*	*	V	*	4	
CP H	BC	*	*	V	*	4	
CP L	BD	*	*	V	*	4	
CP B	B8	*	*	V	*	4	
CP C	B9	*	*	V	*	4	
CP D	BA	*	*	V	*	4	
CP E	BB	*	*	V	*	4	
CP (HL)	BE	*	*	V	*	7	
CP (IX+S)	DD BE S	*	*	V	*	19	
CP (IY+S)	FD BE S	*	*	V	*	19	

Команды логики. Таблица 2.18.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
AND N	E6 N	*	*	P	*	7	
AND A	A7	O	*	P	*	4	

Продолжение таблицы 2.18

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
AND H	A4	O	*	P	*	4	
AND L	A5	O	*	P	*	4	
AND B	A0	O	*	P	*	4	
AND C	A1	O	*	P	*	4	
AND D	A2	O	*	P	*	4	
AND E	A3	O	*	P	*	4	
AND (HL)	A6	O	*	P	*	7	
AND (IX+S)	DD A6 S	O	*	P	*	19	
AND (IY+S)	FD A6 S	O	*	P	*	19	
OR N	F6 N	*	*	P	*	7	
OR A	B7	O	*	P	*	4	
OR H	B4	O	*	P	*	4	
OR L	B5	O	*	P	*	4	
OR B	B0	O	*	P	*	4	
OR C	B1	O	*	P	*	4	
OR D	B2	O	*	P	*	4	
OR E	B3	O	*	P	*	4	
OR (HL)	B6	O	*	P	*	7	
OR (IX+S)	DD B6 S	O	*	P	*	19	
OR (IY+S)	FD B6 S	O	*	P	*	19	
XOR N	EE N	*	*	P	*	7	
XOR A	AF	O	*	P	*	4	
XOR H	AC	O	*	P	*	4	
XOR L	AD	O	*	P	*	4	
XOR B	A8	O	*	P	*	4	
XOR C	A9	O	*	P	*	4	
XOR D	AA	O	*	P	*	4	
XOR E	AB	O	*	P	*	4	
XOR (HL)	AE	O	*	P	*	7	
XOR (IX+S)	DD AE S	O	*	P	*	19	
XOR (IY+S)	FD AE S	O	*	P	*	19	

Команды перехода. Таблица 2.20.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
JP NN	C3 N N	-	-	-	-	10	
JP (HL)	E9	-	-	-	-	4	
JP (IX)	DD E9	-	-	-	-	8	
JP (IY)	FD E9	-	-	-	-	8	
JP C,NN	DA N N	-	-	-	-	10	
JP NC,NN	D2 N N	-	-	-	-	10	
JP Z,NN	CA N N	-	-	-	-	10	
JP NZ,NN	C2 N N	-	-	-	-	10	
JP P,NN	F2 N N	-	-	-	-	10	
JP M,NN	FA N N	-	-	-	-	10	
JP PE,NN	EA N N	-	-	-	-	10	
JP PO,NN	E2 N N	-	-	-	-	10	
JR C,S	38 S	-	-	-	-	7	Если условие не выполняется
						12	Если условие выполняется
JR Z,S	28 S	-	-	-	-	7	Если условие не выполняется
						12	Если условие выполняется
JR NZ,S	20 S	-	-	-	-	7	Если условие не выполняется
						12	Если условие выполняется
DJNZ	10 S	-	-	-	-	8	Если B=0
						13	Если B<>0

Команды работы со стеком. Таблица 2.21.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
PUSH AF	F5	-	-	-	-	11	
PUSH BC	C5	-	-	-	-	11	
PUSH DE	D5	-	-	-	-	11	
PUSH HL	E5	-	-	-	-	11	
PUSH IX	DD E5	-	-	-	-	15	
PUSH IY	FD E5	-	-	-	-	15	
POP AF	F1	-	-	-	-	10	

Продолжение таблицы 2.21.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
POP BC	C1	-	-	-	-	10	
POP DE	D1	-	-	-	-	10	
POP HL	E1	-	-	-	-	10	
POP IX	DD E1	-	-	-	-	14	
POP IY	FD E1	-	-	-	-	14	
EX (SP),HL	E3	-	-	-	-	19	
EX (SP),IX	DD E3	-	-	-	-	23	
EX (SP),IY	FD E3	-	-	-	-	23	

Команды обращения к ПЗУ. Таблица 2.22.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
RST 0	C7	-	-	-	-	11	
RST 8	CF	-	-	-	-	11	
RST 10	D7	-	-	-	-	11	
RST 18	DF	-	-	-	-	11	
RST 20	E7	-	-	-	-	11	
RST 28	EF	-	-	-	-	11	
RST 30	F7	-	-	-	-	11	
RST 38	FF	-	-	-	-	11	

Команды вызова подпрограмм и возврата Таблица 2.23.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
CALL NN	CD N N	-	-	-	-	17	
CALL C,NN	DC N N	-	-	-	-	10/17	Условие выполняется/нет
CALL NC,NN	D4 N N	-	-	-	-	10/17	---"---"---"--
CALL Z,NN	CC N N	-	-	-	-	10/17	---"---"---"--
CALL M,NN	FC N N	-	-	-	-	10/17	---"---"---"--
CALL P,NN	F4 N N	-	-	-	-	10/17	---"---"---"--
CALL PE,NN	EC N N	-	-	-	-	10/17	---"---"---"--
CALL PO,NN	E4 N N	-	-	-	-	10/17	---"---"---"--

Продолжение таблицы 2.23

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
RET	C9	-	-	-	-	10	
RET C	D8	-	-	-	-	5/11	Условие выполняется/нет
RET NC	D0	-	-	-	-	5/11	---"---"---"--
RET Z	C8	-	-	-	-	5/11	---"---"---"--
RET NZ	C0	-	-	-	-	5/11	---"---"---"--
RET P	F0	-	-	-	-	5/11	---"---"---"--
RET M	F8	-	-	-	-	5/11	---"---"---"--
RET PE	E8	-	-	-	-	5/11	---"---"---"--
RET P0	E0	-	-	-	-	5/11	---"---"---"--

Команды сдвига битов. Таблица 2.24.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
SRL A	CB 3F	*	*	P	*	8	
SRL H	CB 3C	*	*	P	*	8	
SRL L	CB 3D	*	*	P	*	8	
SRL B	CB 38	*	*	P	*	8	
SRL C	CB 39	*	*	P	*	8	
SRL D	CB 3A	*	*	P	*	8	
SRL E	CB 3B	*	*	P	*	8	
SRL (HL)	CB 3E	*	*	P	*	15	
SRL (IX+S)	DD CB S 3E	*	*	P	*	23	
SRL (IY+S)	FD CB S 3E	*	*	P	*	23	
SRA A	CB 2F	*	*	P	*	8	
SRA H	CB 2C	*	*	P	*	8	
SRA L	CB 2D	*	*	P	*	8	
SRA B	CB 28	*	*	P	*	8	
SRA C	CB 29	*	*	P	*	8	
SRA D	CB 2A	*	*	P	*	8	
SRA E	CB 2B	*	*	P	*	8	
SRA (HL)	CB 2E	*	*	P	*	15	
SRA (IX+S)	DD CB S 2E	*	*	P	*	23	
SRA (IY+S)	FF CB S 2E	*	*	P	*	23	
SLA A	CB 27	*	*	P	*	8	
SLA H	CB 24	*	*	P	*	8	
SRA L	CB 25	*	*	P	*	8	

Продолжение таблицы 2.24

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
SRA B	CB 20	*	*	P	*	8	
SRA C	CB 21	*	*	P	*	8	
SRA D	CB 22	*	*	P	*	8	
SRA E	CB 23	*	*	P	*	8	
SRA (HL)	CB 26	*	*	P	*	15	
SRA (IX+S)	DD CB S 26	*	*	P	*	23	
SRA (IY+S)	FF CB S 26	*	*	P	*	23	

Команды ротации битов. Таблица 2.25.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
RL A	CB 17	*	*	P	*	8	
RL H	CB 14	*	*	P	*	8	
RL L	CB 15	*	*	P	*	8	
RL B	CB 10	*	*	P	*	8	
RL C	CB 11	*	*	P	*	8	
RL D	CB 12	*	*	P	*	8	
RL E	CB 13	*	*	P	*	8	
RL (HL)	CB 16	*	*	P	*	15	
RL (IX+S)	DD CB S 16	*	*	P	*	23	
RL (IY+S)	FD CB S 16	*	*	P	*	23	
RR A	CB 1F	*	*	P	*	8	
RR H	CB 1C	*	*	P	*	8	
RR L	CB 1D	*	*	P	*	8	
RR B	CB 18	*	*	P	*	8	
RR C	CB 19	*	*	P	*	8	
RR D	CB 1A	*	*	P	*	8	
RR E	CB 1B	*	*	P	*	8	
RR (HL)	CB 1E	*	*	P	*	15	
RR (IX+S)	DD CB S 1E	*	*	P	*	23	
RR (IY+S)	FD CB S 1E	*	*	P	*	23	
RLC A	CB 07	*	*	P	*	8	
RLC H	CB 04	*	*	P	*	8	
RLC L	CB 05	*	*	P	*	8	
RLC B	CB 00	*	*	P	*	8	

Продолжение таблицы 2.25.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
RLC C	CB 01	*	*	P	*	8	
RLC D	CB 02	*	*	P	*	8	
RLC E	CB 03	*	*	P	*	8	
RLC (HL)	CB 06	*	*	P	*	15	
RLC (IX+S)	DD CB S 06	*	*	P	*	23	
RLC (IY+S)	FD CB S 06	*	*	P	*	23	
RRC A	CB 0F	*	*	P	*	8	
RRC H	CB 0C	*	*	P	*	8	
RRC L	CB 0D	*	*	P	*	8	
RRC B	CB 08	*	*	P	*	8	
RRC C	CB 09	*	*	P	*	8	
RRC D	CB 0A	*	*	P	*	8	
RRC E	CB 0B	*	*	P	*	8	
RRC (HL)	CB 0E	*	*	P	*	15	
RRC (IX+S)	DD CB S 0E	*	*	P	*	23	
RRC (IY+S)	FD CB S 0E	*	*	P	*	23	
RLA	17	*	-	-	-	4	
RRA	1F	*	-	-	-	4	
RLCA	07	*	-	-	-	4	
RRCA	0F	*	-	-	-	4	
RLD	ED 6F	-	*	P	*	18	
RRD	ED 67	-	*	P	*	18	

Команды включения битов. Таблица 2.26.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
SET 0,A	CB C7	-	-	-	-	8	
SET 0,H	CB C4	-	-	-	-	8	
SET 0,L	CB C5	-	-	-	-	8	
SET 0,B	CB C0	-	-	-	-	8	
SET 0,C	CB C1	-	-	-	-	8	
SET 0,D	CB C2	-	-	-	-	8	
SET 0,E	CB C3	-	-	-	-	8	
SET (HL)	CB C6	-	-	-	-	15	
SET 0,(IX+S)	DD CB S C6	-	-	-	-	23	
SET 0,(IY+S)	FD CB S C6	-	-	-	-	23	
SET 1,A	CB CF	-	-	-	-	8	

Продолжение таблицы 2.26.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
SET 1,H	CB CC	-	-	-	-	8	
SET 1,L	CB CD	-	-	-	-	8	
SET 1,B	CB C8	-	-	-	-	8	
SET 1,C	CB C9	-	-	-	-	8	
SET 1,D	CB CA	-	-	-	-	8	
SET 1,E	CB CB	-	-	-	-	8	
SET 1,(HL)	CB CE	-	-	-	-	15	
SET 1,(IX+S)	DD CB S CE	-	-	-	-	23	
SET 1,(IY+S)	FD CB S CE	-	-	-	-	23	
SET 2,A	CB D7	-	-	-	-	8	
SET 2,H	CB D4	-	-	-	-	8	
SET 2,L	CB D5	-	-	-	-	8	
SET 2,B	CB D0	-	-	-	-	8	
SET 2,C	CB D1	-	-	-	-	8	
SET 2,D	CB D2	-	-	-	-	8	
SET 2,E	CB D3	-	-	-	-	8	
SET 2,(HL)	CB D6	-	-	-	-	15	
SET 2,(IX+S)	DD CB S D6	-	-	-	-	23	
SET 2,(IY+S)	FD CB S D6	-	-	-	-	23	
SET 3,A	CB DF	-	-	-	-	8	
SET 3,H	CB DC	-	-	-	-	8	
SET 3,L	CB DD	-	-	-	-	8	
SET 3,B	CB D8	-	-	-	-	8	
SET 3,C	CB D9	-	-	-	-	8	
SET 3,D	CB DA	-	-	-	-	8	
SET 3,E	CB DB	-	-	-	-	8	
SET 3,(HL)	CB DE	-	-	-	-	15	
SET 3,(IX+S)	DD CB S DE	-	-	-	-	23	
SET 3,(IY+S)	FD CB S DE	-	-	-	-	23	
SET 4,A	CB E7	-	-	-	-	8	
SET 4,H	CB E4	-	-	-	-	8	
SET 4,L	CB E5	-	-	-	-	8	
SET 4,B	CB E0	-	-	-	-	8	
SET 4,C	CB E1	-	-	-	-	8	
SET 4,D	CB E2	-	-	-	-	8	
SET 4,E	CB E3	-	-	-	-	8	
SET 4,(HL)	CB E6	-	-	-	-	15	
SET 4,(IX+S)	DD CB S E6	-	-	-	-	23	
SET 4,(IY+S)	FD CB S E6	-	-	-	-	23	
SET 5,A	CB EF	-	-	-	-	8	
SET 5,H	CB EC	-	-	-	-	8	

Продолжение таблицы 2.26.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
SET 5,L	CB ED	-	-	-	-	8	
SET 5,B	CB E8	-	-	-	-	8	
SET 5,C	CB E9	-	-	-	-	8	
SET 5,D	CB EA	-	-	-	-	8	
SET 5,E	CB EB	-	-	-	-	8	
SET 5,(HL)	CB EE	-	-	-	-	15	
SET 5,(IX+S)	DD CB S EE	-	-	-	-	23	
SET 5,(IY+S)	FD CB S EE	-	-	-	-	23	
SET 6,A	CB F7	-	-	-	-	8	
SET 6,H	CB F4	-	-	-	-	8	
SET 6,L	CB F5	-	-	-	-	8	
SET 6,B	CB F0	-	-	-	-	8	
SET 6,C	CB F1	-	-	-	-	8	
SET 6,D	CB F2	-	-	-	-	8	
SET 6,E	CB F3	-	-	-	-	8	
SET 6,(HL)	CB F6	-	-	-	-	15	
SET 6,(IX+S)	DD CB S F6	-	-	-	-	23	
SET 6,(IY+S)	FD CB S F6	-	-	-	-	23	
SET 7,A	CB FF	-	-	-	-	8	
SET 7,H	CB FC	-	-	-	-	8	
SET 7,L	CB FD	-	-	-	-	8	
SET 7,B	CB F8	-	-	-	-	8	
SET 7,C	CB F9	-	-	-	-	8	
SET 7,D	CB FA	-	-	-	-	8	
SET 7,E	CB FB	-	-	-	-	8	
SET 7,(HL)	CB FE	-	-	-	-	15	
SET 7,(IX+S)	DD CB S FE	-	-	-	-	23	
SET 7,(IY+S)	FD CB S FE	-	-	-	-	23	

Команды выключения битов Таблица 2.27.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
RES 0,A	CB 87	-	-	-	-	8	
RES 0,H	CB 84	-	-	-	-	8	
RES 0,L	CB 85	-	-	-	-	8	
RES 0,B	CB 80	-	-	-	-	8	
RES 0,C	CB 81	-	-	-	-	8	

Продолжение табл. 2.27

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
RES 0,D	CB 82	-	-	-	-	8	
RES 0,E	CB 83	-	-	-	-	8	
RES 0,(HL)	CB 86	-	-	-	-	15	
RES 0,(IX+S)	DD CB S 86	-	-	-	-	23	
RES 0,(IY+S)	FD CB S 86	-	-	-	-	23	
RES 1,A	CB 8F	-	-	-	-	8	
RES 1,H	CB 8C	-	-	-	-	8	
RES 1,L	CB 8D	-	-	-	-	8	
RES 1,B	CB 88	-	-	-	-	8	
RES 1,C	CB 89	-	-	-	-	8	
RES 1,D	CB 8A	-	-	-	-	8	
RES 1,E	CB 8B	-	-	-	-	8	
RES 1,(HL)	CB 8E	-	-	-	-	15	
RES 1,(IX+S)	DD CB S 8E	-	-	-	-	23	
RES 1,(IY+S)	FD CB S 8E	-	-	-	-	23	
RES 2,A	CB 97	-	-	-	-	8	
RES 2,H	CB 94	-	-	-	-	8	
RES 2,L	CB 95	-	-	-	-	8	
RES 2,B	CB 90	-	-	-	-	8	
RES 2,C	CB 91	-	-	-	-	8	
RES 2,D	CB 92	-	-	-	-	8	
RES 2,E	CB 93	-	-	-	-	8	
RES 2,(HL)	CB 96	-	-	-	-	15	
RES 2,(IX+S)	DD CB S 96	-	-	-	-	23	
RES 2,(IY+S)	FD CB S 96	-	-	-	-	23	
RES 3,A	CB 9F	-	-	-	-	8	
RES 3,H	CB 9C	-	-	-	-	8	
RES 3,L	CB 9D	-	-	-	-	8	
RES 3,B	CB 98	-	-	-	-	8	
RES 3,C	CB 99	-	-	-	-	8	
RES 3,D	CB 9A	-	-	-	-	8	
RES 3,E	CB 9B	-	-	-	-	8	
RES 3,(HL)	CB 9E	-	-	-	-	15	
RES 3,(IX+S)	DD CB S 9E	-	-	-	-	23	
RES 3,(IY+S)	FD CB S 9E	-	-	-	-	23	
RES 4,A	CB A7	-	-	-	-	8	
RES 4,H	CB A4	-	-	-	-	8	
RES 4,L	CB A5	-	-	-	-	8	
RES 4,B	CB A0	-	-	-	-	8	
RES 4,C	CB A1	-	-	-	-	8	
RES 4,D	CB A2	-	-	-	-	8	

Продолжение табл. 2.27

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
RES 4,E	CB A3	-	-	-	-	8	
RES 4,(HL)	CB A6	-	-	-	-	15	
RES 4,(IX+S)	DD CB S A6	-	-	-	-	23	
RES 4,(IY+S)	FD CB S A6	-	-	-	-	23	
RES 5,A	CB AF	-	-	-	-	8	
RES 5,H	CB AC	-	-	-	-	8	
RES 5,L	CB AD	-	-	-	-	8	
RES 5,B	CB A8	-	-	-	-	8	
RES 5,C	CB A9	-	-	-	-	8	
RES 5,D	CB AA	-	-	-	-	8	
RES 5,E	CB AB	-	-	-	-	8	
RES 5,(HL)	CB AE	-	-	-	-	15	
RES 5,(IX+S)	DD CB S AE	-	-	-	-	23	
RES 5,(IY+S)	FD CB S AE	-	-	-	-	23	
RES 6,A	CB B7	-	-	-	-	8	
RES 6,H	CB B4	-	-	-	-	8	
RES 6,L	CB B5	-	-	-	-	8	
RES 6,B	CB B0	-	-	-	-	8	
RES 6,C	CB B1	-	-	-	-	8	
RES 6,D	CB B2	-	-	-	-	8	
RES 6,E	CB B3	-	-	-	-	8	
RES 6,(HL)	CB B6	-	-	-	-	15	
RES 6,(IX+S)	DD CB S B6	-	-	-	-	23	
RES 6,(IY+S)	FD CB S B6	-	-	-	-	23	
RES 7,A	CB BF	-	-	-	-	8	
RES 7,H	CB BC	-	-	-	-	8	
RES 7,L	CB BD	-	-	-	-	8	
RES 7,B	CB B8	-	-	-	-	8	
RES 7,C	CB B9	-	-	-	-	8	
RES 7,D	CB BA	-	-	-	-	8	
RES 7,E	CB BB	-	-	-	-	8	
RES 7,(HL)	CB BE	-	-	-	-	15	
RES 7,(IX+S)	DD CB S BE	-	-	-	-	23	
RES 7,(IY+S)	FD CB S BE	-	-	-	-	23	

Команды проверки битов. Таблица 2.28.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
BIT 0,A	CB 47	-	*	?	?	8	
BIT 0,H	CB 44	-	*	?	?	8	
BIT 0,L	CB 45	-	*	?	?	8	
BIT 0,B	CB 40	-	*	?	?	8	
BIT 0,C	CB 41	-	*	?	?	8	
BIT 0,D	CB 42	-	*	?	?	8	
BIT 0,E	CB 43	-	*	?	?	8	
BIT 0,(HL)	CB 46	-	*	?	?	12	
BIT 0,(IX+S)	DD CB S 46	-	*	?	?	20	
BIT 0,(IY+S)	FD CB S 46	-	*	?	?	23	
BIT 1,A	CB 4F	-	*	?	?	8	
BIT 1,H	CB 4C	-	*	?	?	8	
BIT 1,L	CB 4D	-	*	?	?	8	
BIT 1,B	CB 48	-	*	?	?	8	
BIT 1,C	CB 49	-	*	?	?	8	
BIT 1,D	CB 4A	-	*	?	?	8	
BIT 1,E	CB 4B	-	*	?	?	8	
BIT 1,(HL)	CB 4E	-	*	?	?	12	
BIT 1,(IX+S)	DD CB S 4E	-	*	?	?	20	
BIT 1,(IY+S)	FD CB S 4E	-	*	?	?	23	
BIT 2,A	CB 57	-	*	?	?	8	
BIT 2,H	CB 54	-	*	?	?	8	
BIT 2,L	CB 55	-	*	?	?	8	
BIT 2,B	CB 50	-	*	?	?	8	
BIT 2,C	CB 51	-	*	?	?	8	
BIT 2,D	CB 52	-	*	?	?	8	
BIT 2,E	CB 53	-	*	?	?	8	
BIT 2,(HL)	CB 56	-	*	?	?	12	
BIT 2,(IX+S)	DD CB S 56	-	*	?	?	20	
BIT 2,(IY+S)	FD CB S 56	-	*	?	?	23	
BIT 3,A	CB 5F	-	*	?	?	8	
BIT 3,H	CB 5C	-	*	?	?	8	
BIT 3,L	CB 5D	-	*	?	?	8	
BIT 3,B	CB 58	-	*	?	?	8	
BIT 3,C	CB 59	-	*	?	?	8	
BIT 3,D	CB 5A	-	*	?	?	8	
BIT 3,E	CB 5B	-	*	?	?	8	
BIT 3,(HL)	CB 5E	-	*	?	?	12	
BIT 3,(IX+S)	DD CB S 5E	-	*	?	?	20	
BIT 3,(IY+S)	FD CB S 5E	-	*	?	?	23	
BIT 4,A	CB 67	-	*	?	?	8	

Продолжение таблицы 2.28.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
BIT 4,H	CB 64	-	*	?	?	8	
BIT 4,L	CB 65	-	*	?	?	8	
BIT 4,B	CB 60	-	*	?	?	8	
BIT 4,C	CB 61	-	*	?	?	8	
BIT 4,D	CB 62	-	*	?	?	8	
BIT 4,E	CB 63	-	*	?	?	8	
BIT 4,(HL)	CB 66	-	*	?	?	12	
BIT 4,(IX+S)	DD CB S 66	-	*	?	?	20	
BIT 4,(IY+S)	FD CB S 66	-	*	?	?	23	
BIT 5,A	CB 6F	-	*	?	?	8	
BIT 5,H	CB 6C	-	*	?	?	8	
BIT 5,L	CB 6D	-	*	?	?	8	
BIT 5,B	CB 68	-	*	?	?	8	
BIT 5,C	CB 69	-	*	?	?	8	
BIT 5,D	CB 6A	-	*	?	?	8	
BIT 5,E	CB 6B	-	*	?	?	8	
BIT 5,(HL)	CB 6E	-	*	?	?	12	
BIT 5,(IX+S)	DD CB S 6E	-	*	?	?	20	
BIT 5,(IY+S)	FD CB S 6E	-	*	?	?	23	
BIT 6,A	CB 77	-	*	?	?	8	
BIT 6,H	CB 74	-	*	?	?	8	
BIT 6,L	CB 75	-	*	?	?	8	
BIT 6,B	CB 70	-	*	?	?	8	
BIT 6,C	CB 71	-	*	?	?	8	
BIT 6,D	CB 72	-	*	?	?	8	
BIT 6,E	CB 73	-	*	?	?	8	
BIT 6,(HL)	CB 76	-	*	?	?	12	
BIT 6,(IX+S)	DD CB S 76	-	*	?	?	20	
BIT 6,(IY+S)	FD CB S 76	-	*	?	?	23	
BIT 7,A	CB 7F	-	*	?	?	8	
BIT 7,H	CB 7C	-	*	?	?	8	
BIT 7,L	CB 7D	-	*	?	?	8	
BIT 7,B	CB 78	-	*	?	?	8	
BIT 7,C	CB 79	-	*	?	?	8	
BIT 7,D	CB 7A	-	*	?	?	8	
BIT 7,E	CB 7B	-	*	?	?	8	
BIT 7,(HL)	CB 7E	-	*	?	?	12	
BIT 7,(IX+S)	DD CB S 7E	-	*	?	?	20	
BIT 7,(IY+S)	FD CB S 7E	-	*	?	?	23	

Команды перемещения блоков. Таблица 2.29

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
LDIR	ED B0	-	-	O	-	21	BC<>0
						16	BC=0
LDDR	ED B8	-	-	O	-	21	BC<>0
						16	BC=0
LDI	ED A0	-	-	*	-	16	P/V=0,если BC=0 иначе 1
LDD	ED A8	-	-	*	-	16	P/V=0,если BC=0 иначе 1

Команды блочного поиска. Таблица 2.30.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
CPIR	ED B1	-	*	*	*	21	если BC<>0 и A<>(HL)
						16	если BC=0 или A=(HL)
							P/V=0,если BC=0 иначе 1
							Z=1,если A=(HL) иначе 0
CPDR	ED B9	-	*	*	*	21	если BC<>0 и A<>(HL)
						16	если BC=0 или A=(HL)
							P/V=0,если BC=0 иначе 1
							Z=1,если A=(HL) иначе 0
CPI	ED A1	-	*	*	*	16	P/V=0,если BC=0 иначе 1
							Z=1,если A=(HL) иначе 0
CPD	ED A9	-	*	*	*	16	P/V=0,если BC=0 иначе 1
							Z=1,если A=(HL) иначе 0

Команды ввода от внешних устройств. Таблица 2.31.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
IN A,(N)	DB N	-	-	-	-	11	N - A0...A7 рег.A -A8...A15
IN A,(C)	ED 78	-	*	P	*	12	C - A0...A7 B - A7...A15
IN H,(C)	ED 60	-	*	P	*	12	---"---"---"---
IN L,(C)	ED 68	-	*	P	*	12	---"---"---"---
IN B,(C)	ED 40	-	*	P	*	12	---"---"---"---
IN C,(C)	ED 48	-	*	P	*	12	---"---"---"---
IN D,(C)	ED 50	-	*	P	*	12	---"---"---"---
IN E,(C)	ED 58	-	*	P	*	12	---"---"---"---
IN F,(C)	ED 70	*	*	*	*	12	---"---"---"---
INIR	ED B2	?	1	?	?	21	B <> 0
						16	B = 0
							C - A0...A7 B - A7...A15
INDR	ED BA	?	1	?	?	21	B <> 0
						16	B = 0
							C - A0...A7 B - A7...A15
INI	ED A2	?	*	?	?	16	C - A0...A7 B - A7...A15 Z=1 если B=1, иначе 0
IND	ED AA	?	*	?	?	16	C - A0...A7 B - A7...A15 Z=1 если B=1, иначе 0

Команды вывода на внешние устройства. Таблица 2.32

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
OUT (N),A	D3 N	-	-	-	-	11	N - A0...A7 рег.A -A8...A15
OUT (C),A	ED 79	-	*	P	*	12	B - A7...A15

Продолжение таблицы 2.32.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
OUT (C),H	ED 61	-	*	P	*	12	C - A0...A7 B - A7...A15
OUT (C).L	ED 69	-	*	P	*	12	---"---"---"---
OUT (C),B	ED 41	-	*	P	*	12	---"---"---"---
OUT (C),C	ED 49	-	*	P	*	12	---"---"---"---
OUT (C),D	ED 51	-	*	P	*	12	---"---"---"---
OUT (C),E	ED 59	-	*	P	*	12	---"---"---"---
OTIR	ED B3	?	1	?	?	21	B <> 0
						16	B = 0 C - A0...A7 B - A7...A15
OTDR	ED BB	?	1	?	?	21	B <> 0
						16	B = 0 C - A0...A7 B - A7...A15
OUTI	ED A3	?	*	?	?	16	C - A0...A7 B - A7...A15 Z=1 если B=1, иначе 0
OUTD	ED AB	?	*	?	?	16	C - A0...A7 B - A7...A15 Z=1 если B=1, иначе 0

Команды обработки прерываний. Таблица 2.33

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
EI	FB	-	-	-	-	4	
DI	F3	-	-	-	-	4	
IM 0	ED 46	-	-	-	-	8	
IM 1	ED 56	-	-	-	-	8	
IM 2	ED 5E	-	-	-	-	8	
RET I	ED 4D	-	-	-	-	14	
RET N	ED 45	-	-	-	-	14	

Прочие команды. Таблица 2.34.

Мнемоника	Код	Влияние на флаги				Время	Примечание
		C	Z	P/V	S	(тактов)	
NOP	00	-	-	-	-	4	
CPL	2F	-	-	-	-	4	
NEG	ED 44	*	*	V	*	8	
SCF	37	1	-	-	-	4	
CCF	3F	*	-	-	-	4	
HALT	76	1	-	-	-	4	
DAA	27	*	*	P	*	4	Влияет также на флаг H (полупереноса).

3. СИСТЕМА КОМАНД КАЛЬКУЛЯТОРА

Таблица 3.

Код	Команда	Содержимое вершины стека		Нарушение регистров	Назначение команды
		до	после		
00 s	jump true	x	-	-	Выполняется относительный переход на s шагов, если x имеет значение TRUE.
01	exchange	x,y	y,x	-	Меняются местами два числа на вершине стека
02	delete	x	-	-	Верхнее число удаляется со стека
03	subtract	x,y	x-y	-	Вычитание
04	multiply	x,y	x*y	-	Умножение
05	divide	x,y	x/y	-	Деление
06	power	x,y	x^y	B,M0...M3	Возведение в степень
07	or	x,y	TRUE или FALSE	-	Логическое "ИЛИ"
08	n and	x,y	TRUE/ FALSE	-	Логическое "И" для чисел

Продолжение таблицы 3.

Код	Команда	Содержимое вершины стека		Нарушение регистров	Назначение команды
		до	после		
07	or	x,y	TRUE или FALSE	-	Логическое "ИЛИ"
08	n and	x,y	TRUE/ FALSE	-	Логическое "И" для чисел
09	n le	x,y	TRUE/ FALSE	-	"МЕНЬШЕ ИЛИ РАВНО" В рег-ре B д.б.09
0A	n ge	x,y	TRUE/ FALSE	-	"БОЛЬШЕ ИЛИ РАВНО" В рег-ре B д.б.0A
0B	n ne	x,y	TRUE/ FALSE	-	"НЕ РАВНО" В рег-ре B д.б.0B
0C	n gt	x,y	TRUE/ FALSE	-	"БОЛЬШЕ" В рег-ре B д.б.0C
0D	n lt	x,y	TRUE/ FALSE	-	"МЕНЬШЕ" В рег-ре B д.б.0D
0E	n eq	x,y	TRUE/ FALSE	-	"РАВНО" В рег-ре B д.б.0E
0F	add	x,y	x+y	-	Сложение (для чисел)
10	s and	x$,y	TRUE/ FALSE	-	Логическое "И" для стринга и числа
11	s le	x$,y$	TRUE/ FALSE	-	"МЕНЬШЕ ИЛИ РАВНО" В регистре B д.б.11
12	s ge	x$,y$	TRUE/ FALSE	-	"БОЛЬШЕ ИЛИ РАВНО" В регистре B д.б.12
13	s ne	x$,y$	TRUE/ FALSE	-	"НЕ РАВНО" В регистре B д.б.13
14	s gt	x$,y$	TRUE/ FALSE	-	"БОЛЬШЕ" В рег-ре B д.б.14
15	s lt	x$,y$	TRUE/ FALSE	-	"МЕНЬШЕ" В рег-ре B д.б.15
16	s eq	x$,y$	TRUE/ FALSE	-	"РАВНО" В рег-ре B д.б.16
17	s add	x$,y$	x$+y$	-	Слияние стрингов
18	val$	x$	val$ x$	-	Расчет x$ без ограничительных кавычек как стрингового выражения. в рег-ре B д.б.18
19	usr s	x$	usr x$	-	Определение адреса

Продолжение таблицы 3.

Код	Команда	Содержимое вершины стека		Нарушение регистров	Назначение команды
		до	после		
					символа графики пользователя, установленного для клавиши с символом x$.
1A	read in	x	inkey$#x	-	Определение символа, поступающего от канала подключенного к потоку x.
1B	negate	x	-x	-	Изменение знака на противоположный
1C	code	x$	code x$	-	Определение кода символа.
1D	val	x$	val x$	-	Расчет x$ без ограничительных кавычек как числового выражения. В регистре B д.б.1D
1E	len	x$	len x$	-	Определение длины стринга.
1F	sin	x	sin x	B,M0...M2	Вычисление синуса
20	cos	x	cos x	B,M0...M2	Вычисление косинуса
21	tan	x	tan x	B,M0...M2	Вычисление тангенса
22	asn	x	asn x	B,M0...M2	Вычисление тангенса
23	acs	x	acs x	B,M0...M2	Вычисление арккосинуса
24	atn	x	atn x	B,M0...M2	Вычисление арктангенса
25	ln	x	ln x	B,M0...M2	Вычисление натурального логарифма
26	exp	x	exp x	B,M0...M3	Вычисление e в степени x. Здесь e - основание натуральных логарифмов
27	int	x	int(x)	M0	Преобразование действительного числа в целое. Округление выполняется вниз.
28	sqr	x	sqr(x)	B,M0...M3	Вычисление корня квадратного

Продолжение таблицы 3.

Код	Команда	Содержимое вершины стека		Нарушение регистров	Назначение команды
		до	после		
8 n	series n	x	Pn(x)	B,M0...N2	Вычисление многочлена Чебышева n-го порядка (Применяется при расчетах алгебраических и тригонометрических функций)
9 n	series n+10	x	P (x) n-10	B,M0...M2	То же
A0	const zero	-	0	-	Помещает на стек 0
A1	const one	-	1	-	Помещает на стек 1
A2	const half	-	1/2	-	Помещает на стек 0.5
A3	const pi/2	-	1.57079	-	Помещает на стек одну вторую числа Пи
A4	const ten	-	10	-	Помещает на стек 10
Cn	store Mn	x	x	-	Копирование числа с вершины стека в n-ую ячейку памяти калькулятора. Нормально n=0...5, но можно установить n от 6 до 1F путем изменения содержимого системной переменной MEM.
Dn	Store M(n+10)	x	x	-	То же, но n=10...1F
En	recall Mn	-	Mn	-	Вызов на стек данных из ячеек памяти
Fn	recall M(n+10)	-	M(n+10)	-	То же
29	sgn	x	sgn(x)	-	Определение знака
2A	abs	x	abs(x)	-	Определение абсолютной величины числа
2B	peek	x	peek(x)	-	Определение содержимого памяти по адресу
2C	in	x	in x	-	Ввод числа с внешнего порта x
2D	usr n	x	usr x	?	Вычисление процедуры, записанной в машинных кодах

Продолжение таблицы 3.

Код	Команда	Содержимое вершины стека		Нарушение регистров	Назначение команды
		до	после		
2E	str$	x	str$ x	M0...M5	Преобразование числа в соответствующий стринг. В системной переменной MEM должен храниться адрес MEMBOT
2F	chr$	x	chr$ x	-	Определение символа числа по его коду
30	not	x	T или F	-	Условие "НЕ"
31	duplicate	x	x,x	-	Повторение числа на вершине стека
32	mod,div	x,y	x-y* int(x/y), int(x/y)	M0	Выделение остатка от деления двух чисел и целого частного
33 s	jump	-	-	-	Безусловный переход
34	stk data	-	x	-	Помещает на стек число x; оно должно быть записано в упакованной форме.
35 s	djnz	-	-	-	Если в регистре B не 0, то выполняется переход на s байтов, а содержимое регистра B уменьшается на 1.
36	lt zt	x	T или F	-	Условие "МЕНЬШЕ НУЛЯ"
37	gt z	x	T или F	-	Условие "БОЛЬШЕ НУЛЯ"
38	end calc	-	-	-	Выход из калькулятора
39	get argt	x	2*asn(sin	M0	Вычисление формулы
3A	truncate	x	sgn(x)* int(abs x)	-	Выделение целой части числа (округление "к нулю")
3B	execute B	-	-	-	Выполнение той команды калькулятора, код которой содержится в регистре B.
3C	e to fp	x	?	-	Не работает по причине ошибки в ПЗУ
3D	restack	x	x	-	Перевод целого числа в действительное

4. СЛОВАРЬ МНЕМОНИК АССЕМБЛЕРА

Ниже мы даем в алфавитном порядке все мнемоники машинного кода процессора Z-80, их полную форму на английском языке, русский эквивалент и необходимые пояснения. Практика показывает, что эта информация позволяет ускорить освоение системы команд процессора начинающими.

ADC - ADDITION with CARRY = СЛОЖЕНИЕ с учетом ПЕРЕНОСА. (Операция сложения с учетом состояния флага переноса - флага C регистра F).

ADD - ADDITION = СЛОЖЕНИЕ.

AND = логическое "И".

BIT - test BIT = Проверить БИТ. (Выполнется проверка включен или нет данный бит в регистре или в ячейке памяти).

CALL = ВЫЗОВ. (Вызывается исполнение подпрограммы).

CCF - COMPLEMENT CARRY FLAG = ДОПОЛНИТЬ (до единицы) ФЛАГ ПЕРЕНОСА. (Эта команда вызывает изменение состояние флага переноса на противоположное).

CP - COMPARE = СРАВНИТЬ.

CPD - COMPARE and DECREMENT = СРАВНИТЬ и УМЕНЬШИТЬ на единицу. (Сравнивается содержимое аккумулятора с содержимым ячейки памяти, адрес которой находится в процессоре. Если они не равны, то адрес и счетчик уменьшаются на единицу).

CPDR - COMPARE, DECREMENT and REPEAT = СРАВНИТЬ, УМЕНЬШИТЬ на единицу и ПОВТОРИТЬ. (Сравнивается содержимое аккумулятора с содержимым заданной ячейки памяти. Если они не равны, то адрес и счетчик уменьшаются на единицу и процесс повторяется).

CPI - COMPARE and INCREMENT = СРАВНИТЬ и УВЕЛИЧИТЬ на единицу. (Сравнивается содержимое аккумулятора с содержимым заданной ячейки памяти. Если они не равны, то адрес увеличивается на единицу, а счетчик уменьшается на единицу).

CPIR - COMPARE, INCREMENT and REPEAT = СРАВНИТЬ, УВЕЛИЧИТЬ на ЕДИНИЦУ и ПОВТОРИТЬ. Сравнивается содержимое аккумулятора с содержимым ячейки памяти, адрес которой находится в процессоре. Если они не равны, то адрес увеличивается на единицу, а счетчик уменьшается на единицу и процесс повторяется).

CPL - COMPLEMENT = ДОПОЛНИТЬ аккумулятор. (Каждый бит аккумулятора переключается на противоположный).

DAA - DECIMAL ADJUST ACCUMULATOR = ДЕСЯТИРИЧНАЯ НАСТРОЙКА АККУМУЛЯТОРА. (Содержимое аккумулятора перестраивается в соответствии с правилами BCD-арифметики).

DEC - DECREMENT = УМЕНЬШЕНИЕ (на единицу).

DI - DISABLE INTERRUPTS = ОТКЛЮЧЕНИЕ ПРЕРЫВАНИЙ.

DJNZ - DECREMENT and JUMP if NOT ZERO = УМЕНЬШИТЬ (на единицу) и ПЕРЕЙТИ, если НЕ НОЛЬ. (Уменьшается на единицу содержимое регистра BC и, если ноль еще не достигнут, выполняется от-

носительный переход на заданную величину байтов).

EI - ENABLE INTERRUPTS = РАЗРЕШЕНИЕ ПРЕРЫВАНИЙ.

EX - EXCHANGE = ОБМЕН. (Обмен содержимого регистров).

HALT = СТОП

IM 0 - INTERRUPT MODE 0 = РЕЖИМ ПРЕРЫВАНИЙ 0.

IM 1 - INTERRUPT MODE 1 = РЕЖИМ ПРЕРЫВАНИЙ 1.

IM 2 - INTERRUPT MODE 2 = РЕЖИМ ПРЕРЫВАНИЙ 2.

IN - INPUT = ВВОД (данных с внешнего порта).

INC - INCREMENT = УВЕЛИЧИТЬ (на единицу).

IND - INPUT and DECREMENT = ВВОД и УМЕНЬШЕНИЕ. (После ввода байта с внешнего порта происходит уменьшение указателя адреса на единицу).

INDR - INPUT, DECREMENT and REPEAT = ВВОД, УМЕНЬШЕНИЕ и ПОВТОР (После ввода байта с внешнего порта происходит уменьшение указателя адреса на единицу, и далее процесс повторяется до тех пор, пока не обнулится счетчик байтов).

INI - INPUT and INCREMENT = ВВОД и УВЕЛИЧЕНИЕ. (После ввода байта с внешнего порта происходит увеличение указателя адреса на единицу).

INIR - INPUT,INCREMENT and REPEAT = ВВОД, УВЕЛИЧЕНИЕ и ПОВТОР. (После ввода байта с внешнего порта происходит увеличение указателя адреса на единицу и далее процесс повторяется до тех пор, пока не обнулится счетчик байтов).

JP - JUMP = ПЕРЕХОД (абсолютный).

JR - JUMP RELATIVE = ПЕРЕХОД ОТНОСИТЕЛЬНЫЙ.

LD - LOAD = ЗАГРУЗКА. (Выполняется копирование данных).

LDD - LOAD and DECREMENT - ЗАГРУЗИТЬ и УМЕНЬШИТЬ. (Производится копирование данных из одной области памяти в другую, после чего происходит уменьшение указателей адресов источника и места назначения).

LDDR - LOAD, DECREMENT and REPEAT - ЗАГРУЗИТЬ, УМЕНЬШИТЬ и ПОВТОРИТЬ. (Производится копирование данных из одной области памяти в другую, после чего происходит уменьшение указателей адресов источника и места назначения, а также счетчика байтов. Если счетчик не равен нулю, то процесс повторяется).

LDI - LOAD and INCREMENT = ЗАГРУЗИТЬ И УВЕЛИЧИТЬ. (Производится копирование данных из одной области паямти в другую, после чего происходит увеличение указателей адресов источника и места назначения).

LDIR - LOAD, INCREMENT and REPEAT = ЗАГРУЗИТЬ, УВЕЛИЧИТЬ и ПОВТОРИТЬ. (Производится копирование данных из одной области памяти в другую, после чего происходит увеличение указателей адресов источника и места назначения, а также уменьшение счетчика байтов. Если он не равен нулю, то процесс повторяется).

NEG - NEGATE = ИЗМЕНИТЬ ЗНАК (аккумулятора).

NOP - NO OPERATION = НЕТ ОПЕРАЦИИ

OR - логическое "ИЛИ"

OUT - OUTPUT = ВЫВОД (данных на внешнее устройство).

OUTD - OUTPUT and DECREMENT = ВЫВОД и УМЕНЬШЕНИЕ. (После вывода байта уменьшается на единицу указатель адреса).

OUTDR - OUTPUT, DECREMENT and REPEAT = ВЫВОД, УМЕНЬШЕНИЕ и ПОВТОР. (После вывода из памяти на внешнее устройство уменьшаются на единицу указатель адреса и содержимое счетчика байтов. Если счетчик не достиг нуля, операция повторяется).

OUTI - OUTPUT and INCREMENT = ВЫВОД и УВЕЛИЧЕНИЕ. (После вывода байта увеличивается на единицу указатель адреса).

OTIR - OUTPUT, INCREMENT and REPEAT = ВЫВОД, УВЕЛИЧЕНИЕ и ПОВТОР. (После вывода байта увеличивается на единицу указатель адреса и уменьшается содержимое счетчика байтов. Если счетчик не достиг нуля, операция повторяется).

POP = ВЫТОЛКНУТЬ. (Перенос данных с вершины стека в заданную регистровую пару).

PUSH = ЗАТОЛКНУТЬ. (Копирование содержимого регистровой пары на вершину машинного стека).

RET - RETURN = ВОЗВРАТ.

RETI - RETURN from INTERRUPT = ВОЗВРАТ после обработки ПРЕРЫВАНИЯ.

RETN - RETURN from NON-MASCABLE INTERRUPT = ВОЗВРАТ после обработки НЕМАСКИРОВАННОГО ПРЕРЫВАНИЯ).

RL - ROTATE LEFT = ВРАЩАТЬ ВЛЕВО (Биты в регистре или в ячейке памяти).

RLA - ROTATE LEFT ACCUMULATOR = ВРАЩАТЬ ВЛЕВО биты в АККУМУЛЯТОРЕ.

RLC - ROTATE LEFT without CARRY - ВРАЩАТЬ ВЛЕВО без флага ПЕРЕНОСА.(Флаг переноса хоть и изменяется с учетом результата операции, но во вращении не участвует).

RLCA - ROTATE ACCUMULATOR LEFT without CARRY - ВРАЩАТЬ АККУМУЛЯТОР ВЛЕВО без флага ПЕРЕНОСА. (Флаг переноса хоть и изменяется с учетом результата операции, но во вращении не участвует).

RLD - DECIMAL ROTATE LEFT = ДЕСЯТИРИЧНОЕ ВРАЩЕНИЕ ВЛЕВО. (Вращение влево полубайтов, содержащих десятиричные разряды в BCD-арифметике).

RR - ROTATE RIGHT = ВРАЩАТЬ ВПРАВО (биты в регистре или в ячейке памяти).

RRA - ROTATE RIGHT ACCUMULATOR = ВРАЩАТЬ ВПРАВО БИТЫ в АККУМУЛЯТОРЕ).

RRC - ROTATE RIGHT without CARRY - ВРАЩАТЬ ВПРАВО без флага ПЕРЕНОСА. (Флаг переноса хоть и изменяется с учетом результата операции, но во вращении не участвует).

RRCA - ROTATE ACCUMULATOR RIGHT without CARRY - ВРАЩАТЬ АККУМУЛЯТОР ВПРАВО без флага ПЕРЕНОСА. (Флаг переноса хоть и изменяется с учетом результата операции, но во вращении не участвует).

RRD - DECIMAL ROTATE RIGHT = ДЕСЯТИРИЧНОЕ ВРАЩЕНИЕ ВПРАВО. (Вращение вправо полубайтов в BCD-арифметике).

RST - RESTART = ПОВТОРНЫЙ ЗАПУСК. (Вызов на исполнение заданной программы из системного ПЗУ компьютера).

SBC - SUBTRACT with CARRY = ВЫЧИТАНИЕ с учетом флага ПЕРЕНОСА.

SET = ВКЛЮЧИТЬ (указанный бит в указанном регистре или ячейке памяти).

SLA - SHIFT LEFT ARITHMETIC = АРИФМЕТИЧЕСКИЙ СДВИГ ВЛЕВО.

SRA - SHIFT RIGHT ARITHMETIC = АРИФМЕТИЧЕСКИЙ СДВИГ ВПРАВО

SRL - SHIFT RIGHT LOGICAL = ЛОГИЧЕСКИЙ СДВИГ ВПРАВО.

5. ТАБЛИЦА УКАЗАТЕЛЕЙ ПРЕРЫВАНИЙ ИЗ ПЗУ

Значение регистра "I"		Адрес перехода		Точка ПЗУ
DEC	HEX	DEC	HEX	DEC
0	00	20430	4FCE	255
1	01	52818	CE52	511
2	02	22269	56FD	767
3	03	39020	986C	1023
4	04	10419	28B3	1279
5	05	2294	08F6	1535
6	06	29149	71DD	1791
7	07	16039	3EA7	2047
8	08	2088	0828	2303
9	09	65129	FE69	2559
10	0A	32802	8022	2815
11	0B	58888	E608	3071
12	0C	53183	CFBF	3327
13	0D	52503	CD17	3583
14	0E	14367	381F	3839
15	0F	27928	6D18	4095
16	10	51984	CB10	4351
17	11	8729	2219	4607
18	12	52481	CD01	4863
19	13	49749	C255	5119
20	14	25705	6469	5375
21	15	51673	C9D9	5631
22	16	51568	C970	5888
23	17	12493	30CD	6143
24	18	15582	3CDE	6399
25	19	23842	5D22	6655
26	1A	13824	3600	6911
27	1B	7306	1C8A	7167
28	1C	49947	C31B	7423
29	1D	2344	0928	7679

Продолжение таблицы указателей.

30	1E	26573	67CD	7935
31	1F	3360	0D20	8191
32	20	52513	CD21	8447
33	21	33485	82CD	8703
34	22	544	0220	8959
35	23	49537	C181	9215
36	24	8527	214F	9471
37	25	23670	5C76	9727
38	26	20444	4FDC	9983
39	27	288	0120	10239
40	28	32348	7E5C	10495
41	29	58154	E32A	10751
42	2A	19754	4D2A	11007
43	2B	23653	5C65	11263
44	2C	7117	1BCD	11519
45	2D	55781	D9E5	11775
46	2E	23713	5CA1	12031
47	2F	4569	11D9	12287
48	30	60208	EB30	12543
49	31	57640	E128	12799
50	32	13627	353B	13055
51	33	13256	33C8	13331
52	34	1560	0618	13567
53	35	57124	DF24	13823
54	36	34307	8603	14079
55	37	41231	A10F	14335
56	38	65535	FFFF	14591
57	39	65535	FFFF	14847
58	3A	65535	FFFF	15103
59	3B	65535	FFFF	15359
60	3C	255	00FF	15615
61	3D	0	00	15871
62	3E	255	00FF	16127
63	3F	60	003C	16383

6. СПРАВОЧНЫЕ ТАБЛИЦЫ ПО РУСИФИКАЦИИ КОМПЬЮТЕРА.

Стандартный код ASCII определяет только символы с номерами от 32 (пробел) до 127 ("копирайт"). В этом диапазоне расположены знаки препинания, цифры, а также прописные и строчные буквы латинского алфавита. Символы от 0 до 31 являются управляющими кодами и стандартизированы лишь частично. Разработчики компьютеров могут некоторые из кодов этого диапазона использовать по-разному.

Символы выше 127-го этот стандарт не определяет и оставляет этот диапазон для размещения в нем национальных шрифтов (немецкого, французского, шведского и др.), а также для размещения там символов блочной графики.

В компьютерах типа "ZX-Spectrum" его разработчики разместили в верхней части кодовой таблицы символы блочной графики (128 - 143), символы графики пользователя (144 - 164) и токены ключевых слов встроенного БЕЙСИКа (165 - 255).

Наличие в этом диапазоне токенов ключевых слов делает невозможным размещение в верхней части кодовой таблицы символов русского алфавита и приходится искать другие пути.

Наши читатели, очевидно, знают, что русификацию компьютера проводят двумя наиболее распространенными методами. Во-первых, используя символы графики пользователя UDG, а во-вторых, сменой знакогенератора и переключением системной переменной CHARS.

6.1. Русификация с использованием символов UDG.

Символы графики пользователя могут быть использованы для задания собственных символов. К сожалению, их всего только 21 и поэтому русский шрифт создать удается только для прописных букв и то при условии использования букв английского алфавита, имеющих с русскими одинаковое начертание.

Стандартов размещения прописных русских букв на клавишах компьютера в графическом режиме нет и быть не может, поскольку сам метод использования символов UDG является специфически "синклеровским" и нестандартен. Одним словом, Вы можете размещать свои буквы на клавишах графики пользователя, как хотите.

Тем не менее, мы все же рекомендовали бы обратить внимание на следующий факт. Где-то году примерно в 1984-1985, когда компьютеры системы "Синклер" только начали появляться в СССР, появилась и несложная программа, русифицирующая прописные буквы с помощью символов UDG. Это программа "ABC" неизвестного автора. Она получила очень широкое распространение и, возможно, стоит придерживаться принятой в ней системы. Неважно, хороша она или плоха, но она нашла широкое распространение и это, пожалуй главное. Мы приводим заданные в ней соотношения:

Клавиша в граф.режиме	Буква	Клавиша в граф.режиме	Буква	Клавиша в граф.режиме	Буква
Q	Ы	D	Д	P	П
E	Э	F	Ф	A	Я
R	Ь	G	Г	S	Й
T	Ъ	H	Ч	B	Б
U	Ю	J	Ж	N	Ш
I	И	L	Л	M	Щ
O	У	C	Ц		

Прочие прописные буквы имеют одинаковые написания и берутся не в графическом, а в обычном режиме - курсор "L".

6.2 Русификация заменой генератора.

С использованием графики пользователя русский текст набирать очень утомительно. Этот метод следует использовать, если текст имеет достаточно малый удельный вес в Вашей программе - до нескольких десятков слов. В том же случае, если работа с большими массивами текста - Ваша специальность, то следует провести замену знакогенератора. О том, как это сделать, мы неоднократно писали в своих ранних работах и это знают, по-видимому, почти все.

Перед Вами стоит выбор - "какого стандарта придерживаться?". Здесь возможны два варианта. Во-первых, работая с большими объемами текста, Вы по-видимому, планируете выводить его на принтер. Тогда можно провести русификацию так, чтобы принтер правильно понимал коды русских букв. Во-вторых, если Вы имеете богатый опыт работы с пишущей машинкой, то можно русификацию провести под стандартную клавиатуру русской пишущей машинки, а привязку к принтеру осуществить уже программным путем. Тогда код, перед тем, как отправляться на печать, должен конвертироваться в стандарт, поддерживаемый принтером.

6.2.1. Русификация "под принтер".

Мы уже установили, что на "Спектруме" Вы можете изменять только нижнюю половину кодовой таблицы. Разместить там и английский и русский шрифты одновременно невозможно, поэтому их надо переключать, изменяя содержимое системной переменной CHARS по мере необходимости. Точно так же подачей управляющего кода на принтер его можно переключать то на латинский, то на русский шрифт (см. инструкцию к принтеру). Таким образом, если воспользоваться принтером, у которого русский шрифт тоже размещен в нижней половине кодовой таблицы (а именно так и есть у большинства наиболее распространенных у нас в стране дешевых принтеров ROBOTRON, D-100 и т.п.), то коды должны соответствовать стандарту КОИ-7 (см. табл. 6.2 - символы до 63-го в ней не показаны, т.к. знаки препинания и цифры совпадают).

Если Ваш принтер - EPSON или IBM - совместимый, то у него русский шрифт лежит в верхней половине кодовой таблицы, т.е. символы имеют такие коды, которые "Спектрум" для символов не использует. В этом случае Вам придется написать программу перекодировки кода символа перед выдачей его на принтер. Зато здесь не надо переключать принтер с русского набора на латинский, и наоборот, с помощью управляющих кодов. В таблице 6.3 показаны коды русских букв для EPSON и для IBM-совместимых принтеров (кодировка ГОСТ-альтернативная).

Таблица 6.2.

Код КОИ-7

Код	Англ.	Рус.	Код	Англ.	Рус.
64	@	ю	96	фунт ст.	Ю
65	A	а	97	a	А
66	B	б	98	b	Б
67	C	ц	99	c	Ц
68	D	д	100	d	Д
69	E	е	101	e	Е
70	F	ф	102	f	Ф
71	G	г	103	g	Г
72	H	х	104	h	Х
73	I	и	105	i	И
74	J	й	106	j	Й
75	K	к	107	k	К
76	L	л	108	l	Л
77	M	м	109	m	М
78	N	н	110	n	Н
79	O	о	111	o	О
80	P	п	112	p	П
81	Q	я	113	q	Я
82	R	р	114	r	Р
83	S	с	115	s	С
84	T	т	116	t	Т
85	U	у	117	u	У
86	V	ж	118	v	Ж
87	W	в	119	w	В
88	X	ь	120	x	Ь
89	Y	ы	121	y	Ы
90	Z	з	122	z	З
91	[	ш	123	{	Ш
92	\	э	124	|	Э
93	]	щ	125	}	Щ
94	□	ч	126	~	Ч
95	_	ъ	127	"копирайт"	

Таблица 6.3.

Кодировка русских символов ГОСТ-альтернативная.

Код	Символ	Код	Символ	Код	Символ
128	А	151	Ч	174	о
129	Б	152	Ш	175	п
130	В	153	Щ	176	

Продолжение таблицы 6.3.

Код	Символ	Код	Символ	Код	Символ
131	Г	154	Ъ	.	псевдо-
132	Д	155	Ы	.	гра-
133	Е	156	Ь	.	фика
134	Ж	157	Э	224	р
135	З	158	Ю	225	с
136	И	159	Я	226	т
137	Й	160	а	227	у
138	К	161	б	228	ф
139	Л	162	в	229	х
140	М	163	г	230	ц
141	Н	164	д	231	ч
142	О	165	е	232	ш
143	П	166	ж	233	щ
144	Р	167	з	234	ъ
145	С	168	и	235	ы
146	Т	169	й	236	ь
147	У	170	к	237	э
148	Ф	171	л	238	ю
149	Х	172	м	239	я
150	Ц	173	н		

6.2.2. Русификация "под пишущую машинку".

Те, кто хорошо знают клавиатуру русской пишущей машинки, могут предпочесть провести русификацию так, чтобы раскладка клавиатуры "Спектрума" более или менее соответствовала бы раскладке пишущей машинки. В этом случае Вам безусловно придется конвертировать коды перед выдачей их на принтер, независимо от того, какой у Вас принтер - ROBOTRON или EPSON и в какой половине кодовой таблицы он содержит русский шрифт - в нижней или в верхней.

Соответствие клавиш "Спектрума" и русских букв, закрепляемых за ними (как прописных, так и строчных), показано в табл. 6.4.

Таблица 6.4.

Q/Й	W/Ц	E/У	R/К	T/Е	Y/Н	U/Г	I/Ш	O/Щ	P/З
A/Ф	S/Ы	D/В	F/А	G/П	H/Р	J/О	K/Л	L/Д	
Z/Я	X/Ч	C/С	V/М	B/И	N/Т	M/Ь			

В результате такой "привязки" нескольким буквам не хватило места. Это буквы: Б,б,Ж,ж,Х,х,Ъ,ъ,Э,э,Ю,ю. Их придется "привязать" к символам, набираемым с нажатой клавишей SYMBOL SHIFT. Выберите те символы, которые редко используются и привяжите эти буквы к ним, например, @,#,$,%,&,<=,<>,>=,<,>,"фунт","□".

6.3 Полезные советы.

Если Вы сами пишете программы, то пожалуйста прислушайтесь к нескольким практическим рекомендациям, связанным с русификацией компьютера. Они проверены многолетним опытом и могут Вам пригодиться.

1. Никогда не рассчитывайте на то, что у будущего пользователя Вашей программы компьютер русифицирован так же, как у Вас. Если Вы используете русский знакогенератор, даже если он у Вас "зашит" в ПЗУ, все равно дистрибутивную программу поставляйте со своим знакогенератором. Прогружайте его в компьютер перед загрузкой программы и переключайте CHARS перед выводом всех сообщений на печать.

2. Никогда не устраивайте в программе двусторонних диалогов на русском языке. Сами пишите пользователю все, что хотите, но не требуйте от него никаких вводов на русском языке, особенно если есть шанс, что с программой будет работать ребенок. Не доводите дело до того, чтобы он размазывал слезы по клавиатуре в поисках клавиши, на которую Вы "подвесили" букву "Щ", особенно если она "висит" на символе "}", который не так-то просто нажать.

3. Даже в тех случаях, когда надо дать простой ответ типа "Д/Н" (Да/Нет) не надо требовать нажатия клавиш "Д" или "Н". Можете сделать запрос типа "Y/N, как это принято во всем мире. Если не хотите, то сделайте так:

1 - "Да"
2 - "Нет"

- и проверьте, что он нажал - клавишу "1" или "2".

4. Если очень нужно, чтобы пользователь ввел что-то по-русски, "подсуньте" ему готовые альтернативы:

1. Идти налево.
2. Идти направо.
3. Идти вверх.
4. Копать.
5. Кричать "Караул!"

Организуйте ему выбор из такого меню то ли нажатием соответствующей цифры, то ли перемещением визуального указателя, то ли изменением цвета выбираемой опции - как хотите.

5. Даже если Вам хочется, чтобы пользователь ввел свое имя, не надо это делать. Для установления дружеского диалога дайте ему несколько псевдонимов, пусть выберет себе тот, кото-

рый понравится.

1. Джонни 3. Бекки
2. Робби 4. Терри

Если перед Вами целый класс учеников, то ведь у каждого есть номер по классному журналу и Вам надо подумать, как в игровом духе обыграть этот номер. Номер ученик вполне может ввести сам, например:

Агент 007
Пилот Б-13
Капитан космического фрегата
"Грозный", бортовой номер 22.

6. Устраивая игры типа "Поле Чудес", Вы рассчитываете, на то, что пользователь будет вводить некую букву. Не рассчитывайте - это нетактично. Подумайте, как покрасивее дать ему возможность эту букву выбрать без ввода с клавиатуры. Может быть, ему надо сбить самолет с пролетающей нужной буквой, может быть ему надо покрутить какой-то барабан, пока в нем не высветится нужная буква - используйте все, что хотите, но не просите ничего вводить с клавиатуры русскими буквами.

7. Если же Ваша программа именно для того и предназначена, чтобы вводить что-то русскими буквами (например, если это база данных), то во-первых, перед началом программы дайте табличку, где какая буква находится, а во-вторых, обеспечьте, чтобы пользователь мог эту табличку вызвать для справки в любое время. Например, назначьте какой-то "горячей клавише", скажем "SYMBOL SHIFT" + "2" вызов на экран этой таблички и во всех состояниях ожидания действий пользователя организуйте проверку нажатия этой клавиши. Ту часть экрана, которая "портится" при выдаче этой таблички, надо где-то сохранить переброской командой LDIR, а после выхода из таблички - точно так же и восстановить.

7. ЗАКЛЮЧЕНИЕ.

Глубокоуважаемый читатель. Мы благодарим Вас за внимание, проявленное к нашей книге. Если Вы прочитали ее до конца и все в ней Вам понятно, то Вы сделали большой шаг вперед, но все же это только самый первый шаг.

Мы надеемся на то, что Вы не остановитесь на этом и пойдете дальше, ведь у персонального компьютера есть одно замечательное свойство - он дает возможность человеку самоутвердиться. Неважно сколько ему лет, неважен уровень его образования, неважно с какими программами он работает, но каждый день он самоутверждается. Он одерживает большие и малые победы, ощущает себя то ли первооткрывателем, то ли творцом, то ли покорителем - и всегда это так и есть на самом деле.

Мы готовы поддерживать Вас и далее на этом пути. Наша следующая задача - многотомное издание по графике "Спектрума". Всего в работе находятся 4 тома, объемом по 200 - 220 страниц каждый.

т.1 "Элементарная графика". Рассмотрены вопросы, связанные графикой на БЕЙСИКе, подробно рассмотрена работа с графикой из машинного кода, даны сведения о применении встроенных процедур ПЗУ, обращено внимание на тонкости их использования, связанные с наличием в ПЗУ системных ошибок и неточностей, имеется масса практических рекомендаций и приемов, даны ранее не опубликованные сведения, касающиеся особенностей 128 килобайтных машин. Это базовая книга всей серии.

т.2 "Прикладная графика". В нем рассмотрены вопросы конкретного применения графики для деловых, обучающих, научных, игровых и прочих программ. Рассмотрена растровая, векторная, блочная, трехмерная, теневая графика и пр.

т.3 "Динамическая графика" - рассмотрены вопросы анимации

графических изображений. Основная направленность - на создание обучающих и игровых программ.

т.4."Дизайн Ваших программ" - квинтэссенция того, к чему читатель шел, работая над первыми томами серии.

Кроме того, мы рекомендуем Вам приобрести выпуски нашего периодического издания "ZX-РЕВЮ", издаваемого с 1991 года. В настоящий момент есть возможность приобретения комплектов за 91-ый и 92-ой годы, выполненных в виде отдельной книжки того же формата, как и данная книга. Их объем - по 264 страницы, но за счет очень плотной печати и мелкого шрифта, каждый сборник содержит материал, равный примерно 1200 страницам машинописного текста, охватывающий практически весь спектр интересов пользователей "ZX-Spectrum", в том числе и любителей игрового программного обеспечения.

Есть возможность подписаться и на выпуски текущего года.

По всем вопросам приобретения нашей литературы и подписки на "ZX-РЕВЮ" Вы можете обратиться с запросом по нашему адресу:

121019, Москва, Г-19, а/я 16,
"ИНФОРКОМ"

Приложите конверт с заполненным обратным адресом и Вам будет выслан рекламный листок с предложением имеющейся на данный момент литературы и бланк-заказ.

Если Вы проживаете на Украине, Вы можете также обращаться к нашему представителю в этом регионе:

320030, Украина,
г.Днепропетровск,
ул. Шевченко, 34,