

NNNN - номер строки, содержащей ошибку.

Ниже приводятся коды ошибок и их описания:

- 1 - вход в цикл не через оператор FOR, или переменная, использованная в FOR, не соответствует переменной, использованной в NEXT.
- 2 - синтаксическая ошибка: неправильная запись операторов, неправильное использование символов (скобок, запятых и других символов).
- 3 - оператор RETURN был обнаружен без выполнения GOSUB.
- 4 - при выполнении оператора READ обнаружено, что список оператора DATA исчерпан.
- 5 - значение аргумента оператора (или функции) выходит за пределы допустимого интервала. Неправильная запись команд оператора DRAW.
- 6 - результат арифметической операции не может быть записан в формате, принятом для чисел в Бейсике.
- 7 - переполнение памяти: не хватает места в памяти для программы (или для размещения переменных).
- 8 - в операторе перехода используется номер несуществующей строки.
- 9 - неправильно указаны индексы элементов массива.
- 10 - попытка повторного определения массива.
- 11 - деление на нуль или возведение нуля в отрицательную степень.
- 12 - использование команды в тексте программы.
- 13 - неправильное использование типов данных. Например, попытка присвоить строковой переменной числовое значение.
- 14 - в памяти не хватает места строковым переменным.
- 15 - попытка создать строку длиной более 255 символов.
- 17 - продолжение выполнения программы невозможно.
- 18 - неопределена функция пользователя оператором DEF.
- 19 - ошибка устройства ввода/вывода.
- 24 - в выражении отсутствует операнд (число или переменная).

Глава 4. ПРОГРАММИРОВАНИЕ В МАШИННЫХ КОДАХ

4.1. Что понимается под программированием в кодах

Для чего нужно умение программировать в кодах ? Если Вы уже освоили с помощью предыдущих глав программирование на языке Бейсик, то, вероятно, можете сами ответить на этот вопрос. Основных причин может быть две. Во-первых, Бейсик не является чистым компилятором; создаваемый им шитый код выполняется в самом лучшем случае в несколько раз медленнее, чем аналогичная программа в машинных кодах. Если говорить о Фокале, то это чистый интерпретатор и работает еще медленнее.

Решение проблемы быстрогодействия очень просто: надо использовать не интерпретаторы, а компиляторы с языков высокого уровня. Компиляторами называются программы, сразу полностью переводящие программу в машинные коды. Полученную программу в машинных кодах можно сохранять и запускать сколько угодно раз. Только вот беда - ни один из компиляторов (например, с языков FORTRAN, PASCAL и т.п.) в БК не поместится. Для того, чтобы подготовить программу на этих языках для БК, нужно использовать инструментальную ЭВМ с большими возможностями (очень удобно это делать на ДБК в КУВТ). Но при отсутствии ДБК единственный выход - писать программы в кодах.

Вторая причина, по которой программирование в кодах на БК предпочтительнее, это ограниченный объем памяти БК. Чем плох Бейсик ? Тем, что он хранит в памяти как сам исходный текст программы, так и полученный при его компиляции шитый код. Да еще нужно отвести место под данные программы. Если команда сложения на Бейсике "съедает" 52 байта (см. табл. 7), то в кодах она займет максимум 6 байт.

Итак, Ваш выбор – программирование в кодах. Несмотря на распространенность этого понятия, оно не совсем соответствует действительности. Непосредственно в кодах на БК практически никто не программирует. Обычно пользуются разными средствами облегчения этой работы – ассемблерами и отладчиками, которые позволяют, как минимум, записывать машинные команды в удобной для человека форме (мнемонике).

Система МИКРО (А.Сомов, С.Шмытов, С.Кумандин, г.Москва) даже обеспечивает редактирование исходного текста и компиляцию программы на языке низкого уровня – ассемблере. Этот компилятор проще, чем компиляторы языков высокого уровня, и помещается в памяти БК, однако на текст программы и ее код остается мало места. Находящийся в ОЗУ текст программы можно транслировать, при этом получаем так называемый объектный модуль, который можно записать на магнитофон. Затем из небольших объектных модулей, загружаемых с магнитофона, собирается (компонуются) загрузочный модуль – собственно программа в кодах. Вынужденная работа мелкими кусочками, требующая частого использования магнитофона – плата за удобство программирования на ассемблере.

Наиболее удачным ассемблером-отладчиком для БК на сегодняшний день является система MIRAGE С.Зильберштейна (г. Киров). Здесь текст программы на ассемблере как таковой отсутствует. MIRAGE переводит коды, хранящиеся в ОЗУ с указанного адреса, в ассемблерную мнемонику (дизассемблирует). С помощью редактора в пределах одного экрана можно корректировать команды или добавлять новые, при этом они ассемблируются и записываются в виде машинных кодов в то же место ОЗУ. Такой способ работы имеет ряд недостатков – например, MIRAGE не может отличить данные от программы и также пытается их дизассемблировать. Но главное неудобство в том, что программисту все время приходится самому заботиться об адресах, и при вставке новой команды (или удалении) изменять адреса во всех командах перехода. Однако при определенной дисциплине и навыках работы это неудобство не смертельно. Зато в распоряжении программиста находится все ОЗУ в полном объеме, а магнитофон нужен только для периодического сохранения измененной программы. Плюс к тому MIRAGE позволяет отлаживать программу в кодах – запускать отдельные участки программы, прогонять ее по шагам с распечаткой текущих значений регистров процессора и т.п.

Только в том случае, когда у Вас не окажется ни одного из этих инструментов, программу можно писать непосредственно в кодах. Ввести ее в БК можно с помощью ТС-отладчика (см.п.2.5), либо с помощью Бейсика, как будет показано ниже. Еще одним способом улучшения качества программ может служить сочетание основной программы на Бейсике с подпрограммами в кодах. Об этом также будет рассказано ниже.

Рекомендуем после прочтения этой главы изучить следующую, взять имеющийся у Вас отладчик (п.п.5.1, 5.2) и с его помощью подробно изучить работу каждой команды обработки данных, описанной в данной главе. С помощью отладчиков Вы можете набирать короткие программы и, трассируя их, воочию увидеть, как работают те или иные команды, как при этом меняется содержимое регистров и как выставляются признаки в слове состояния процессора.

Рекомендуем также Вашему вниманию журнал "Информатика и образование", который в 1990 году начал публикацию цикла статей Ю.Зальцмана по архитектуре БК и программированию в кодах. В "Клубе пользователей БК", располагающемся в этом журнале, часто выступают известные программисты и любители БК, делящиеся полезными советами и разными тонкостями программирования.

4.2. Используемые в БК типы данных

Как Вы знаете, Бейсик позволяет работать с данными арифметического и текстового (строкового) типов, причем арифметические данные могут быть целого и вещественного (одинарной и двойной точности). Конечно, процессор БК не имеет таких широких возможностей. Например, обработка

вещественных чисел может быть организована только программно, причем различные системы программирования (Бейсик, Фокал, PASCAL) могут использовать даже различные способы представления этих чисел).

Основными данными, с которыми может работать процессор БК, являются байт и слово. Соответственно, в слове может быть размещено целое число, а в байте – один символ текста.

В одном слове 16 бит – двоичных разрядов, поэтому из них можно составить $2^{16} = 65536$ различных комбинаций. Если учесть, что машинные команды работы с целыми числами считают старший разряд (15-й бит) слова знаковым (0 – число положительное, 1 – число отрицательное), то диапазон целых чисел в БК будет от -32768 до +32767. Если в результате вычислений в Бейсике получится больший результат, он выдаст ошибку 6. Если то же самое произойдет в Вашей программе в кодах, никто этого не заметит, если не принять специальных мер. Тогда, прибавив единицу к числу 32767, Вы получите -32768.

Для того, чтобы представить символы в памяти ЭВМ, их кодируют (см. Приложение 1). В одном байте – 8 бит, что позволяет закодировать $2^8 = 256$ различных символов. Этого вполне достаточно, чтобы разместить строчные и заглавные буквы двух алфавитов, цифры и другие спецзнаки. Именно для удобства кодировки символов и был выбран такой размер байта как единицы памяти ЭВМ.

Следует также отметить, что для кодирования символов в БК использован советский стандарт КОИ – 8 бит, базирующийся на международном стандарте ASCII (American Standard Code for Information Interchange).

4.3. Программная модель процессора БК

Процессор ЭВМ – это сложная электронная схема. Однако знать все его особенности при программировании в кодах незначит. То, что нужно знать программисту о каком-либо устройстве ЭВМ, называется его программной моделью. Сюда относятся программно-доступные ячейки памяти устройства (их называют регистрами), а также алгоритм функционирования устройства.

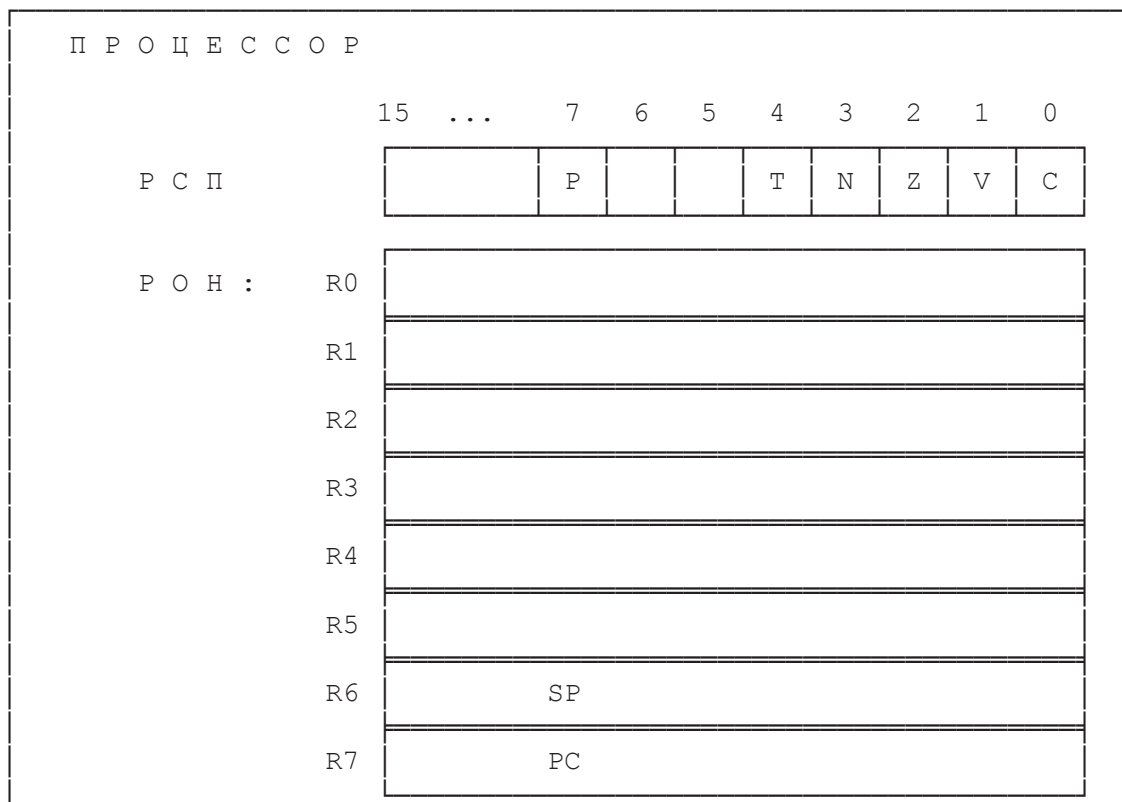


Рис.11. Программная модель микропроцессора K1801BM1

Итак, используемый в БК микропроцессор K1801BM1 (рис.11) имеет 8 регистров общего назначения (РОН) и регистр слова состояния процессора (РСП). Все эти регистры 16-разрядные.

Регистры общего назначения R0, R1, R2, R3, R4, R5 предназначены для хранения промежуточных результатов вычислений. Операции пересылки данных из одного регистра в другой выполняются гораздо быстрее, чем пересылка данных в ОЗУ. Оптимальное использование РОН в часто выполняемых циклах программы позволяет иногда ускорить ее на порядок.

Регистр R6 обозначается в ассемблере "SP" (Stack Pointer – указатель стека) и используется в этом качестве как при вызове подпрограмм, так и при обработке прерываний. О нем будет подробно рассказано далее.

Регистр R7 обозначается "PC" (Program Counter – счетчик команд) всегда содержит адрес следующей команды, которую должен выполнить процессор. Рассмотрим подробнее алгоритм работы процессора, то есть опишем, как он выполняет программу.

Программа должна начинать работу с определенного адреса памяти (с определенной команды). Для этого двоичный код адреса, с которого начинается программа, должен быть предварительно записан в регистр PC процессора (счетчик команд) – так как регистр PC всегда хранит адрес очередной команды, подлежащей выполнению.

Допустим, в регистр PC записали восьмеричное число 7000 (все коды и адреса будем писать в восьмеричной системе счисления). Это значит, что следующей командой, исполняемой процессором, будет команда, хранящаяся по адресу 7000. Произойдет это так: из памяти с адреса 7000 в процессор перепишется слово – двоичный код команды для выполнения. Например, пусть по адресу 7000 была записана команда 60001.

По этой команде процессор должен сложить содержимое R0 с содержимым R1. Причем результат сложения (сумма) запишется в R1. После выполнения процессором этой команды содержимое регистра PC автоматически увеличится на 2 (станет равным 7002). Теперь уже в процессор будет передана команда из ОЗУ с адреса 7002, и так далее.

Действие команд перехода заключается в записи в регистр PC нового значения, и тогда процессор продолжит выполнять программу с указанного адреса.

А что будет, если процессор в результате ошибки программиста прочитает непонятную ему команду? Ничего страшного, просто в этом случае выполнение программы будет прервано и процессор начнет выполнять программу из системного ПЗУ (как говорят, программа вылетит в МОНИТОР), либо управление возьмет на себя используемый отладчик (например, MIRAGE). Аналогичная ситуация возникнет, если в PC записать нечетное число (Вы помните, что длина команды кратна машинному слову, т.е. двум байтам, и ее адрес должен быть четным), либо такого адреса в микроЭВМ не существует.

Остается только добавить, что после выполнения каждой команды программы меняется содержимое РСП. Этот регистр предназначен для хранения PSW (Processor Status Word – слово состояния процессора). В PSW имеют значение следующие биты (разряды):

бит 0 (C) устанавливается в 1, если при выполнении команды произошел перенос единицы из старшего разряда результата;

бит 1 (V) устанавливается в 1, если при выполнении арифметической команды (например, сложения) произошло арифметическое переполнение;

бит 2 (Z) устанавливается в 1, если результат равен нулю;

бит 3 (N) устанавливается в 1, если результат отрицателен. Эти биты PSW используются командами условного перехода. Остальные биты PSW устанавливаются программистом для задания режимов работы процессора:

бит 4 (T), установленный в 1, вызывает после выполнения очередной команды прерывание по вектору 14. Это используется программами –

отладчиками для трассировки программы;

бит 7 (P), установленный в 1, запрещает (маскирует) прерывания от внешних устройств (например, клавиатуры). Таким образом, командой установки PSW MTPS #200 можно запретить прерывания от клавиатуры до тех пор, пока не выполнится команда MTPS #0.

4.4. Система команд процессора БК

Процессор K1801BM1 "понимает" 64 разные команды. Все эти команды, вместе взятые, составляют систему команд процессора. Как уже говорилось, система команд процессора БК практически совпадает с системой команд целой серии мини- и микроЭВМ (Электроника-60, ДВК, СМ).

Команды процессора K1801BM1 условно можно разделить на 4 группы: однооперандные команды, двухоперандные команды, команды передачи управления и безоперандные команды.

Двоичный код безоперандной команды содержит только код операции – информацию для процессора о том, что нужно делать по этой команде.

Двоичный код однооперандной команды содержит код операции и информацию для процессора о местонахождении обрабатываемого числа (операнда), над которым нужно произвести операцию.

Двухоперандные команды, помимо кода операции, содержат информацию для процессора о местонахождении 2-х чисел (операндов). Например, для сложения 2-х чисел команда должна содержать код операции сложения и информацию о том, откуда взять слагаемые.

Далее коды команд процессора, а также коды операндов будем писать в восьмеричной системе счисления (как и коды адресов). Однако полезно помнить, что команды и операнды хранятся в памяти в двоичном коде – восьмеричный код мы будем использовать только для удобства. На рис.12 показано, как переводить 16-разрядный двоичный код числа в восьмеричный код.

0	101	000	110	001	111		- двоичное число.
□	□	□	□	□	□		
0	5	0	6	1	7		- восьмеричное число.

Рис.12. Перевод двоичного кода числа в восьмеричный

Число 050617 в восьмеричной системе счисления получено из 16-разрядного двоичного кода 0101000110001111 таким образом. Начиная с младшего разряда (справа) двоичное число делится на триады (группы по 3 цифры). Правда, старший разряд 16-разрядного числа при этом остается без "соседей". Затем для каждой триады записывается ее представление в восьмеричной системе счисления. В результате вместо 16-разрядного кода числа получаем 6-разрядный восьмеричный код. Разумеется, при написании программ удобнее работать с 6-разрядными кодами команд и чисел, чем с 16-разрядными – благо, все имеющиеся программы, облегчающие программирование в кодах, включая пультный отладчик, понимают восьмеричную систему счисления.

4.4.1. Способы адресации операнда

В однооперандных командах процессора первые 4 восьмеричные цифры определяют код операции. Оставшиеся 2 цифры в коде команды процессор использует для определения местонахождения операнда, над которым нужно произвести операцию.

Например, команда 005004 обнуляет регистр R4. Здесь первые 4 цифры кода команды "0050" являются кодом операции и указывают на то, что процессор должен произвести обнуление. Оставшиеся 2 цифры "04" указывают, что производится обнуление регистра R4. Нетрудно догадаться, что последняя цифра является номером того регистра, содержимое которого используется процессором при определении местонахождения операнда.

Предпоследняя цифра (код способа адресации) указывает, каким образом содержимое регистра используется при определении местонахождения операнда, то есть определяет способ адресации.

В таблице 8 приведены способы адресации, использующие регистры общего назначения, кроме регистра РС.

В первом столбце таблицы приведены названия, а во втором столбце – соответствующие восьмеричные коды способов адресации. В третьем столбце дается описание каждого способа адресации, а в четвертом столбце – соответствующие примеры на языке ассемблера.

Таблица 8. Способы адресации через регистры R0–R6

способ адресации	код	описание способа адресации	примеры
регистровый	0	регистр содержит операнд	CLR R1
косвенно – регистровый	1	регистр содержит адрес операнда	CLR (R1) CLR @R1
автоинкрементный	2	регистр содержит адрес операнда. Содержимое регистра после его использования в качестве адреса увеличивается на 2 (для команд над словами) или на 1 (для байтовых команд)	CLR (R2) +
косвенно – автоинкрементный	3	регистр содержит адрес адреса операнда. Содержимое регистра после его использования в качестве адреса увеличивается на 2	CLR @(R2) +
автодекрементный	4	содержимое регистра уменьшается на 2 (для команд над словами) или на 1 (для байтовых команд) и используется как адрес операнда	CLR -(R2)
косвенно – автодекрементный	5	содержимое регистра уменьшается на 2 и используется как адрес адреса операнда.	CLR @-(R1)
индексный	6	содержимое регистра складывается с числом, записанным после команды, и полученная сумма используется в качестве адреса операнда	CLR 2(R5) CLR MT(R0)
косвенно – индексный	7	содержимое регистра складывается с числом, записанным после команды и полученная сумма используется в качестве адреса адреса операнда	CLR @22(R1)

В команде 005004, например, применен регистровый способ адресации, использующий регистр R4.

Если адресация операнда происходит через регистр РС, то способы адресации другие (таблица 9).

Таблица 9. Способы адресации через регистр PC

способ адресации	код	описание способа адресации	примеры
непосредственный	2	операнд хранится в слове, следующем за командой за командой.	MOV #21,R3 MOV #IN,R0
абсолютный	3	адрес операнда хранится в слове, следующем за командой	CLR @#7000 JMP @#BEN
относительный	6	содержимое PC складывается со словом, записанным в памяти за командой, и полученная сумма используется как адрес операнда.	JMP TV CLR 5554
косвенно - относительный	7	содержимое PC складывается со словом, следующим за командой, полученная сумма используется как адрес адреса операнда.	CLR @MET INC @15342

4.4.2. Однооперандные команды

Перейдем к рассмотрению однооперандных команд. Одна из них, команда обнуления регистра процессора или ячейки памяти, нам уже знакома:

0050DD

Так как для каждого конкретного способа адресации последние две цифры в коде однооперандной команды будут разными (какими - зависит от программиста), то здесь и далее при написании кода какой-либо однооперандной команды вместо последних 2-х цифр будем писать "DD". Ту часть кода команды, которая обозначена двумя буквами "DD", будем называть полем адресации операнда (рис.13).

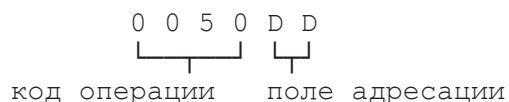


Рис.13. Пример однооперандной команды

То, что команда имеет один операнд, еще ничего не говорит о длине команды. Это зависит от способа адресации операнда. Если способ адресации регистровый, то команда занимает одно слово, если индексный либо с использованием PC - два слова (во втором слове помещается операнд, его адрес либо индекс).

В таблице 10 приводится описание однооперандных команд.

Таблица требует пояснений. Некоторые команды могут обрабатывать как целые слова, так и байты. В таблице вместо первой цифры кода каждой такой команды стоит символ "*". В том случае, если команда предназначена для работы с байтом, то в ее коде первой цифрой ставится "1", а иначе - "0". Если в качестве операнда байтовой команды служит один из регистров общего назначения процессора, то имеется в виду его младший байт.

Каждая команда системы команд, помимо своего числового кода, имеет свою мнемонику (мнемокод) - обозначение в виде последовательности нескольких латинских букв, что используется при программировании на

языке ассемблера. Например, мнемокод команды обнуления содержимого какого-либо регистра (или слова памяти) состоит из трех букв: CLR (от английского слова Clear – очистить).

Команда, предназначенная для обработки байта (байтовая команда), имеет такой же мнемокод, что и соответствующая команда для обработки слова, но к мнемокоду справа добавляется буква "B". Например, команда, предназначенная для очистки отдельного байта, имеет мнемокод: CLRB (от английского Clear Byte – очистить байт).

Для команд, которые могут обрабатывать как слова, так и байты, в таблице в скобках указан возможный суффикс "B".

Таблица 10. Основные однооперандные команды

код команды	мнемокод команды	действие команды
*050DD	CLR(B)	Clear – очистить
*051DD	COM(B)	Complement – инвертировать
*052DD	INC(B)	Increment – увеличить
*053DD	DEC(B)	Decrement – уменьшить
*054DD	NEG(B)	Negate – изменить знак
*055DD	ADC(B)	Add Carry – сложить с переносом
*056DD	SBC(B)	Subtract Carry – вычесть перенос
*057DD	TST(B)	Test – тестировать, проверить
*060DD	ROR(B)	Rotate Right – циклически сдвинуть вправо
*061DD	ROL(B)	Rotate Left – циклически сдвинуть влево
*062DD	ASR(B)	Arithmetic Shift Right – арифметически сдвинуть вправо
*063DD	ASL(B)	Arithmetic Shift Left – арифметически сдвинуть влево
0003DD	SWAB	Swap Bytes – переставить байты
1064DD	MTPS	Move To PSW(Processor Status Word) – записать слово состояния процессора)
1067DD	MFPS	Move From PSW – прочитать PSW

Рассмотрим подробнее каждую команду процессора (для определенности будем иметь в виду команды, обрабатывающие слово памяти или содержимое регистра процессора, поэтому далее при написании кода команды вместо символа "*" будем проставлять цифру "0").

Первая команда (CLR) нам уже знакома – перейдем к рассмотрению остальных команд.

Команда 0051DD. Мнемокод: COM

По этой команде образуется инверсный (обратный) код операнда – во

всех разрядах операнда нули заменяются единицами, а единицы - нулями. Это действие представляет собой логическую операцию отрицания ("НЕ").

Пример:

```
005121          COM (R1)+
```

Команда образует инверсный код слова, адрес которого хранится в регистре R1, после чего содержимое регистра R1 увеличивается на 2.

Далее в примерах справа от каждой команды будем писать соответствующую ассемблерную мнемонику.

Команда 0052DD. Мнемокод: INC

Увеличивает значение операнда на 1. Пример:

```
005237          INC @#10000
010000
```

Команда 005237 увеличивает на 1 содержимое слова памяти с адресом 10000 - на 1 увеличивается число, хранящееся в 2-х ячейках с адресами 10000 и 10001. Две последние цифры "37" в коде команды указывают, что адрес операнда хранится в памяти за кодом команды (абсолютная адресация).

Значение адреса 10000 в примере записано под кодом команды 005237 (в один столбец). Такая запись означает, что начиная с определенного адреса (например, с адреса 7000) в памяти записан код команды 005237 (два байта), затем (по адресу 7002) записано число 10000.

Команда 0053DD. Мнемокод: DEC

Уменьшает значение операнда на 1. Пример:

```
005312          DEC (R2)
```

Команда уменьшает на 1 слово, адрес которого хранится в регистре R2.

Команда 0054DD. Мнемокод: NEG

Изменяет знак операнда. Пример:

```
005412          NEG @R2
```

Команда изменяет знак числа, хранящегося по адресу, указанному в R2. Если число было положительным, то станет отрицательным, и наоборот.

Операция изменения знака числа в процессоре БК эквивалентна получению дополнительного кода числа (то есть числа, которое, будучи сложено с исходным, даст в сумме ноль). Формирование дополнительного двоичного кода числа состоит из двух последовательных операций: получения инверсного (обратного) двоичного кода числа и прибавления 1 (к полученному инверсному коду).

Пример:

```
01001011 - двоичный код операнда (байта)
10110101 - дополнительный двоичный код операнда
```

Отметим также, что в двух предыдущих примерах обозначения "(R2)" и "@R2" равнозначны.

Команда 0055DD. Мнемокод: ADC

Увеличивает значение операнда на содержимое разряда C PSW.

Команда 0056DD. Мнемокод: SBC

Уменьшает значение операнда на содержимое разряда C PSW.

Команда 0057DD. Мнемокод: TST

По этой команде производится тестирование (проверка) значения операнда и установка в "1" (или сброс в "0") разрядов Z и N PSW, а

разряды V и C при этом сбрасываются в "0". Если значение операнда отрицательно, то разряд N PSW устанавливается в "1", иначе сбросится в "0". Если значение операнда нулевое, то разряд Z устанавливается в "1", иначе сбросится в "0". Значение операнда при этом не изменяется.

Команда 0060DD. Мнемокод: ROR

Производит циклический сдвиг значений разрядов операнда вправо на один разряд (рис.14). Значение 15-го разряда загружается в 14-ый разряд, значение 14-го разряда - в 13-ый разряд и так далее.

Значение разряда C PSW загружается в 15-ый разряд операнда, а значение нулевого (младшего) разряда операнда - в разряд C PSW.

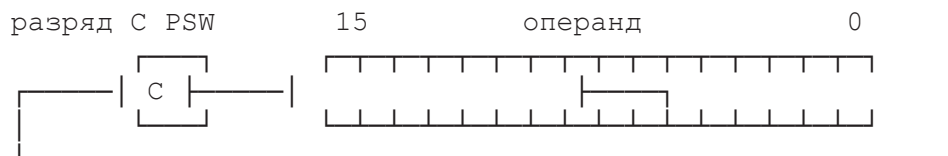


Рис.14. Циклический сдвиг вправо

Команда 0061DD. Мнемокод: ROL

Команда выполняется так же, как и предыдущая, но сдвиг выполняется влево и значение разряда C PSW загружается в нулевой разряд операнда, а значение 15-ого разряда операнда - в разряд C PSW.

Команда 0062DD. Мнемокод: ASR

Производит арифметический сдвиг вправо. При этом значение каждого разряда операнда сдвигается на один разряд вправо. Значение нулевого разряда операнда загружается в разряд C PSW. В 14-ый и 15-ый разряды записывается значение 15-ого разряда операнда (рис.15).

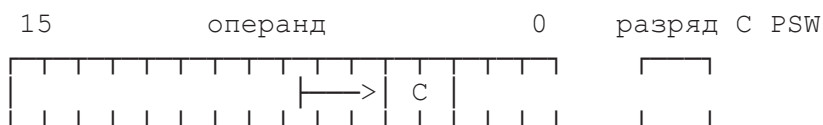


Рис.15. Арифметический сдвиг вправо

Команда 0063DD. Мнемокод: ASL

Производит арифметический сдвиг значения каждого разряда операнда на один разряд влево. Нулевой разряд операнда очищается, а значение 15-ого разряда операнда загружается в разряд C PSW (рис.16).

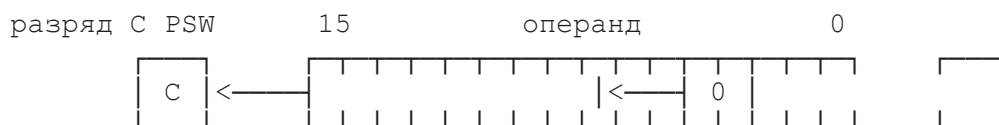


Рис.16. Арифметический сдвиг влево

Арифметический сдвиг влево равносителен умножению операнда на 2, а арифметический сдвиг вправо - делению операнда на 2.

Команда 0003DD. Мнемокод: SWAB

Меняет местами старший и младший байты операнда (слова).

Команда 1064DD. Мнемокод: MTPS

Команда записывает в РСР новое значение слова состояния процессора (PSW), равное значению операнда. Пример:

```
106427          MTPS #200
000200
```

В данном примере в РСР загружается новое значение PSW, равное 200. Таким образом, устанавливается в 1 бит приоритета "P" в PSW, что

запрещает процессору обрабатывать прерывания от клавиатуры до тех пор, пока не будет выполнена команда MTPS #0.

Команда 1067DD. Мнемокод: MFPS

Команда пересылает PSW в место, определяемое полем адресации операнда. Пример:

106702 MFPS R2

Производится пересылка PSW в регистр R2.

4.4.3. Двухоперандные команды

Код двухоперандной команды, кроме кода операции, должен содержать 2 поля адресации операнда. Первое поле адресации операнда, обозначаемое в коде команды двумя буквами "SS", определяет местонахождение 1-ого операнда команды и называется полем адресации операнда источника. Второе поле адресации операнда, обозначаемое в коде команды буквами "DD", определяет местонахождение 2-ого операнда команды и называется полем адресации операнда приемника (рис.17). Первый операнд команды называется операндом источника, второй операнд – операндом приемника. Смысл обозначений "SS" и "DD" такой же, что и для обозначения "DD" в коде однооперандной команды.

Основные двухоперандные команды, рассматриваемые здесь, приведены в таблице 11.

Команда 01SSDD. Мнемокод: MOV

По этой команде операнд, местонахождение которого определяется полем адресации "SS", пересылается по адресу, определяемому полем адресации "DD". При этом содержимое источника, откуда берется операнд для пересылки, не изменяется. При выполнении байтовой команды MOVБ с использованием регистрового способа адресации (для операнда приемника) все разряды старшего байта операнда приемника устанавливаются в "1", если знаковый разряд (старший разряд) младшего байта установлен в "1", иначе разряды старшего байта сбрасываются в "0". Пример:

010204 MOV R2,R4

Копия содержимого регистра R2 пересылается (загружается) в регистр R4. Содержимое R2 при этом не изменяется. Более сложный пример:

Команда	Текст на ассемблере
012737	MOV #177777,@#70000
177777	
070000	

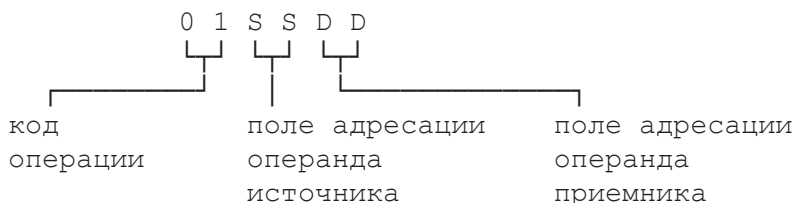


Рис.17. Пример двухоперандной команды

Таблица 11. Двухоперандные команды

код команды	мнемокод команды	действие команды
*1SSDD	MOV (B)	Move - переслать
*2SSDD	CMP (B)	Compare - сравнить

*3SSDD	BIT(B)	Bit Test - тестировать биты
*4SSDD	BIC(B)	Bit Clear - очистить биты
*5SSDD	BIS(B)	Bit Set - установить биты
06SSDD	ADD	Add - сложить
16SSDD	SUB	Subtract - вычесть
074RDD	XOR	логическая операция XOR (исключающее ИЛИ) над регистром R и приемником DD

Здесь одна команда занимает 3 слова памяти. В 1-ом слове записан код самой команды - число 012737. Во 2-ом слове хранится операнд источника (число 177777), предназначенный для пересылки, и в 3-ем слове хранится адрес приемника 70000, куда пересылается число 177777.

Две цифры "27" в коде команды, записанные в поле адресации операнда источника, указывают, что операнд источника (число 177777) хранится в памяти сразу же за кодом команды. Две цифры "37", записанные в поле адресации операнда приемника указывают, что значение адреса приемника хранится в памяти также за кодом команды - но в данном случае место в памяти сразу же за кодом команды занято числом 177777, поэтому значение адреса приемника (число 70000) записывается в памяти за числом 177777.

Команда 02SSDD. Мнемокод: CMP

Данная команда вычитает из операнда источника операнд приемника. Но при этом значения самих операндов не изменяются. Изменяются лишь значения разрядов C,V,Z,N PSW.

Разряд N устанавливается в "1", если результат вычитания - отрицательное число. Разряд Z устанавливается в "1", если результат - равен нулю (операнды равны). Разряд V устанавливается в "1", если было арифметическое переполнение - если операнды были противоположного знака, а знак результата совпадает со знаком операнда приемника. Разряд C обнуляется, если был перенос из старшего разряда результата вычитания. Пример:

020103

CMP R1,R3

Команда 020103 производит сравнение содержимого регистра R1 с содержимым регистра R3.

Команда 03SSDD. Мнемокод: BIT

Значение каждого разряда результата образуется логическим умножением значений соответствующего разряда операнда источника и операнда приемника. Например, значение 12-ого разряда результата образуется логическим умножением значений 12-ого разряда операнда источника и 12-ого разряда операнда приемника. В таблице 12 показано, чему равен результат логического умножения (операции "И") при разных комбинациях значений какого-либо разряда операнда источника и операнда приемника.

Значения операнда источника и операнда приемника при выполнении команды не изменяются. Изменяются лишь значения разрядов V,Z,N PSW. Разряд N устанавливается в "1", если в результате поразрядного логического умножения 2-х операндов получилось число, знаковый (старший) разряд которого установлен в "1". Разряд Z устанавливается в "1", если в результате логического умножения получилось число, все разряды которого сброшены в "0". Разряд V PSW сбрасывается в "0".

Данная команда используется для проверки состояния (значения) отдельных разрядов одного из операндов. Пример:

Команда Текст программы на ассемблере
 032737 BIT #100,@#177716 ;нажата ли клавиша ?
 000100
 177716
 0014.. BEQ MET ;переход к обработке нажатой клавиши

Таблица 12. Логические операции

значение разряда операнда источника	0	0	1	1
значение разряда операнда приемника	0	1	0	1
результат логического умножения (операции "И")	0	0	0	1
результат логического сложения (операции "ИЛИ")	0	1	1	1
результат операции "исключающее ИЛИ" (XOR)	0	1	1	0

Команда 04SSDD. Мнемокод: BIC

По этой команде сбрасываются в "0" (обнуляются) разряды операнда приемника, соответствующие установленным в "1" разрядам операнда источника. Остальные разряды операнда приемника остаются без изменений. Значение операнда источника при выполнении команды не изменяется. Действие этой команды эквивалентно выполнению последовательности логических операций: инвертирования (отрицание) операнда источника; логического умножения ("И") результата и операнда приемника; запись результата в операнд приемника. Пример:

```

1111111100000000 <---- двоичный код операнда источника
1010011110001010 <---- двоичный код операнда приемника
                        до выполнения команды
0000000010001010 <---- двоичный код операнда приемника
                        после выполнения команды
```

Команда 05SSDD. Мнемокод: BIS

По этой команде устанавливаются в "1" разряды операнда приемника, соответствующие установленным в "1" разрядам операнда источника. Остальные разряды операнда приемника остаются без изменений. Значение операнда источника при выполнении команды не меняется. Действие этой команды эквивалентно операции логического сложения ("ИЛИ") над операндами источника и приемника (см. табл. 12). Пример:

```

1111111100000000 <---- двоичный код операнда источника
1010011110001010 <---- двоичный код операнда приемника
                        до выполнения команды
1111111110001010 <---- двоичный код операнда приемника
                        после выполнения команды
```

При выполнении команд BIC и BIS значения разрядов V,Z,N PSW изменяются в зависимости от значения результата так же, как при выполнении команды BIT.

Команда 06SSDD. Мнемокод: ADD

Команда суммирует операнд источника с операндом приемника. Результат сложения записывается по адресу операнда приемника. Значение операнда источника при этом не изменяется. Значение разрядов Z и N PSW изменяются так же, как и для команды CMP. Разряд V PSW устанавливается в "1", если в результате выполнения команды произошло арифметическое переполнение, то есть, если оба операнда были одного знака, а результат сложения получился противоположного знака. В обычной

арифметике такого не бывает, а в работе процессора такой парадокс возможен в силу того, что разрядность процессора ограничена. Разряд C PSW устанавливается в "1", если был перенос из старшего разряда результата. Пример:

060204

ADD R2,R4

Производится сложение содержимого 2-х регистров: R2 и R4. Результат сложения помещается в регистр R4.

Команда 16SSDD. Мнемокод: SUB

По этой команде из операнда приемника вычитается операнд источника. Результат помещается по адресу операнда приемника. Изменение разрядов C,Z,N PSW происходит так же, как при выполнении команды CMP. Арифметическое переполнение при выполнении данной команды происходит, когда операнды имели разные знаки, а знак результата совпадает со знаком операнда источника. Пример:

Адрес	Команда	Текст программы на ассемблере
20000:	166767	SUB VR1,S
20002:	000004	
20004:	000006	
20006:	000207	RTS PC
20010:	000056	VR1: .#56
20012:	000012	VR2: .#12
20014:	000010	S: .#10

Здесь и далее в примерах по необходимости слева от каждой команды (или операнда) будем писать соответствующий адрес.

В приведенном фрагменте программы команда SUB, хранящаяся по адресу 20000, производит вычитание значения переменной VR1 из значения переменной S. Результат вычитания присваивается переменной S. По адресу 20002 хранится смещение для определения адреса операнда источника (значения переменной VR1), а по адресу 20004 – смещение для определения адреса операнда приемника (значения переменной S).

Для определения адреса операнда источника процессор складывает смещение, хранящееся по адресу 20002, с содержимым регистра PC. После пересылки указанного значения смещения в процессор содержимое регистра PC равно 20004 – поэтому искомое значение адреса операнда источника равно

$$\text{смещение} + \text{PC} = 4 + 20004 = 20010 .$$

Аналогично определяется адрес операнда приемника:

$$\text{смещение} + \text{PC} = 6 + 20006 = 20014 .$$

4.4.4. Команды передачи управления

4.4.4.1. Команды перехода

Каждая команда перехода занимает одно слово памяти (кроме команды JMP) и имеет формат, изображенный на рисунке 18.



Рис.18. Формат команды перехода

Команды JMP и SOB имеют другой формат и поэтому будут рассмотрены позже отдельно.

Команда перехода (ветвления) позволяет изменить содержимое счетчика команд (регистра РС) на величину смещения, указанного в младшем байте кода команды. А изменение содержимого счетчика команд на величину смещения приведет к продолжению выполнения программы с адреса, равного выражению:

$$АП = АК + 2 + 2 * СМ$$

где

АП – адрес, с которого продолжится выполнение программы после исполнения команды перехода (адрес перехода);

АК – адрес, где хранится сама команда перехода;

СМ – смещение, указанное в коде команды перехода.

Старший разряд смещения (разряд 7 в коде команды) является знаковым и равен "1" (для отрицательных смещений) или "0" (для положительных смещений). Так как для хранения значения смещения отводится только младший байт кода команды, то оно не может выйти за границы интервала от -128Д до +127Д.

Команда 000400. Мнемокод: BR

Действия процессора по команде BR соответствуют действию оператора GOTO в программе на языке Бейсик. Если оператор GOTO передает управление на определенную строку программы на Бейсике, то команда BR-на определенный адрес памяти (адрес перехода), начиная с которого будет продолжено выполнение программы. По команде BR произойдет безусловный переход на адрес перехода – процессор продолжит выполнение программы с адреса перехода.

Код команды BR равен 000400 только в том случае, если смещение равно 0. Кстати, в каждой команде, приведенной в таблице 13, значение смещения равно 0 (кроме команды JMP). Полный код команды определяется

Таблица 13. Команды перехода

код команды	мнемокод команды	действие команды
000400	BR	Branch – переход
001000	BNE	Branch if Not Equal – переход, если не равно
001400	BEQ	Branch if Equal – переход, если равно
100000	BPL	Branch if Plus – переход, если плюс
100400	BMI	Branch if Minus – переход, если минус
102000	BVC	Branch if V is Clear – переход, если разряд V PSW очищен (сброшен в "0")
102400	BVS	Branch if V is Set – переход, если разряд V PSW установлен в "1"
002000	BGE	Branch if Greater or Equal – переход, если больше или равно
002400	BLT	Branch if Less Then – переход, если меньше чем
003000	BGT	Branch if Greater Then – переход, если больше чем
003400	BLE	Branch if Less or Equal – переход, если меньше или равно

101000	BHI	Branch if Higher - переход, если выше
101400	BLOS	Branch if Lower or Same - переход, если ниже или столько же
103000	BHIS (BCC)	Branch if Higher or Same - переход, если выше или столько же (или бит C PSW равен 0)
103400	BLO (BCS)	Branch if Lower - переход, если ниже (или бит C PSW равен 1)
077R00	SOB	Subtract One and Branch, if not equal - вычесть 1 и сделать переход, если не 0
0001DD	JMP	Jump - прыгнуть (перейти)

сложением кода команды, указанного в таблице 13, и значения смещения, которое равно выражению

$$CM = (AP - AK - 2) / 2 .$$

Пример:

Адрес	Команда	Текст программы на ассемблере
7000:	000401	BR TT
7002:	005001	CLR R1
7004:	010102	TT: MOV R1,R2

В данном примере первая команда производит переход на адрес 7004. Значение смещения определилось так:

$$CM = (AP - AK - 2) / 2 = (7004 - 7000 - 2) / 2 = 1 .$$

Во фрагменте программы на языке ассемблера использована метка TT, обозначающая адрес памяти, где размещена команда MOV R1,R2. Команда BR TT передает управление на адрес, обозначенный ("помеченный") меткой TT. Имена всех меток на языке ассемблера заканчиваются символом ":".

Рассмотрим следующий пример, когда значение смещения - отрицательное число (переход производится в сторону младших адресов):

Адрес	Команда	Текст программы на ассемблере
35004:	005001	V0: CLR R1
...
35036:	000762	BR V0

Команда 000762 производит переход на адрес 35004.

Значение смещения определяется так:

$$CM = (AP - AK - 2) / 2 = (35004 - 35036 - 2) / 2 = -34 / 2 .$$

Значение смещения - отрицательное число, поэтому оно должно быть представлено в дополнительном коде. Для расчетов удобно использовать Бейсик: PRINT OCT\$(-&O34/2) . На экран выводится восьмеричное число 177762. Так как смещение занимает только младший байт кода команды, то искомое значение смещения равно выражению: 177762-177400=362. Тогда полный код команды равен 000762 (сумме кода команды 000400 и смещения 362).

Команда 001000. Мнемокод: BNE

По команде BNE переход произойдет, если разряд Z PSW сброшен в "0".

Пример:

Адрес	Команда	Текст программы на ассемблере
1000:	020104	CMP R1,R4
1002:	001001	BNE MET
1004:	010102	MOV R1,R2

1006: 010103

МЕТ: MOV R1,R3

В этом примере первая команда сравнивает содержимое регистра R1 с содержимым регистра R4. В результате такого сравнения разряд Z PSW установится в "1" (если содержимые регистров R1 и R4 равны) или сбросится в "0" (если содержимые регистров не равны). Если содержимые регистров не равны (если разряд Z PSW сброшен в "0"), то вторая команда 001001 передает управление на адрес, где хранится 4-ая команда 010103, минуя 3-ю команду 010102. В противном случае (если разряд Z PSW установлен в "1") естественный порядок выполнения программы не нарушается – все 4 команды выполняются одна за другой.

Команда 001400. Мнемокод: BEQ

Команда BEQ является обратной по отношению к команде BNE. Переход по этой команде произойдет, если разряд Z PSW установлен в "1".

Команда 100000. Мнемокод: BPL

По команде BPL произойдет переход, если разряд N PSW сброшен в "0".
Пример:

Адрес	Команда	Текст программы на ассемблере
1000:	005710	TST (R0)
1002:	100002	BPL PL
1004:	005210	INC (R0)
1006:	004001	BR MIN
1010:	005310	PL: DEC (R0)
1012:	011002	MIN: MOV (R0),R2

Команда 005710 (TST) тестирует (проверяет) содержимое слова памяти, адрес которого хранится в R0. По команде 100002 произойдет переход, если при выполнении предыдущей команды (TST) разряд N PSW был сброшен в "0" (это произойдет, если тестируемый операнд больше или равен 0).

Команда 100400. Мнемокод: BMI

Команда является обратной по отношению к команде BPL. По команде BMI переход произойдет, если к моменту выполнения команды BMI разряд N PSW установлен в "1".

Команда 102000. Мнемокод: BVC

Переход по этой команде произойдет, если разряд V PSW сброшен в "0".

Команда 102400. Мнемокод: BVS

Переход по этой команде произойдет, если разряд V PSW установлен в "1".

Команда 002000. Мнемокод: BGE

Рассмотрим пример:

Адрес	Команда	Текст программы на ассемблере
5000:	010103	MOV R1,R3
5002:	020102	CMP R1,R2
5004:	002001	BGE M1
5006:	010203	MOV R2,R3
5010:	...	M1: ...

Команда 020102 (CMP) сравнивает содержимое регистров R1 и R2. По команде 002001 (BGE) произойдет переход, если содержимое регистра R1 оказалось больше или равно содержимому R2. Нетрудно догадаться, что в результате выполнения данной программы в R3 будет находиться максимальное значение из содержащихся в R1 и R2.

Команда 002400. Мнемокод: BLT

Если в предыдущем примере вместо команды BGE поставить команду BLT, то переход по команде BLT произойдет, если содержимое R1 окажется меньше содержимого R2, и в результате программа получит в R3 минимальное значение.

Команда 003000. Мнемокод: BGT

Если команда BGT следует за командой сравнения двух операндов, то переход произойдет, если операнд источника больше операнда приемника.

Команда 003400. Мнемокод: BLE

Если команда BLE следует за командой сравнения двух операндов, то переход произойдет, если операнд источника меньше или равен операнду приемника.

Команды BGE, BLT, BGT и BLE можно использовать для перехода после сравнения двух операндов, как чисел со знаком.

Команды BHI, BLOS, BHIS и BLO, приведенные в таблице 13, также можно использовать для перехода после сравнения двух операндов. Но в этом случае операнды рассматриваются не как числа со знаком, а как 16-разрядные целые числа без знака в диапазоне от 0 до 65535. Таким способом имеет смысл сравнивать адреса.

Ниже приведены соответствия между этими командами:

BGE - BHIS
BGT - BHI
BLT - BLO
BLE - BLOS

Команда 077R00. Мнемокод: SOB

Формат команды SOB изображен на рисунке 19.

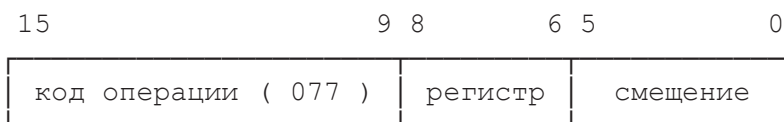


Рис.19. Формат команды SOB

Значение смещения занимает 6 младших разрядов кода команды и рассматривается как число без знака – так как переход по этой команде происходит только в обратном направлении (в сторону уменьшения адресов). Причем переход произойдет только в том случае, если содержимое регистра, указанного в коде команды, после вычитания из него 1 не равно 0. В коде команды 077R00 вместо буквы "R" указывается номер регистра, участвующего в операции. Код команды SOB равен 077R00 только в том случае, если значение смещения равно 0. Для получения кода команды со смещением, отличным от 0, к коду команды 077R00 прибавляется значение смещения, равное значению выражения

$$(AK + 2 - AP) / 2$$

где AK – адрес команды SOB;

АП – адрес перехода.

Значения адресов АК и АП задаются в восьмеричной системе счисления – поэтому все расчеты в указанном выражении ведутся по восьмеричной системе счисления. Например, $(1036+2-1034)/2=2$; $(1056+2-1002)/2=27$. Расчеты значения смещения по данной формуле можно производить, используя десятичные значения адресов АК и АП – но результат после этого должен быть представлен в восьмеричной системе счисления. Например, код команды без смещения равен 077300 и восьмеричное значение смещения равно 27, тогда код команды, содержащий необходимое значение смещения, будет равен 077327.

Пример:

Адрес	Команда	Текст программы на ассемблере
1000:	012700	MOV #1777,R0 ;длительность звука
1002:	001777	; (количество периодов)
1004:	012701	MOV #400,R1 ;длительность полупериода
1006:	000400	
1010:	012737	M3: MOV #100,@#177716 ;первый полупериод
1012:	000100	
1014:	177716	
1016:	010102	MOV R1,R2
1020:	000240	M1: NOP
1022:	077202	SOB R2,M1
1024:	012737	MOV #0,@#177716 ;второй полупериод
1026:	000000	
1030:	177716	
1032:	010102	MOV R1,R2
1034:	000240	M2: NOP
1036:	077202	SOB R2,M2
1040:	077015	SOB R0,M3

При выполнении этого фрагмента программы машина издает короткий звук (периодический сигнал прямоугольной формы). Команда NOP, примененная в этом примере, никаких операций не производит ("холостая" команда). Но так как команда NOP выполняется за определенный промежуток времени, то она здесь использована для задания временных задержек в программе. Комбинация двух команд: MOV #100,@#177716 и MOV #0,@#177716 издает звук. Тон звука зависит от временных задержек между исполнениями этих двух команд. Фрагменты по адресам 1004, 1016-1022 и 1032-1036 предназначены для задания временных задержек, определяющих тон звука. А команды MOV #1777,R0 и SOB R0,M3 задают длительность звука.

Команда 0001DD. Мнемокод: JMP

Команда JMP выполняет такие же действия, что и команда BR. Но если применение команды BR ограничено диапазоном смещений, то команду JMP можно использовать для перехода (передачи управления) на любой адрес программы. Поле адресации "DD" в коде команды задает не адрес операнда (поскольку операнда как такового в этой команде нет), а адрес, с которого будет продолжено выполнение программы после исполнения команды JMP. Поэтому в команде JMP недопустимо использование прямого регистрового способа адресации, так как передача управления на регистр процессора не имеет смысла.

Пример1:

000137	JMP @#7000
007000	

После выполнения этой команды программа продолжит свою работу с адреса 7000 – управление будет передано на адрес 7000.

Пример 2:

Адрес	Команда	Текст на ассемблере
5000:	000137	JMP @#MET
5002:	007554	
...
7554:	005001	MET: CLR R1

Пример 3:

Адрес	Команда	Текст на ассемблере
5000:	000167	JMP MET
5002:	002550	
...
7554:	005001	MET: CLR R1

Результаты выполнения приведенных фрагментов программ одинаковы. Но в первом случае используется абсолютная адресация, а во втором – относительная. Если, например, весь участок программы с адреса 5000 по 7554 переместить в другое место ОЗУ, оператор JMP с относительной адресацией будет работать правильно, поскольку относительное смещение метки МЕТ от команды JMP не изменится. А оператор с абсолютной адресацией отправит программу все равно на адрес 7554, и перемещенная программа будет работать неправильно.

При работе с отладчиками типа GPOT, MIRAGE рекомендуем внутри каждого отлаженного куска программы использовать относительную адресацию, чтобы можно было весь этот кусок безбоязненно перемещать в памяти. А при работе на МИКРО годятся оба варианта, так как ассемблер при трансляции программы вычисляет адреса сам, а программист работает только с метками.

4.4.4.2. Команды для работы с подпрограммами

Их всего две:

JSR (Jump to Subroutine – перейти на подпрограмму) и

RTS (Return from Subroutine – возврат из подпрограммы).

Команда 004RDD. Мнемокод: JSR

Работа команды JSR напоминает работу оператора GOSUB в программе на языке Бейсик. По команде JSR адрес возврата (адрес команды, следующей за командой JSR) запоминается в регистре, номер которого указывается в коде команды 004RDD вместо буквы "R", а содержимое самого регистра процессора до этого запоминается в стеке (в специально отведенном месте ОЗУ). Если в коде команды указан номер регистра R7 (цифра "7"), то адрес возврата запоминается в стеке.

Назначение поля адресации "DD" в коде команды такое же, что и для команды JMP. Рекомендации по выбору режима адресации те же, что и для команды JMP – в программах (или участках программы), которые могут в процессе работы (или при отладке) перемещаться в ОЗУ, следует использовать относительную адресацию.

Команда 00020R. Мнемокод: RTS

Работа команды RTS напоминает работу оператора RETURN в программе на языке Бейсик. Команда RTS возвращает управление на адрес возврата, который по команде JSR был запомнен в регистре процессора. Номер этого регистра должен быть указан в коде данной команды 00020R вместо буквы "R". При выполнении команды содержимое указанного регистра загружается в счетчик команд, а сам регистр загружается значением, взятым из стека (обычно это то значение регистра, которое было запомнено при выполнении команды JSR). Если в коде команды указан регистр R7 (цифра "7"), то запомненный по команде JSR адрес возврата загружается в счетчик команд из стека. Пример:

Адрес:	Команда	Мнемоника
4000:	004737	JSR PC,@#7500
4002:	007500	
...
7500:	012700	MOV #14,R0
7502:	000014	
7504:	104016	EMT 16
7506:	000207	RTS PC

Команда JSR PC,@#7500 передает управление на подпрограмму, начинающуюся с адреса 7500. Возврат из подпрограммы производится командой RTS PC – выполнение основной программы продолжится с адреса 4004.

4.4.4.3. Команды прерываний

Таблица 14. Прерывания

адрес вектора прерывания	источник прерывания	тип прерывания
4	нечетный адрес команды или несуществующий адрес операнда;	аппаратное
	нажатие на клавишу "СТОП";	
	команда останова HALT	программное
10	несуществующая команда	программное
14	прерывание по Т-биту PSW и по команде BPT	программное
20	прерывание по команде IOT	программное
24	авария сетевого питания	аппаратное
30	команда EMT	программное
34	команда TRAP	программное
60	клавиатура	аппаратное
100	прерывание от внешнего таймера	аппаратное
274	клавиатура при нажатой клавише "AP2"	аппаратное
360	прерывание от приемника ИРПС	аппаратное
364	прерывание от передатчика ИРПС	аппаратное

В процессе выполнения какой-либо программы на ЭВМ часто могут возникать не зависящие от нее события – например, нажатие на клавишу. Желательно, чтобы процессор сразу "обратил внимание" на такие события, а не ждал, пока программа "соизвоит" проверить состояние внешнего устройства. Для этого предназначен механизм обработки прерываний. Он заключается в том, что при возникновении события выполнение программы временно прерывается и запускается программа обработки прерывания. Когда она выполнит все необходимые действия, выполнение основной программы продолжится. Источниками возникновения прерываний могут быть как сигналы внешних устройств (аппаратные прерывания), так и некоторые ситуации, возникающие при выполнении программы (программные прерывания). Некоторые из них перечислены в таблице 14.

Каждый источник прерывания имеет свой вектор прерывания. Это адрес ОЗУ, по которому записан адрес программы обработки данного прерывания. Обычно прерывания обрабатываются монитором БК, но можно написать и свою программу обработки какого-либо прерывания. Для того, чтобы привести ее в действие, нужно будет записать адрес ее первой команды в соответствующий вектор прерывания. Чтобы обеспечить возврат к прерванной программе, в конце программы обработки прерывания должна стоять команда RTI (Return from Interrupt – возврат из прерывания). Код команды RTI – 000002.

Команда RTT (код 000006) аналогична RTI, но предназначена для возврата из отладочного прерывания.

Существует также ряд команд, которые вызывают программные прерывания по соответствующим векторам. К ним относятся:

EMT (коды 104000–104377) – вызов системных подпрограмм;

TRAP (коды 104400–104777) – вызов подпрограмм исполняющей системы (например, Бейсика, Фокала);

IOT (код 000004) – прерывание для ввода-вывода (на БК практически не используется);

BPT (код 000003) – прерывание для отладки.

4.4.4.3.1. Аппаратные прерывания

Рассмотрим пример возникновения прерывания от внешнего устройства

(клавиатуры). Это, пожалуй, самый сложный и практически интересный случай прерывания. В момент нажатия клавиши устанавливается в 1 седьмой бит (бит готовности) в регистре состояния клавиатуры (по адресу 177760; см. п.п. 1.5.1, 1.5.2). Далее анализируется шестой бит (бит разрешения прерываний) того же регистра. Если он равен нулю (прерывания разрешены), то клавиатура выдает запрос на прерывание. Теперь дело за процессором. Он заканчивает выполнение текущей команды программы и анализирует бит Р в РСР. Если он равен нулю, то процессор начинает обработку прерывания.

Первым делом процессор помещает в стек текущее значение PSW, потом содержимое счетчика команд РС (а он, как Вы догадываетесь, показывает на следующую команду после только что выполненной). После этого процессор берет из соответствующего вектора прерывания (в данном случае из ячейки 60) адрес программы обработки прерывания и помещает его в РС, а из следующей по порядку ячейки 62 берет новое значение PSW и помещает его в РСР. Если в 62 ячейке было записано число 200, то теперь бит Р в РСР будет установлен, и новое прерывание от клавиатуры не будет обрабатываться до тех пор, пока не закончится обработка данного прерывания.

Стандартная программа обработки прерывания читает из регистра данных клавиатуры 177762 код нажатой клавиши (при этом сбросится бит готовности в регистре состояний клавиатуры). Также стандартная программа обработки издает звук, после чего с помощью команды RTI возвращает управление прерванной программе. По команде RTI процессор восстанавливает из стека сохраненное там содержимое РС и PSW, и программа продолжает свою работу.

Обратите внимание, что в БК маскируются (то есть запрещаются установкой бита Р в PSW) только прерывания от клавиатуры. Возникновение прерывания по клавише "СТОП" запретить нельзя.

Пример 1. Организация перезапуска программы клавишей "СТОП". Здесь показано самое начало программы (все программы в кодах, как правило, начинаются с адреса 1000).

Адрес	Код команды	Текст программы на ассемблере
1000	012706	MOV #1000,SP ;установить стек;
1002	001000	;установить начальное
1004	104014	EMT 14 ;состояние экрана;
1006	012737	MOV #1000,@#4 ;теперь при нажатии
1010	001000	; "СТОП" программа будет
1012	000004	;запущена заново
1014	. . .	;продолжение программы

Пример 2. Запрещение ввода символов по клавише "AP2". Это полезный прием, поскольку к таким символам относятся клавиши переключения режимов экрана "БЛ.РЕД", "ИНВ.ЭКР", "РП" и т.п., которые могут испортить изображение при работающей программе.

; эту команду можно вставить в начале программы (после команд ; предыдущего примера)

```
MOV #TN,@#274 ;установить свою программу
. . . ;обработки прерываний по "AP2";
```

; а это – подпрограмма обработки прерывания по "AP2"

```
TN: TST @#177762 ;снять готовность клавиатуры;
RTI ;выйти из прерывания
```

Пример 3. Работа с клавиатурой в режиме опроса регистра состояния. Работа с внешними устройствами (например, клавиатурой) может быть организована как с использованием прерываний (при этом достигается максимально быстрая реакция программы на событие), так и без использования прерываний – в режиме опроса регистра состояний. Последний способ применяют тогда, когда программа должна реагировать

на события (например, на нажатие клавиш) только в определенные моменты. В игровых программах обычно имеет смысл управление движущимися объектами делать по прерываниям, а моменты диалога типа "вопрос-ответ" – в режиме опроса регистра состояния. В этом случае имеет смысл запретить стандартный механизм обработки прерываний, чтобы нажимаемые клавиши не портили картинку на экране; а перед ожиданием нажатия нужной клавиши "очистить входной буфер" клавиатуры, чтобы избежать неприятных последствий дребезга (повторного срабатывания) этого ненадежного устройства. Вот как это делается:

```
;начало программы
      MOV      #100,@#177760      ;запретить прерывания
      . . .                      ;от клавиатуры
;ожидание ввода символа с предварительной очисткой буфера ввода
      TST      @#177762          ;снять готовность клавиатуры;
M:     TSTB     @#177760          ;ожидание ввода
      BPL      M                ;символа;
      MOVB     @#177762,R0       ;принять код введенного символа
      . . .
```

В этом примере командой TSTB @#177760 проверяется 7-й бит в регистре состояния клавиатуры (бит готовности). Используется тот факт, что 7-й бит для байта является знаковым, поэтому удобно использовать команду BPL (или BMI).

Пример 4. Работа с клавиатурой по прерываниям.

```
;начало программы
      MOV      #TI,@#60          ;задать адрес своей программы
      . . .                      ;обработки прерываний;
      MOV      #200,@#62        ;запретить прерывания на время
      . . .                      ;работы своей программы обработки
;все действия производятся в программе обработки прерывания
TI:    MOVB     @#177762,R0      ;принять код нажатой клавиши;
      . . .                      ;сделать то, что нужно;
      RTI      ;выйти из прерывания
```

4.4.4.3.2. Программные прерывания

Из всех перечисленных программных прерываний наибольшую практическую ценность для программиста-любителя представляет прерывание по команде EMT. Процессор обрабатывает эту команду таким же образом, как если бы некое внешнее устройство выдало запрос на прерывание по вектору 30. Фактически же эта команда придумана специально, как кратчайший способ вызова системных подпрограмм. Дело в том, что команда EMT выполняется процессором одинаково независимо от содержимого младшего байта команды. Программа обработки прерывания (ее называют EMT-диспетчером) использует младший байт команды EMT как номер подпрограммы из своей системной таблицы и вызывает ее. Свой набор EMT-команд характерен для каждой операционной системы (напомним, что функции операционной системы в БК выполняет зашитый в ПЗУ МОНИТОР).

Выгода при использовании EMT-команд такова: команда EMT вместе с номером вызываемой подпрограммы занимает одно слово, а обычный способ вызова подпрограмм командой "JSR PC,адрес" требует двух слов памяти.

Монитор БК обеспечивает выполнение следующих EMT-команд:

```
EMT 4 – инициализация векторов прерывания клавиатуры;
EMT 6 – чтение кода символа с клавиатуры (выходной параметр –
код нажатой клавиши в R0);
EMT 10 – чтение строки с клавиатуры. Входные параметры:
```

R1 - адрес буфера, куда вводить строку;
 R2 - максимальная длина строки в младшем байте,
 символ-ограничитель в старшем байте.

После записи в память очередного введенного символа содержимое R1 увеличивается на 1, а после окончания ввода в R2 будет разность между максимальной длиной и длиной введенной строки. Ввод строки прекращается, если длина строки станет равна максимальной, либо введен символ, указанный в старшем байте R2. Пример:

```

012701      MOV    #2000,R1 ;расположить строку с адреса 2000;
002000                                ;макс.длина строки = 100 байт,
012702      MOV    #5100,R2 ;строка может быть закончена
005100                                ;клавишей "BK" (код 12 в старшем байте);
104010      EMT    10      ;ввести строку;
005702      TST    R2      ;до конца ли введена строка?
001404      BEQ    M
005301      DEC    R1      ;уберем код 12 из введенной строки
005202      INC    R2
112721  C:  MOVB   #40,(R1)+ ;и заполним остаток строки
000040                                ;пробелами
077203      SOB    R2,C
M:  . . .
  
```

EMT 12 - установка ключей K1-K10 клавиатуры;
 вход: R0 - номер ключа от 1 до 10;
 R1 - адрес текста ключа, текст должен кончатся

нулевым байтом;

EMT 14 - инициализация экрана и установка всех векторов прерывания;

EMT 16 - вывод символа; вход: код символа в R0;

EMT 20 - вывод строки;

вход: R1 - адрес строки;

R2 - длина строки в младшем байте;

символ-ограничитель в старшем байте;

EMT 22 - вывод символа в служебную строку;

вход: R0 - код символа (0 - очистка строки);

R1 - номер позиции в служебной строке;

EMT 24 - установка курсора по координатам X = R1, Y = R2;

EMT 26 - получение координат курсора: R1 = X, R2 = Y;

EMT 30 - рисование точки по координатам X = R1, Y = R2;

R0 = 1 - запись точки, R0 = 0 - стирание;

EMT 32 - рисование вектора (входные параметры те же, что и в EMT 30).

Пример:

;привести экран в исходное состояние

```
104014      EMT    14
```

;установить режим 32 символа в строке

```
012700      MOV    #233,R0
```

```
000233
```

```
104016      EMT    16
```

;нарисовать точку текущим (красным) цветом по координатам (0,0)

```
012700      MOV    #1,R0
```

```
000001
```

```
005001      CLR    R1
```

```
005002      CLR    R2
```

```
104030      EMT    30
```

;нарисовать зеленым цветом вектор из (0,0) в (111,111)

```
012737      MOV    #125252,@#214 ;занести код зеленого цвета
```

```
125252                                ;в служебную ячейку цвета
```

```

000214
012701      MOV      #111,R1
000111
010102      MOV      R1,R2
104032      EMT      32

```

ЕМТ 34 – получение в R0 слова состояния дисплея, в котором каждый разряд является индикатором включения соответствующего режима (табл.15): 0 – выключено, 1 – включено;

Таблица 15. Слово состояния дисплея

номер разряда	соответствующий режим
0	Режим "32 символа в строке"
1	Инверсия экрана
2	Режим расширенной памяти (РП)
3	Русский регистр
4	Подчеркивание символа
5	Инверсия символа
6	Индикация "СУ"
7	Блокировка редактирования
8	Режим текстовой графики "ГРАФ"
9	Запись в режиме "ГРАФ"
10	Стирание в режиме "ГРАФ"
11	Режим "32 символа в служебной строке"
12	Подчеркивание символа в служебной строке
13	Инверсия символа в служебной строке
14	Гашение курсора
15	Не используется

ЕМТ 36 – работа с магнитофоном; в R1 задается адрес блока параметров (обычно блок параметров задается с адреса 320). Формат блока параметров показан в таблице 16.

Если при загрузке задать адрес, равный нулю, то программа (массив) будет загружаться с адреса, указанного на ленте. После успешной загрузки файла адрес загрузки и длина массива записываются также по адресам 264, 266.

Таблица 16. Формат блока параметров для работы с магнитофоном

номер байта	адрес	содержание
0	320	команда: 0 – стоп двигателя, 1 – пуск 2 – запись массива на ленту 3 – чтение массива с ленты 4 – фиктивное чтение (поиск)
1	321	код завершения операции: 0 – без ошибок 1 – не то имя массива 2 – ошибка контрольной суммы 3 – останов по клавише "СТОП"
2	322	адрес массива в ОЗУ (слово)
4	324	длина массива на запись (слово)
6	326	имя массива (16. байт)

22	346	адрес обнаруженного на ленте массива
24	350	длина обнаруженного на ленте массива
26	352	имя обнаруженного на ленте массива (16. байт)

Аналогично EMT действует и команда TRAP, только по другому вектору. Эту команду используют обычно исполняющие системы интерпретаторов (Бейсик, Фокал и пр.) для своих внутренних целей. Например, в Фокале команда TRAP с нечетным кодом в младшем байте выдает соответствующий этому коду текст сообщения об ошибке, а с четным - вызывает соответствующую подпрограмму.

Программист может написать сам TRAP-диспетчер для своей программы, а может с разными целями перехватывать и системные командные прерывания. Например, первые Фокоды брали на себя вектор прерывания по EMT. После этого появлялась возможность вызывать подпрограммы в кодах из программы на Фокале при выводе на дисплей "непечатных" символов (0, 1, 2 и т.д.). При выводе символа Фокал выполняет команду "EMT 16", Фокодовская программа обработки прерывания проверяет, какая EMT-команда вызвала прерывание, и если ее номер 16 и выводимый код символа меньше 7, например, то вызывается подпрограмма в кодах; а иначе управление возвращается в системный EMT-диспетчер. Современные расширители Фокала и Бейсика используют вектор прерывания по TRAP, анализируют адрес команды во "внутренностях" Фокала (Бейсика), которая вызвала это прерывание и, при необходимости, сами выполняют аналогичные вызываемым подпрограммы, но по-другому - так, как нужно данному расширителю.

Приведем пример простейшего TRAP-диспетчера, который Вы можете использовать в своих программах для упрощения вывода символов. Для его понимания напомним, что выполняются командные (программные) прерывания так же, как и аппаратные - сначала в стеке сохраняется текущее состояние PSW, затем - адрес следующей команды, куда после обработки прерывания необходимо вернуться.

```
; в начале программы установим свой вектор прерывания по TRAP
MOV    #TRA,@#34
; теперь в программе можно использовать команду TRAP
TRAP   101 ;вывести символ "A" (его код равен 101)
TRAP   60  ;вывести символ "0" (код 60)
. . . .
; наша программа обработки прерывания по TRAP
TRA: MOV    R0,-(SP) ;сохраним регистр в стеке
MOV     2(SP),R0 ;получим из стека адрес возврата, который
                ;сохранен там командой TRAP
MOV     -(R0),R0 ;поместить в R0 саму команду - виновницу
                ;прерывания
EMT     16        ;вывести на экран символ из младшего байта
MOV     (SP)+,R0 ;восстановить R0
RTI          ;выйти из прерывания
```

Обратите внимание: обычно для вывода символа нужно было использовать две команды, занимающие три слова:

```
MOV     #101,R0
EMT     16
```

а теперь достаточно одной однословной команды "TRAP 101". При частом применении такой конструкции можно получить существенную экономию памяти.

Заметим также, что часто используется и прерывание по вектору 10 - недопустимая команда. Обычно это прерывание используется программами, эмулирующими команды умножения, деления и плавающей арифметики (поскольку у процессора БК нет таких команд).

4.4.5. Безоперандные команды

Таблица 17. Безоперандные команды

восьмеричный код команды	мнемокод команды	действие команды
000000	HALT	Halt - останов
000001	WAIT	Wait - ждать
000005	RESET	Reset - сброс
000240	NOP	No OPeration - нет операции
000241	CLC	Clear C - очистить C (разряд PSW)
000242	CLV	Clear V - очистить V
000244	CLZ	Clear Z - очистить Z
000250	CLN	Clear N - очистить N
000257	CCC	Clear Condition Code - очистить коды условий
000261	SEC	Set C - установить C
000262	SEV	Set V - установить V
000264	SEZ	Set Z - установить Z
000270	SEN	Set N - установить N
000277	SCC	Set Condition Code- установить коды условий

Здесь будут рассмотрены команды управления машиной и команды установки разрядов PSW. Все эти команды приведены в таблице 17.

4.4.5.1. Команды управления машиной

Команда 000000. Мнемокод: HALT

Вызывает прерывание по вектору 4 аналогично клавише "СТОП".

Команда 000001. Мнемокод: WAIT

По этой команде процессор временно прекращает выполнение программы и переходит в режим ожидания прерывания. После того, как произойдет прерывание от внешнего устройства (например, от клавиатуры), процессор обработает прерывание, и выполнение программы продолжится с команды, следующей за командой WAIT.

Команда 000005. Мнемокод: RESET

По этой команде все внешние устройства устанавливаются в состояние, которое они имеют после включения питания, после чего процессор возобновляет работу.

Команда 000240. Мнемокод: NOP

По этой команде процессор не выполняет никаких действий и переходит к выполнению следующей команды. Эта команда может использоваться при отладке программы в кодах для замены нескольких удаляемых команд, а

также для создания временных задержек при работе программы.

4.4.5.2. Команды установки разрядов PSW

Эти команды предназначены для установки в "1" или очистки (сброса в "0") отдельных разрядов PSW.

В мнемокоде каждой из 4-х команд CLC, CLV, CLZ, CLN первые 2 буквы "CL" указывают на то, что команда производит очистку одного разряда PSW. 3-я буква указывает на очищаемый разряд. Например, команда CLZ очищает разряд Z PSW. Команда CCC одновременно очищает разряды C, V, Z, N PSW.

Первые 2 буквы "SE" в мнемокодах команд SEC, SEV, SEZ, SEN указывают на то, что каждая из этих команд предназначена для установки в "1" отдельного разряда PSW, заданного 3-ей буквой мнемокода. Команда SCC одновременно устанавливает в "1" разряды C, V, Z, N PSW.

4.5. Использование стека

Стеком может служить любая свободная область ОЗУ.

Под стеком понимается область ОЗУ, адресация к ячейкам которой осуществляется определенным образом. Стандартный способ работы со стеком, который используют и все команды прерываний и подпрограмм, осуществляется с помощью регистра SP (Stack Pointer - указатель стека).

Стек используется программистом для временного хранения промежуточных данных программы. При записи слова в стек используется автодекрементный способ адресации, а при извлечении из стека данных - автоинкрементный.

Процессор использует стек для временного хранения адреса возврата из подпрограммы (или содержимого какого-либо регистра), а также при обработке прерывания. При этом запись в стек и считывание из стека производятся процессором аппаратно.

Перед выполнением команд, использующих стек, в регистр SP процессора необходимо предварительно записать адрес начала стека (настроить стек). Например, транслятор с языка Бейсик, зашитый в ПЗУ, использует стек, начинающийся с адреса 2000, а все обычные программы в кодах - с адреса 1000.

На рисунке 20 показано выполнение записи в стек по команде "MOV R3, -(SP)". Пусть до выполнения команды MOV регистр SP содержит адрес 1754, а в R3 было записано число 132. Затем указатель стека уменьшается на 2 и содержимое регистра R3 (число 132) записывается по адресу 1752.

С увеличением размеров стека уменьшается содержимое регистра SP - стек растет в сторону младших адресов.

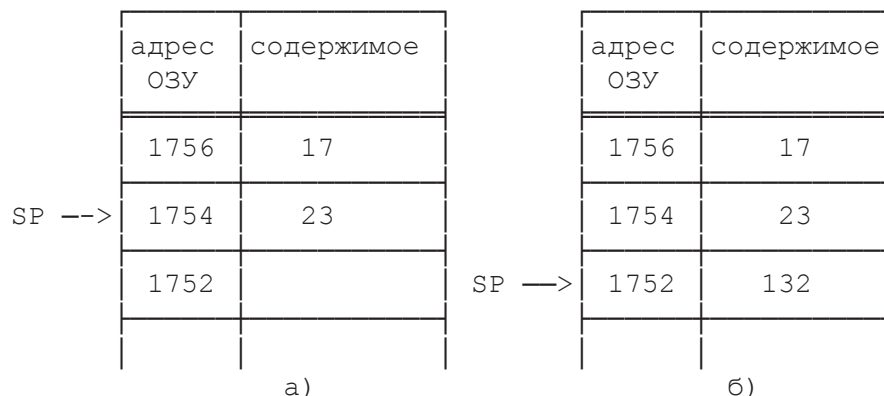


Рис. 20. Запись в стек: а) до выполнения команды; б) после выполнения команды

На рисунке 21 приведен пример чтения из стека. До выполнения

команды "MOV (SP)+,R3" регистр SP содержит адрес 1752. По указанной команде из слова, адрес которого хранится в SP, число 132 пересылается в регистр R3, после чего содержимое регистра SP увеличивается на 2.

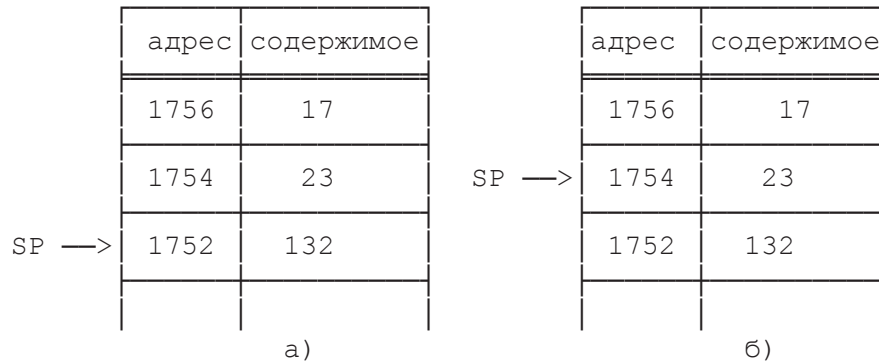


Рис. 21. Чтение из стека: а) до выполнения команды; б) после выполнения команды

4.6. ОЗУ экрана

ОЗУ экрана (видеопамять) занимает область памяти с адреса 40000 по адрес 77777. На рисунке 22 показано соответствие адресов видеопамяти определенным участкам на экране.

Например, левому верхнему углу экрана соответствует адрес 40000 видеопамяти, а нижнему правому углу – адрес 77777.

Для того, чтобы вывести какое-либо изображение на определенный участок экрана, достаточно записать по соответствующему адресу

40000	40001	40002	40003	...	40076	40077
40100	40101	40102	40103	...	40176	40177
...					...	
77600	77601	77602	77603	...	77676	77677
77700	77701	77702	77703	...	77776	77777

Рис. 22. Соответствие адресов видеопамяти определенным участкам на экране.

видеопамяти число, определяющее вид данного изображения. Например, чтобы на участке экрана, соответствующем адресу 56036, высветить точки, расположение которых показано на рисунке 23, достаточно переслать по адресу 56036 двоичный байт 10010011 (восьмеричное число 223). На языке ассемблера такая пересылка выражается командой

```
MOVB #223,@#56036
```





красная		черная		синяя		зеленая		← расположение точек при цветном изображении
								← расположение точек при черно-белом изображении
1	1	0	0	1	0	0	1	← содержимое разрядов байта
0	1	2	3	4	5	6	7	← номера разрядов байта

Рис. 23. Пример изображения байта на экране

Разряды байта выводятся на экран (слева направо), начиная с младшего разряда двоичного числа, что доставляет некоторые неудобства программисту.

Все сказанное выше верно только в том случае, если изображение выводится на черно-белый экран - тогда единичным битам выводимого байта соответствуют белые точки на экране, а нулевым битам - черные. Если же изображение выводится на цветной экран, то в формировании каждой "цветной" точки участвуют по 2 двоичных разряда. Красную точку дает комбинация разрядов 11, зеленую - 01, синюю - 10, и черную - 00.

Из этого следует, что в одном байте можно запрограммировать информацию о 8-ми "черно-белых" точках или о 4-х "цветных" точках.

Изображение, занимающее полную площадь экрана состоит из 256Д точечных строк. В формировании одной строки участвуют 64Д байта видеопамати. Например, для формирования начальной (самой верхней) строки экрана выделена область памяти с адресами от 40000 до 40077.

Верхняя часть экрана с адреса 40000 по 41777 используется для формирования служебной строки. Точка с графическими координатами X=0, Y=0 находится в младших разрядах байта по адресу 42000.

Глава 5. ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ДЛЯ ПРОГРАММИРОВАНИЯ В МАШИННЫХ КОДАХ

5.1. Отладчик MIRAGE

Система MIRAGE С.Зильберштейна (г.Киров) - одно из самых удобных средств, облегчающих программирование в кодах на БК и отладку программы в кодах. В комплект поставки входит описание системы и две версии программы:

MIRAGE26 - загружается с адреса 26000 и позволяет работать с программами в адресах ОЗУ 1000-25777;

MIRAGE - загружается в экранное ОЗУ с адреса 66000 и позволяет работать с программами максимальной длины (вплоть до 37777). Платой за эту возможность служит невозможность отладки программ со спрайтовой графикой, поскольку они могут запортировать отладчик, находящийся в экранном ОЗУ.

Существует также версия этого отладчика, распространенная под названием OS0010F, приспособленная П.Эльтерманом (г.Москва) для работы с Фокалом (возможность перехода из Фокала в MIRAGE и обратно удобна при написании и отладке программ типа "Фокод").

MIRAGE работает в двух основных режимах: командном и экранном.

При начальном запуске система находится в командном режиме. Здесь можно давать команды работы с файлами, с памятью, команды редактирования и трассировки отлаживаемой программы. При описании команд используются обозначения "начало" и "конец" для обозначения начального и конечного адреса, "имя" - имя файла при работе с магнитофоном. Необязательные элементы заключены в квадратные скобки.

LOAD имя - загрузка файла с адреса, указанного на ленте;

LOAD имя/адрес - загрузка с указанного адреса;

SAVE имя/начало/конец - запись программы на магнитофон;

SAVE - запись программы на магнитофон с тем же именем, начальным и конечным адресом, что были указаны в последней выполненной команде SAVE;

SET имя - поиск файла на ленте;

DIR - просмотр названий файлов на ленте;

D адрес - пошаговый дамп содержимого ОЗУ с указанного адреса по словам (распечатка в восьмеричной системе счисления);

D адрес/B - то же по байтам.

В режиме дампа действуют команды: "BK" - продолжить дальше, "." - окончить дамп и выйти в командный режим.

DR - распечатка содержимого регистров процессора R0-R5, SP, PC и признаков PSW N,Z,V,C;

S адрес - пошаговый просмотр и изменение содержимого памяти, начиная с указанного адреса по словам;

S адрес/B - то же по байтам;

SR - пошаговый просмотр и изменение регистров процессора.

В этих режимах можно, нажимая "BK" или "стрелку вверх", просматривать содержимое памяти соответственно в сторону увеличения либо уменьшения адресов, а при необходимости внести изменения: набрать новое значение слова (байта) в восьмеричной системе счисления и нажать "BK" или "стрелку вверх". "." - конец пошагового просмотра и выход в командный режим. В режиме просмотра по байтам "S адрес/B" можно вводить строку символов, начинающуюся кавычками.

F начало/конец/код - заполнение указанного участка ОЗУ значением "код";

M начало/конец/адрес - побайтное копирование участка ОЗУ, заданного начальным и конечным адресом, в область, начиная с указанного адреса;

C начало/конец/адрес - пословное сравнение двух участков ОЗУ одинаковой длины: один задан начальным и конечным адресом, второй начинается с указанного адреса;

W начало/конец/слово[/маска] - поиск "слова" в указанном участке ОЗУ. Если задана "маска", то ведется поиск всех слов, которые совпадают с образцом поиска ("словом") только в тех битах, которые установлены в маске. Например:

W 1000/2000/123 - найти и указать адреса слов в диапазоне от 1000 до 2000, в которых записано значение 123;

W 1000/2000/177600/177600 - найти в указанном диапазоне все обращения к системным регистрам, то есть слова, значения которых лежат в диапазоне от 177600 до 177777. Учтите при этом, что MIRAGE не может в подобных случаях отличить адрес (то, что нам и надо) от данных (что мы вовсе не имели в виду, давая эту команду). В каждом конкретном случае разбираться приходится самому программисту.

W начало/конец/адрес[/маска]/R - поиск относительных ссылок на указанный адрес (или интервал адресов, определенный маской). Например, у нас есть кусок программы:

```
1000:      167      JMP      4000
1002:      2774
```

где применена относительная адресация в команде перехода. Если мы дадим в MIRAGE команду "W 1000/1010/4000/R", то получим сообщение:

```
001002:      004000
```

из которого ясно, что по адресу 1002 найдена относительная ссылка на адрес 4000.

U начало/конец - подсчитать контрольную сумму в указанном участке.

G [начало][/конец] - запуск программы с начального адреса (или с текущего содержимого PC, если "начало" не указано) до конечного адреса. Если "конец" не указан, то управление системе MIRAGE вернется по команде HALT в отлаживаемой программе либо по прерыванию ее по ошибке или по клавише "СТОП".

T [адрес] - пошаговая трассировка программы с указанного адреса (либо с текущего содержимого PC). В процессе трассировки распечатываются текущие значения регистров R0-R5, SP, PC и признаки PSW N,Z,V,C, затем очередная команда, на которую показывает PC. "BK" - продолжение трассировки, "." или "СТОП" - окончание и выход в командный режим.

R адрес - вызов подпрограммы с указанного адреса. MIRAGE вернется в командный режим по достижении команды возврата из подпрограммы RTS PC.

A начало[/конец] - перейти к экранному редактированию программы в

указанном диапазоне адресов; при этом на экране отображаются адреса и мнемоника команд;

ASM начало[/конец] – перейти к экранному редактированию программы в указанном диапазоне адресов; при этом на экране отображаются адреса, коды и мнемоника команд.

В режиме экранного редактирования MIRAGE дизассемблирует часть ОЗУ (то есть переводит коды команд в их ассемблерные мнемоники). Если в этом участке встречаются данные, то они также считаются командами и будут дизассемблированы. Естественно, есть такие коды, которые не соответствуют ни одной команде процессора; в этом случае пишется мнемоника WORD (слово) и содержимое этого слова. По обилию мнемоник WORD среди прочих правдоподобно выглядящих ассемблерных команд можно распознать массивы данных в программе.

MIRAGE для удобства программиста заменяет команды "JSR PC,адрес" на "CALL адрес" и "RTS PC" на "RET"; а для удобства ввода кода символа позволяет писать команды вида "MOV #'A,R0", что тут же переведет в "MOV #101,R0".

В режиме экранного редактирования можно редактировать мнемоники команд с помощью клавиш управления курсором; все сделанные в данной строке изменения будут приняты только после нажатия "BK" в момент, когда курсор стоит в этой строке. Редактор работает только в режиме 64 символа в строке; нельзя также допускать выхода курсора за пределы экрана !

В режиме экранного редактирования действуют клавиши:

"KT" – листание текста на экран вперед;

"CU/T" – листание на один экран назад (только после "KT");

"ШАГ" – возврат к первому листу (с адреса, указанного в команде ASM);

"BC" – вставка команды NOP под курсором (нужно всегда учитывать размер команды, которую Вы желаете вставить, и вставлять для этого столько NOP-ов, сколько слов содержит команда. Об этом надо помнить и при редактировании старой команды);

"CBP" – удалить команду под курсором;

"СТОП" – выход в командный режим.

Рассмотрим пример работы с системой MIRAGE. Попробуем набрать программу буквально из нескольких команд. Загрузим MIRAGE26 и запустим его. Очистим командой "F 1000/2000/0" участок ОЗУ с адреса 1000 до адреса 2000 и будем здесь набирать нашу программу. Для этого перейдем в режим экранного редактирования командой "ASM 1000" и увидим:

```
1000:      000000      HALT
1002:      000000      HALT
1004:      . . .
```

Это – наша, еще пустая, рабочая область. Никакого труда, вероятно, не составит набрать такую короткую программу (нажимая "BK" после ввода каждой команды):

```
1000:      010100      MOV      R1,R0
1002:      104016      EMT      16
1004:      000775      BR       1000
```

Предположим, мы хотим заменить первую команду на команду "MOV #101,R0". Учитывая, что эта команда займет два слова, а в нашем распоряжении только одно, надо поставить курсор на вторую строку и клавишей "BC" вставить пустую команду. Программа приобретет вид:

```
1000:      010100      MOV      R1,R0
1002:      000240      NOP
1004:      104016      EMT      16
1006:      000775      BR       1002
```


Теперь подведем курсор к первой строке, заменим "R1" на "#101" и нажмем "BK". Программа примет вид:

```

1000:      010100 000101      MOV      #101,R0
1004:      104016      EMT      16
1006:      000775      BR       1002

```

Мы видим, что команда "MOV #101,R0" заняла два слова памяти и "съела" при этом вставленную команду "NOP". Если бы мы ее не вставили, пропала бы следующая команда "EMT 16".

Теперь осталось внести последнее исправление. В команде перехода использована относительная адресация, и, сдвинутая со старого места в ОЗУ, команда перехода показывает уже не на тот адрес. Поэтому при вставке или удалении команд приходится вручную корректировать адреса в сдвинутых командах перехода. Это самое большое неудобство при работе с подобными отладчиками. Можно посоветовать всегда иметь перед собой текст программы на бумаге и в процессе работы вносить туда все необходимые изменения. Нужно также иметь отдельную таблицу, в которой перечислены адреса используемых Вами подпрограмм и глобальных переменных. Такая таблица очень поможет Вам в работе.

Еще один полезный совет заключается в правильном использовании режимов адресации. Отлаживая программу по частям или по отдельным подпрограммам, старайтесь, чтобы все переходы внутри этого кусочка использовали относительную адресацию. Тогда после отладки данной подпрограммы ее можно будет перемещать в ОЗУ, заботясь только об изменении ссылок на глобальные имена – на внешние подпрограммы и переменные, которые к данному кусочку программы не привязаны.

Теперь давайте попробуем запустить нашу программу. Для этого нажмем "СТОП" – MIRAGE26 перейдет в командный режим. Запустим программу командой "G 1000". И она заработает – начнет выводить на экран букву "А" (это ее код – 101). Чтобы остановить нашу зацикленную программу, нажмем "СТОП" и вновь окажемся в командном режиме.

5.2. Отладчики типа ГРОТ

Распространено много версий отладчиков под разными названиями (ОТЛ, ОТЛАДЧИКЗ, ГРОТ и др.), которые по существу представляют собой одно и то же. Коротко опишем версию, в командной строке которой появляется надпись "Микро-отладчик <V1.1 860702 Москва-Рига>".

По своему принципу действия этот отладчик аналогичен системе MIRAGE, но имеет несколько меньшие возможности и менее удобен в работе. Он не имеет экранного редактора и удобной функции вставки-удаления команд. На экране его отображается адрес, мнемоника команды, ее коды по словам и по байтам, а также содержимое байтов в символьном виде:

```

1000      MOV      #101,R0      012700      25.300      Ю
                                000101      0.101      А
1004      EMT      16          104016      210.16
1006      HALT          000000      0.0

```

В правом верхнем углу выводятся текущие значения регистров процессора и признаков N, Z, V, C.

Команды отладчика:

адресА – установить текущее значение адреса (как в ТС-отладчике);

К – выход в пусковой монитор;

, (запятая) – перемещение текущего адреса на команду вперед;

- (минус) – перемещение текущего адреса на команду назад;

I – вывести мнемонику команды по текущему адресу;

"BK" – ввести мнемонику команды и записать ее по текущему адресу;

nD - сделать дамп n байт памяти с текущего адреса;
nL - распечатать n команд с текущего адреса;
конецАдрес - скопировать область ОЗУ с текущего по конечный адрес в область, начинающуюся с указанного адреса;
R - вывод регистров процессора;
значениеRn - записать "значение" в регистр Rn;
O - очистить рабочие переменные отладчика;
адресG - запустить программу с указанного адреса; управление вернется отладчику по команде HALT или другому аварийному прерыванию, либо по установленной контрольной точке;
адресТ - установить контрольную точку - адрес команды, где программа должна остановиться при отладке;
Т - снятие контрольной точки;
адрес[- пошаговая трассировка программы;
адресУ - медленная автоматическая трассировка (1 команды в 5 сек.);
адрес] - ускоренная автоматическая трассировка (1 команда в сек.).
Если в этих командах не указать адрес, то выполнение начнется с команды, на которую указывает РС.
адресЧ - чтение программы с магнитофона;
адресЗдлина - запись программы на магнитофон.

5.3. Ассемблер МИКРО

5.3.1. Описание языка

Работа с помощью описанных отладчиков очень близка к программированию в машинных кодах. Отладчики представляют собой первую ступень механизации программирования в кодах - замену кодов машинных команд удобными для человека их мнемоническими обозначениями. Вся же работа по распределению памяти, учету адресов лежит по-прежнему на программисте.

Второй ступенью механизации программирования является программирование на языке низкого (машинного) уровня - языке ассемблера. Как и система команд, свой язык ассемблера характерен для каждого типа ЭВМ. Что же принципиально новое дает программисту язык ассемблера? Прежде всего, он позволяет использовать метки в тексте программы. Метки используются для указания адреса в операциях перехода и пересылки. В этом отношении они похожи на номера строк Бейсика или Фокала. Теперь программист избавлен от необходимости учета физических адресов памяти, он работает только с метками, а адрес получается при трансляции ассемблерной программы. Если Вы добавите или удалите несколько команд, то Вам не придется корректировать адреса в сдвинутых с места командах перехода. Просто текст программы надо оттранслировать заново.

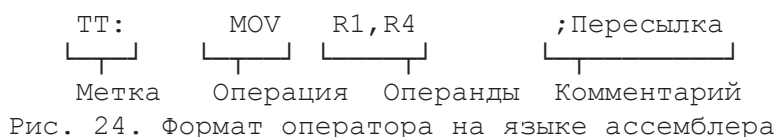
Для перевода программы на языке ассемблера в машинные коды используется специальная программа, называемая ассемблером. Процесс перевода называется ассемблированием. На ЭВМ ДВК, СМ используется мощный ассемблер MACRO, имеющий очень много облегчающих работу программиста средств (макрокоманды, условная трансляция и др.). На БК используются ассемблеры попроще. Наибольшее распространение получила система МИКРО, разработанная С.В.Шмытовым, А.Н.Сомовым и С.А.Кумандиным (г.Москва). Ассемблер МИКРО включает в себя редактор текста, транслятор и компоновщик, которые объединяет командный монитор. Распространено несколько версий системы: МИКРО8С, МИКРО9, МИКРО10, МИКРО11. Особенности каждой новой версии отражаются в соответствующем описании. Здесь приведены основные сведения по МИКРО9. Основные характеристики системы следующие:

- объем ОЗУ, занимаемый системой - 4.5К;
- максимальный объем исходного текста - 8К;
- максимальный размер транслируемой за один раз программы - 2К;
- максимальный размер программы, полученной при компоновке в режиме РП - 16.5К.

Работа с системой будет описана ниже, а пока познакомимся с языком ассемблера БК и его особенностями в системе МИКРО.

Программа на языке ассемблера, как и программа на Бейсике, состоит из строк (операторов). В одной строке может быть записана одна команда в обычной ассемблерной мнемонике, либо комментарий.

Строка программы (оператор) может состоять из 4-х частей (полей). На рис.24 приведен пример оператора на языке ассемблера с обозначением полей.



Поля "метка" и "комментарий" не являются обязательными частями оператора.

Имя метки состоит не более, чем из трех символов (букв латинского алфавита и цифр) и заканчивается двоеточием. В данном примере имя метки состоит из двух букв "ТТ". Меткой обычно "метится" тот оператор, к которому есть передача управления из какого-либо места программы. Для хранения значений переменных используются определенные слова памяти, которые также обычно "метятся" метками – тогда каждая метка является именем соответствующей переменной. Если имя метки начинается с цифры, то метка называется локальной, и ее действие распространяется только между обычными метками, поэтому имена локальных меток в программе могут повторяться. Двух одинаковых обычных меток в программе быть не должно.

Комментарии должны начинаться с символа ";", их можно располагать на отдельной строке или в конце строки оператора.

Поле "операция" содержит мнемокод машинной команды или псевдокоманду ассемблера МИКРО.

Псевдокоманды начинаются с символа "." (точка). В таблице 18 рассмотрены основные псевдокоманды, применяемые в ассемблере МИКРО9.

В конце текста программы на ассемблере должна стоять псевдокоманда "END".

Таблица 18. Основные псевдокоманды ассемблера МИКРО9

формат	выполняемые действия	пример
.+K	резервируется K байт	.+45
.#E	запись значения выражения E (в слово)	.#14
.@E	получение смещения. По адресу этой псевдокоманды записывается число, равное разности адреса метки E (E – имя метки) и адреса самой псевдокоманды	.@МЕТ
.E	здесь E – символ "E". Эта псевдокоманда обнуляет байт (если значение счетчика адресов нечетно) или слово (если значение счетчика адресов четно)	
.В:K	Записывает значение выражения K в текущий байт	.В:57
.А:...	Запись произвольной строки символов "..." с текущего адреса	.А:Привет

Поле "операнды" содержит информацию о местонахождении операндов, участвующих в операции.

Поле "операция" отделяется от поля "операнды" хотя бы одним

пробелом. Если команда двухоперандная, то операнд источника отделяется от операнда приемника запятой. Например:

```
MOV R1,R4
```

Ниже будут рассмотрены некоторые характерные для ассемблера примеры программирования.

```
Пример 1:   LF=12
             KBD=177662
             MOV #LF,R3           MOV #12,R3
             MOV KBD,R0          MOV @#177662,R0
```

Программы слева и справа дадут одинаковый результат.

Строка программы, имеющая вид <имя> = <число>, называется оператором прямого присваивания. Этот оператор похож на описание констант в языках высокого уровня, и вводится для удобства обозначения часто используемых констант или адресов (например, адресов системных регистров, как в данном примере). В данной версии ассемблера МИКРО есть ограничение: с помощью оператора присваивания можно работать только с адресами в диапазоне от 0 до 777 и от 100000 до 177777, при использовании таких имен в программе будет обеспечен абсолютный метод адресации. Этим МИКРО отличается от ассемблера MACRO для ЭВМ ДВК и СМ, где последнюю команду для получения абсолютной адресации следовало бы записать как "MOV @#KBD,R0".

Пример 2:

Адрес	Команда	Текст на ассемблере
6000:	000167	JMP VAR+2
6002:	001004	
...
7006:	005000	VAR: CLR R0
7010:	010203	MOV R2,R3

Команда JMP передает управление на адрес, равный значению выражения VAR+2=7006+2=7010.

Выражение в МИКРО9 – это имя метки и восьмеричное число, соединенные знаком "+" или "-". В частном случае выражение – это имя метки или восьмеричное число.

Пример 3:

Адрес	Команда	Текст на ассемблере
1000:	012700	MOV #6,R0
1002:	000006	
1004:	005060	A: CLRB TAB-1(R0)
1006:	001777	
1010:	077003	SOB R0,A
		...
2000:		TAB: .+6

В этом примере описан массив данных размером 6 байт, который размещается с адреса 2000. Программа очищает (заполняет нулями) этот массив. Для работы с массивом используется индексный метод адресации. Поскольку индекс – R0 меняется от 6 до 1, в качестве смещения используется выражение "TAB-1"; при значении R0=1 команда "CLRB TAB-1(R0)" очистит байт по адресу 2000).

Пример 4:

Адрес	Команда	Текст на ассемблере
1000:	010701	MOV PC,R1
1002:	062701	ADD (PC)+,R1

```

1004:      000012      .@T+2
1006:      005002      CLR  R2
1010:      104020      EMT  20
1012:      000000      HALT
1014:              T:   .A:Привет !
                          .E

```

Этот пример показывает, как на МИКРО можно писать перемещаемые программы. Коды этой программы можно записать с любого адреса ОЗУ, и она везде будет работать одинаково, так как адрес выводимой на дисплей строки "Привет !" вычисляется относительно текущего значения PC после выполнения команды "MOV PC,R1": $1002 + 12 = 1014$, где .@T = 10 – смещение от адреса этой псевдокоманды до начала текста (метки T), а еще 2 добавлено для учета самой команды "ADD (PC)+,R1".

А вот так предлагается организовать подпрограмму для вывода текста в описании МИКРО#10S (программа также перемещаемая). Предлагаем читателю самостоятельно разобраться в ее работе, напомним только, что после вызова подпрограммы командой "JSR PC,..." в стеке (по @SP) сохраняется адрес следующей команды (адрес возврата):

```

JSR  PC,TEX ;вызов подпрограммы вывода текста и передача ей
.@T          ;смещения до текста в качестве параметра
HALT         ;продолжение основной программы
T:  .A:Привет ! ;строка текста
    .E          ;нулевой байт в конце строки и выравнивание
          ;адреса на границу слова
TEX: MOV  @SP,R1 ;получение адреса псевдокоманды .@T
      ADD  @R1,R1 ;получение абсолютного адреса строки T
      CLR  R2     ;вывод строки на дисплей
      EMT  20     ;
      ADD  #2,@SP ;установить адрес возврата на следующую за .@T
      RTS  PC     ;команду программы (в данном случае – HALT)

```

5.3.2. Работа с системой МИКРО

Для получения готовой программы в машинных кодах необходимо проделать следующие операции:

- набор исходного текста (текста программы на языке ассемблера) при помощи редактора текста;
- трансляция (перевод) исходного текста программы в промежуточную заготовку (объектный модуль) при помощи ассемблера (транслятора);
- компоновка (составление) из нескольких объектных модулей готовой к выполнению программы в машинных кодах (загрузочного модуля) при помощи программы, называемой компоновщиком.

Все три программы: редактор текста, ассемблер и компоновщик соединены в одной программе МИКРО9, которая загружается и запускается на выполнение с адреса 1000. Программа МИКРО9 может работать в одном из четырех режимов:

- редактирование текста программы;
- трансляция (ассемблирование);
- компоновка;
- командный режим.

Сразу же после запуска МИКРО9 работает в командном режиме, о чем свидетельствует символ "*". В командном режиме возможна подача одной из следующих команд:

```

LO - Загрузка исходного текста с магнитной ленты (МЛ);
LF - Подстыковка текста с МЛ к тексту, находящемуся в памяти;
ST - Запись текста на МЛ;
SF - Запись на МЛ части текста от начала до строки, в которой
      находился курсор при выходе из редактора текста;
SS - Запись на МЛ части текста от строки, в которой находился

```

курсор при выходе из редактора текста, до конца текста;

CO - трансляция текста с выводом его на экран;

CN - трансляция текста без вывода его на экран;

Примечание: после подачи команд CO и CN на экран выводится вопрос "E/N?", ответ на который служит заданием действий, выполняемых программой после возникновения ошибки:

E - прекратить трансляцию,

N - продолжить трансляцию;

SC - вход в редактор текста;

SA - запись на МЛ загрузочного модуля (готовой программы);

SL - запись на МЛ объектного модуля (промежуточной программы);

LI - компоновка объектного модуля, записанного на МЛ к модулю, находящемуся в памяти (при этом получится программа, загружаемая с адреса 1000);

LA - компоновка объектных модулей, записанных на МЛ, с указанием адреса последующей загрузки (эта команда применяется только для головного модуля, остальные модули компонуются с помощью команды "LI");

LL - компоновка объектного модуля сразу после трансляции (эта команда применяется для программ, состоящих из одного модуля; при этом получится программа, загружаемая с адреса 1000);

LS - компоновка объектного модуля сразу после трансляции, с указанием адреса последующей загрузки.

В некоторых версиях МИКРО предусмотрена команда запуска скомпонованной программы "RU" или "GO".

5.3.2.1. Работа в редакторе текста

Для набора текста программы на языке ассемблера в редакторе текста используются следующие клавиши:

- клавиши перемещения курсора;
- клавиши "сдвигка в строке" и "раздвигка в строке";
- клавиша удаления части строки, расположенной справа от курсора;
- "СУ/Т" - раздвигка строк программы;
- "ВС" - сдвигка строк программы;
- "СБР" - переход в конец текста;
- "СУ/Э" - переход в начало текста;
- "забой" - удаление символа, стоящего перед курсором;
- "СУ/Ъ" - выход из редактора;
- "КТ" - переход к строке с задаваемым номером.

Клавиши и особенности редакторов в различных версиях МИКРО отражаются в соответствующих описаниях.

После изменения текста строки, а также для ввода новой строки необходимо нажать клавишу "BK".

В процессе редактирования в левом верхнем углу экрана отображается количество оставшихся байт памяти, отведенных для текста.

5.3.2.2. Ассемблирование

Если в исходном тексте программы на языке ассемблера нет ошибок, то в результате трансляции образуется объектный модуль, иначе ассемблер выдает сообщение об ошибках в виде:

ОШИБКА NN В СТР. ММММММ

где NN - номер ошибки;

ММММММ - восьмеричный номер строки текста.

Ниже приведены номера ошибок и их описание:

- 1 - недопустимый символ в строке;

- 2 - неверный оператор;
- 3 - ошибка длины перехода по команде BR;
- 4 - недопустимое имя метки;
- 5 - недопустимый символ в поле операнда;
- 6 - неверное число операндов;
- 7 - нестандартное имя (там, где это необходимо);
- 10 - неопределенное имя метки в операторе "SOB";
- 11 - неверная псевдокоманда;
- 12 - ошибка индексной адресации;
- 20 - неправильная команда в командном режиме.

5.3.2.3. Компоновка загрузочного модуля

Если исходный текст программы на языке ассемблера невозможно набрать в один прием, то он набирается частями. Каждая часть исходного текста при трансляции образует отдельный объектный модуль, затем все объектные модули связываются (компонуются) компоновщиком в один загрузочный модуль - в программу в машинных кодах, готовую к выполнению.

Если загрузочный модуль компонуется из одного объектного модуля, то компоновку можно произвести сразу же после трансляции исходного текста подачей одной из следующих команд: LL или LS.

После успешной трансляции распечатывается таблица использованных в модуле меток, при этом в инверсном виде распечатываются метки, которые не определены в данном модуле, но используются в командах программы. Это могут быть глобальные метки (определенные в другом модуле); тогда необходима компоновка программы. Если Ваша программа состоит всего из одного модуля, появление инверсных меток в таблице будет свидетельствовать о Ваших опечатках.

Для компоновки нескольких модулей с магнитной ленты программу МИКРО9 нужно привести в исходное состояние командой "RS".

5.3.3. Пример программ на языке ассемблера

В первом примере приведен простейший генератор случайных чисел, который можно использовать в игровых программах. Для начальной установки генератора случайных чисел используется такой прием: программа крутится в цикле, вызывая каждый раз генерацию нового случайного числа, до тех пор, пока пользователь не нажмет любую клавишу. Поскольку подгадать нажатие клавиши под нужное количество циклов практически невозможно, то работа программы каждый раз будет начинаться при новом значении случайного числа. Во время прохождения очередного цикла можно производить какие-либо действия на экране (в данном примере случайным образом рисуются точки случайного цвета).

; Пример генератора случайных чисел

```

EMT      14          ;очищаем экран;
MOV      #INE,R1     ;приведем экран в порядок;
CLR      R2          ;
EMT      20          ;
MOV      #100,@#177660 ;запретим прерывания от клавиатуры;
TST      @#177662;снять готовность, если клавиша была нажата;
```

; Начальная установка генератора случайных чисел

```

ING:     MOV      #2,R0   ;получаем случайный цвет (число от 0 до 2);
         JSR      PC,NRN
         ADD      #221,R0 ;получаем код символа установки цвета;
         EMT      16      ;устанавливаем цвет;
         MOV      #377,R0 ;получаем случайное число от 0 до 377
         JSR      PC,NRN  ;и используем его в качестве
         MOV      R0,R1   ;координаты X для вывода точки;
         MOV      #357,R0 ;получаем случайное число от 0 до 357
```

```

JSR      PC, NRN      ;и используем его в качестве
MOV      R0, R2        ;координаты Y для вывода точки;
MOV      #1, R0        ;рисуем точку;
EMT      30            ;
TSTB     @#177660;если клавиша не нажата,
BPL      ING           ;продолжаем цикл;
TST      @#177662;снять запрос на прерывание;
CLR      @#177660;разрешить прерывания;
MOV      RAN, R0       ;напечатать полученное
JSR      PC, OUT       ;случайное число;
HALT                                ;далее можно продолжать программу
;-----
; Строка инициализации экрана (ставится режим 32 символа в строке,
; убирается курсор и служебная строка)
INE:      .B:233.B:232.B:224.B:236.B:221.B:0
;-----
; Подпрограмма получения случайного целого числа
FRN:      .#000327      ;это код команды "SWAB #RAN"
RAN:      .#0           ;здесь будет целое случайное число
          INCB          RAN      ;все эти операторы
          ROLB          FRN+3    ;предназначены
MM:      ADD          #0, RAN    ;для получения
          ADD          #3337, MM+2 ;псевдослучайного числа
          RTS          PC
;-----
; Получение в R0 случайного числа в диапазоне 0..R0
NRN:      JSR          PC, FRN   ;получаем новое случайное число;
          MOV          R1, -(SP) ;сохраняем регистры в стеке;
          MOV          R2, -(SP)
          MOV          RAN, R1  ;отсекаем старшие разряды
          MOV          #100000, R2 ;случайного числа,
1A:      BIC          R2, R1     ;
          ASR          R2        ;
          CMP          R1, R0    ;пока оно не войдет в заданный
          BHI          1A        ;диапазон;
          MOV          R1, R0    ;результат в R0;
          MOV          (SP)+, R2 ;восстанавливаем регистры;
          MOV          (SP)+, R1
          RTS          PC
;-----
; Подпрограмма печати содержимого R0 в десятичной форме
OUT:      TST          R0        ;если число отрицательное,
          BPL          1A        ;то
          NEG          R0        ;изменить его знак на "+",
          MOV          R0, -(SP)  ;сохранить R0 в стеке;
          MOV          #55, R0    ;вывести знак "-"
          EMT          16        ;на экран;
          MOV          (SP)+, R0  ;восстановить из стека R0
1A:      JSR          PC, OU1    ;вывести само число
          RTS          PC
; Рекурсивная подпрограмма печати R0 в десятичной системе счисления
OU1:      MOV          R0, -(SP) ;делим в цикле содержимое стека
          DEC          R1        ;на 10 и получаем там количество единиц
          CLR          R0        ;(то есть младший десятичный разряд
1A:      INC          R0        ;исходного содержимого R0),
          SUB          #12, @SP  ;а в R0 получаем количество десятков
          BGE          1A        ;в исходном числе.
          ADD          #12, @SP  ;
          DEC          R0        ;
          BEQ          2A        ;пока еще остаются в числе десятки
          JSR          PC, OU1   ;(т.е. старшие разряды), снова печатаем их,

```

```

;для чего выполняем рекурсивный вызов
;подпрограммы OU1
2A:    MOV        (SP)+,R0 ;когда все число переведено и все
;разряды сохранены в стеке,
;вытаскиваем из стека очередной
ADD    #60,R0    ;разряд и печатаем соответствующую
EMT     16        ;цифру
RTS     PC
END

```

В этом примере для получения случайного числа используется простая подпрограмма FRN. Псевдослучайная последовательность чисел, получаемая с помощью этой подпрограммы, образует достаточно равномерное распределение. Но все же определенная закономерность в этой последовательности ощущается ("случайные" точки на экране более плотно располагаются вдоль нескольких прямых). Случайное 16-разрядное число генерируется в ячейке с адресом (меткой) RAN.

Для получения числа в заданном диапазоне используется подпрограмма NRN. Сначала в R0 нужно записать верхнюю границу диапазона, потом вызвать NRN, и в R0 будет находиться случайное положительное число в из заданного диапазона.

Очень интересна подпрограмма печати числа в десятичной форме. Здесь используется особый прием программирования - рекурсия. С помощью рекурсивных процедур (то есть подпрограмм, которые вызывают сами себя) можно вычислять факториал (классический пример из всех учебников), закрашивать участки экрана и многое другое. В Фокале рекурсия допустима, в Бейсике - нет.

Подпрограмма OU1 сохраняет в стеке количество единиц в печатаемом числе, вызывает саму себя, чтобы напечатать старшие разряды числа, после чего берет положенное в стек количество единиц и печатает соответствующую цифру. Эта подпрограмма является достаточно универсальной - Вы можете задать любую систему счисления, для чего вместо константы 12 (10 в десятичной системе) записать любое другое число, и программа будет печатать число в заданной системе счисления.

Следующий пример показывает, как можно сделать вывод изображения (спрайта) на экран и управлять его движением.

Спрайтом называется прямоугольное изображение, его размеры обычно кратны знакоместу экрана. Для редактирования спрайтов удобно использовать графические редакторы, например, ГРЕД4 (О.Туйкин, Д.Баранов). Этот редактор позволяет рисовать изображения цветными точками и записывать их на магнитофон в виде файла следующего формата:

```

ширина спрайта в байтах (одно слово);
высота спрайта в точечных строках (одно слово);
далее по строкам пишется содержимое спрайта.

```

Предположим, мы нарисовали человечка (рис.25). Размер этого спрайта совпадает с размером одного широкого знакоместа: 2 байта в ширину и 12 точечных строк в высоту (числа восьмеричные). Соответственно, ширина спрайта в цветном режиме составляет 8 цветных точек. На рисунке буквами "К" обозначены красные точки, "З" - зеленые.

```

. . З З К . . .
. . . К К . . .
. . . К . . . К
. К К К К К К .
К . . К К . . .
. . . К К . . .
. . К . . К . .
. . К . . К . .
. К К . . К К .

```

Рис.25. Пример рисования спрайта

Если теперь записать этот спрайт на магнитофон, то в файле будет содержаться последовательность слов:

2,12,1640,1700,140300,37774,1703,1700,1700,6060,6060,36074,

где первые два слова - ширина спрайта в байтах и высота его в строках. Можно кодировать спрайт и вручную, заменяя каждую точку изображения двумя битами и складывая их в слова, но это менее удобно.

Теперь приступим к написанию программы:

```
;Пример вывода спрайта на экран и управления им с клавиатуры
      EMT      14              ;инициализация экрана
;---- вывод изображения (спрайта) на экран ----
BEG:   MOV     #MEN,R0        ;адрес спрайта;
      MOV     ADR,R1          ;адрес ОЗУ, куда будем выводить;
      MOV     (R0)+,R2        ;ширина спрайта в байтах;
      MOV     (R0)+,R3        ;высота спрайта в точечных строках;
A:     MOV     R2,R4          ;начало цикла по строкам;
      MOV     R1,R5          ;запоминаем адрес начала строки;
B:     MOVB    (R0)+,(R1)+    ;цикл вывода одной строки
      SOB     R4,B            ;по байтам;
      MOV     R5,R1          ;восстанавливаем адрес начала строки;
      ADD     #100,R1         ;переходим к следующей строке;
      SOB     R3,A            ;конец цикла по строкам.
;---- временная задержка для замедления движения спрайта ----
      MOV     #4000,R3        ;число 4000 задает длительность задержки
W:     SOB     R3,W
;---- очистка спрайта на экране -----
      MOV     #MEN,R0        ;адрес спрайта;
      MOV     ADR,R1          ;адрес ОЗУ, где будем стирать
      MOV     (R0)+,R2        ;ширина спрайта в байтах;
      MOV     (R0)+,R3        ;высота спрайта в точечных строках;
C:     MOV     R2,R4          ;далее все аналогично выводу спрайта,
      MOV     R1,R5
D:     CLRB    (R1)+          ;только здесь вместо пересылки -
      SOB     R4,D            ;стирание
      MOV     R5,R1
      ADD     #100,R1
      SOB     R3,C
;-----
      MOV     @#177662,R0     ;код нажатой клавиши в регистр R0
      CMP     R0,#10          ;нажата клавиша "курсор влево"?
      BNE     RIT             ;если нет, идти на метку RIT
      MOV     ADR,R2          ;иначе проверяется, не достигло
      BIC     #177700,R2      ;ли изображение левого края экрана
      BEQ     BEG             ;если достигло, идти на метку BEG
      DEC     ADR             ;иначе уменьшить на 1 адрес изображения
      BR      BEG             ;и идти на метку BEG - переместить изо-
                              ;бражение левее на один байт
;-----
RIT:   CMP     R0,#31          ;нажата клавиша "курсор вправо"?
      BNE     UP              ;если нет, идти на метку UP
      MOV     ADR,R2          ;иначе проверяется,
      BIC     #177700,R2      ;достигло ли изображение
      CMP     #76,R2          ;правого края экрана
      BEQ     BEG             ;если достигло, идти на метку BEG
      INC     ADR             ;иначе увеличить адрес изображения на 1
      BR      BEG             ;и идти на метку BEG - переместить изо-
                              ;бражение правее на один байт
;-----
UP:    CMP     R0,#32          ;нажата клавиша "курсор вверх"?
```

```

BNE     DON      ;если нет, идти на метку DON
CMP     ADR,#42100 ;изображение достигло верхнего
                        ;края экрана?

BLO     BEG      ;если достигло, идти на метку BEG
SUB     #200,ADR  ;иначе адрес изображения уменьшить на 200
BR      BEG      ;и идти на метку BEG (переместить
                        ;изображение на две строки выше)
;-----
DON:    CMP     R0,#33 ;нажата клавиша "курсор вниз"?
BNE     BEG      ;если нет, идти на метку BEG
CMP     ADR,#76500 ;иначе проверяется достигну ли
                        ;изображение нижнего края экрана
BHI     BEG      ;если достигну, идти на метку BEG
ADD     #200,ADR  ;иначе адрес изображения увеличить на 200
BR      BEG      ;и идти на метку BEG (переместить
                        ;изображение на две строки ниже)
;-----
ADR:    .#56036   ;текущий адрес начала изображения
;---- таблица изображения (спрайта) ----
MEN:    .#2.#12.#1640.#1700.#140300.#37774.#1703.#1700.#1700
        .#6060.#6060.#36074
END

```

В приведенной программе в переменной ADR (ячейке памяти, помеченной меткой ADR) хранится адрес, по которому рисуется спрайт в экранном ОЗУ. С этого адреса начинается вывод левого верхнего угла спрайта. Начальное значение (56036) определяет, что в первый раз изображение будет выведено около центра экрана.

Сначала спрайт выводится в экранное ОЗУ с заданного адреса, а затем, после истечения некоторой задержки, стирается с экрана, и программа анализирует код клавиши. Если нажата любая клавиша, кроме стрелок управления курсором, переменная ADR не изменяется, и спрайт вновь рисуется на прежнем месте. Если нажата стрелка, то ADR изменяется, и спрайт будет в следующий раз нарисован уже в другом месте экрана.

Глава 6. НЕСТАНДАРТНЫЕ ПРИЕМЫ ПРОГРАММИРОВАНИЯ

6.1. Использование подпрограмм в кодах при работе на Бейсике

Одним из приемов увеличения возможностей программы на Бейсике является использование подпрограмм в кодах. Правда, этот путь несколько половинчатый – почему бы сразу не писать всю программу на ассемблере? Поскольку Бейсик по-прежнему занимает много памяти, то и эффективность этого способа сомнительна. Использовать подпрограммы в кодах в Бейсике, пожалуй, целесообразно только при отсутствии описанных в гл.5 инструментальных средств, а также в тех случаях, когда Вы не хотите или не умеете писать программы в кодах, а улучшить свою программу на Бейсике хочется. В последнем случае можно использовать чьи-либо готовые подпрограммы в кодах.

Для подготовки подпрограммы в кодах с помощью Бейсика часто используется такой прием: коды команд записываются в операторе DATA, из которого затем в цикле читаются оператором READ и заносятся в память оператором POKE.

Для того, чтобы зарезервировать достаточно места в памяти для размещения подпрограмм, необходимо перед началом работы дать команду CLEAR в непосредственном режиме Бейсика:

CLEAR B,A

где

В - количество байт, отводимых под символьные переменные;
 А - адрес верхней границы памяти, отведенной для программы на Бейсике.

Например, если начальный адрес подпрограммы в кодах равен 30000, то можно ввести оператор CLEAR 250,&O30000 . После исполнения этого оператора транслятор с языка Бейсик использует только область ОЗУ, расположенную до адреса 30000. А с адреса 30000 могут быть расположены подпрограммы в машинных кодах.

Две команды Бейсика позволяют записывать и читать с магнитофона участки ОЗУ, в которых могут размещаться подпрограммы в кодах или таблицы изображений (спрайтов).

BSAVE"имя", начало, конец

По этой команде область ОЗУ с указанным начальным и конечным адресом записывается на магнитофон в виде файла.

Программа с МЛ может быть загружена в ОЗУ и запущена на выполнение командой Бейсика:

BLOAD"имя", R, адрес

Если в данной команде отсутствует первая запятая с буквой "R", то произойдет только загрузка программы в ОЗУ с указанного адреса.

Обычно, чтобы не возиться с загрузкой множества файлов, подпрограмму в кодах не записывают отдельно, а так и хранят в программе на Бейсике в операторе DATA. При выполнении программы коды переписываются в ОЗУ, потом задается адрес запуска подпрограммы и она вызывается в любом нужном месте программы на Бейсике.

Здесь будут рассмотрены два примера использования в программах на языке Бейсик подпрограмм в машинных кодах.

Пример 1. Вывод спрайта и управление его движением.

Этот пример аналогичен второму примеру из п.5.3.3. Для вывода и стирания спрайта используются подпрограммы в кодах, а управление движением спрайта осуществляется программой на Бейсике.

Подпрограммы вывода и стирания спрайта аналогичны приведенным в п.5.3.3:

```
; вывод спрайта на экран
035000      012700      MOV      #MEN,R0
035002      035070
035004      011501      MOV      (R5),R1      ;адрес вывода спрайта
035006      012002      MOV      (R0)+,R2
035010      012003      MOV      (R0)+,R3
035012      010204      A:      MOV      R2,R4
035014      010146      MOV      R1,-(SP)
035016      112021      B:      MOV      (R0)+,(R1)+
035020      077402      SOB      R4,B
035022      012601      MOV      (SP)+,R1
035024      062701      ADD      #100,R1
035026      000100
035030      077310      SOB      R3,A
035032      000207      RTS      PC      ;выход из подпрограммы
; чистка спрайта на экране
035034      012700      MOV      #MEN,R0
035036      035070
035040      011501      MOV      (R5),R1      ;адрес стирания спрайта
035042      012002      MOV      (R0)+,R2
035044      012003      MOV      (R0)+,R3
035046      010204      C:      MOV      R2,R4
035050      010146      MOV      R1,-(SP)
035052      105021      D:      CLRB      (R1)+
```



```

035054      077402      SOB      R4,D
035056      012601      MOV      (SP)+,R1
035060      062701      ADD      #100,R1
035062      000100
035064      077310      SOB      R3,C
035066      000207      RTS      PC          ;выход из подпрограммы
;спрайт, закодированный в формате ГРЕД
035070      MEN: .#2.#12.#1640.#1700.#140300.#37774.#1703
              .#1700.#1700.#6060.#6060
035116      .#36074

```

В отличие от примера п.5.3.3, где операции вывода и стирания спрайта были встроены в программу, здесь они оформлены в виде подпрограмм, которые можно вызывать функцией Бейсика `USR`. Выход из подпрограммы осуществляется, как обычно, командой `RTS PC`.

Одновременно в программе могут быть определены 10 различных функций `USR0..USR9`. В данном примере мы используем только две подпрограммы (соответственно, функции `USR0` и `USR1`).

Функция `USR` позволяет передать подпрограмме в кодах один параметр и вернуть одно значение того же типа. Функция `USR` помещает в `R5` адрес передаваемого параметра, а в `R3` - его тип (единица в 15-м разряде означает символьную строку, информация о других типах хранится в младшем байте `R3`: -1 - целый, 0 - вещественный двойной точности, 1 - вещественный одинарной точности; если аргумент - символьная строка, то в качестве аргумента передаются два слова - длина и адрес строки). В данном примере мы передаем подпрограммам адрес ОЗУ экрана, куда нужно вывести (или где стереть) спрайт. Поскольку адрес будет размещаться в переменной целого типа, анализировать `R3` нет нужды.

Отметим также, что в п.5.3.3 "по ходу действия" в `R5` сохранялось значение `R1`, здесь же мы `R1` сохраним в стеке, чтобы не испортить адрес аргумента.

Подпрограммы и закодированный спрайт мы разместим с адреса 35000. Тогда перед набором программы (или загрузкой ее с магнитной ленты) необходимо выполнение оператора `CLEAR ,&035000`. Этот оператор можно вставить первой строкой в программу, чтобы он выполнялся автоматически, но вот беда - не все экземпляры БК-0010-01 отработают его правильно. Это связано с различиями версий Бейсика в ПЗУ БК разных лет выпуска.

Для того, чтобы автоматизировать ввод подпрограмм, их коды мы разместим в операторе `DATA`, откуда потом в цикле будем читать и переписывать по адресам ОЗУ, начиная с 35000.

```

10 DATA &0012700,&0035070,&0011501,&0012002,&0012003,&0010204,&0010146
15 DATA &0112021,&0077402,&0012601,&0062701,&0100,&0077310,&0207
20 DATA &0012700,&0035070,&0011501,&0012002,&0012003,&0010204,&0010146
25 DATA &0105021,&0077402,&0012601,&0062701,&0100,&0077310,&0207
30 DATA 2,&012,&01640,&01700,&0140300,&037774,&01703
35 DATA &01700,&01700,&06060,&06060,&036074
40 ?CHR$(140);CHR$(140)          'инициализируем экран
50 FOR A%=&035000 TO &035116 STEP 2% 'записываем подпрограммы в ОЗУ
60 READ D%
70 POKE A%,D%
80 NEXT A%
90 DEF USR0=&035000      'задаем адрес подпрограммы вывода спрайта
95 DEF USR1=&035034      'задаем адрес подпрограммы стирания спрайта
100 S%=&056036           'начальный адрес ОЗУ, где располагается спрайт
110 L%=USR0(S%)          'выводим спрайт
115 FOR L%=0% TO 150%    'задержка
116 NEXT
117 L%=USR1(S%)          'стираем спрайт
120 I%=PEEK(&0177662)    'смотрим, какая клавиша была нажата

```

```

130 IF I%=&O10 THEN IF S% MOD 64% <> 0% THEN S%=S%-1% 'влево
140 IF I%=&O31 THEN IF S% MOD 64% < &O76 THEN S%=S%+1% 'вправо
150 IF I%=&O32 THEN IF S%>&O43000 THEN S%=S%-&O200 'вверх
160 IF I%=&O33 THEN IF S%<&O76000 THEN S%=S%+&O200 'вниз
170 GOTO 110

```

Строки 50 – 80 производят загрузку команд и данных в область памяти с адреса 35000 по 35116.

Оператор DEF (строки 90-95) определяет подпрограммы в машинных кодах под именем USR0 и USR1 и задает их начальный адрес.

Текущее значение адреса спрайта в ОЗУ экрана хранится в целой переменной S%.

В строке 120 переменной I% присваивается код нажатой клавиши, в зависимости от которого в строках 130 – 160 изменяется адрес спрайта S%.

Начальная установка экрана (строка 40) необходима для того, чтобы привести экран в исходное состояние, когда верхний левый угол экрана соответствует адресу 40000.

Пример 2:

```

10 DATA &O13701,&O35102,&O12737,&O100,&O177716,&O13700,&O35100
12 DATA &O77001,&O12737,0,&O177716,&O13700,&O35100,&O77001,&O77115
14 DATA &O12702,&O400,&O77201,&O207
20 FOR A%=&O35000 TO &O35044 STEP 2
30 READ B%
40 POKE A%,B%
50 NEXT A%
60 DEF USR0=&O35000
70 DATA 128,128,112,128,99,640,128,128,112,128,99,128,128,128
75 DATA 94,128,99,128,128,256,112,128,99,128,112,640
80 FOR C%=1 TO 13
90 READ T%,D%
100 POKE &O35100,T%
110 POKE &O35102,D%
120 I%=USR0(I%)
130 NEXT C%

```

Программа на Бейсике, приведенная в этом примере, использует подпрограмму в машинных кодах для получения мелодии.

Строки 10-60 подготавливают в памяти, начиная с адреса 35000, подпрограмму в машинных кодах.

Вся мелодия состоит из звуков. Количество звуков равно количеству пар чисел, перечисленных в строках 70-75. В каждой паре чисел первое число задает тональность, а второе – длительность звука.

Строки 80-130 "исполняют" мелодию.

6.2. Использование вещественной арифметики Бейсика при программировании в кодах

Вы уже знаете, что в системе команд процессора БК нет команд работы с вещественными (действительными) числами. Как в Бейсике, так и в Фокале работа с вещественными числами организуется чисто программно. В таком случае, почему бы не использовать записанные в ПЗУ БК подпрограммы, если надо производить какие-либо вычисления?

Использование подпрограмм вещественной арифметики Фокала описано в [15]. Здесь мы опишем использование арифметики Бейсика [8]. Сразу же заметим, что формат вещественного числа в Бейсике и Фокале различен.

Учтите также, что применение описываемого здесь метода сделает программу непереносимой – она уже не сможет работать на БК-0010 с Фокалом или на БК-0010-01 при включенном блоке МСТД (равно как и

программы, использующие Фокал, не будут работать с Бейсиком). Хорошим выходом было бы использование стандартных подпрограмм (из программного обеспечения Электроники-60) или эмуляторов вещественной арифметики, но это доступно, пожалуй, только профессиональным программистам, использующим ЭВМ более высокого класса в качестве инструментальной. Что же касается использования подпрограмм Бейсика, то платой за это будет 2К памяти, так как область ОЗУ ниже адреса 4000 используется Бейсиком для своих рабочих переменных и стека.

Все расчеты в Бейсике производятся с вещественными числами двойной точности. Если в программе на Бейсике используются числа одинарной точности, то перед вычислениями они все равно преобразуются к двойной. Каков же формат вещественного числа в Бейсике ?

Числа двойной точности занимают в памяти 4 слова (8 байт). При этом старший разряд 1-го слова (63-ий разряд числа) – знак числа, разряды 7-14 (разряды 55-62 числа) – порядок (p), а остальные 6 разрядов 1-го слова и 2-ое, 3-е, 4-е слова (разряды 0-54 числа) – мантисса (M) (рис.26).

Тогда значение числа N определяется по формуле:

$$N=(M+1)^2 = (1 + \frac{a_{54}}{2} + \frac{a_{53}}{2} + \frac{a_{52}}{2} + \dots + \frac{a_1}{2} + \frac{a_0}{2})^2, \quad \text{p-201}$$

где a_i - значение i -го бита числа (0 или 1).

Число 201 -восьмеричное, остальные - десятичные.

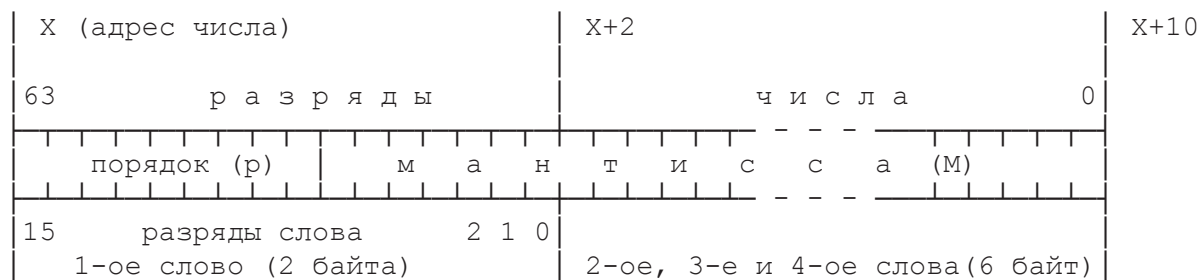


Рис.26. Представление числа двойной точности

Как же перевести число из привычного для нас десятичного вида в двоичное представление числа двойной точности ?

Для этого берется исходное число и делится на 2 до тех пор, пока его целая часть не будет равна 1.

Если исходное число меньше 1, то вместо деления производится умножение на 2 до тех пор, пока целая часть не станет равна 1. Затем дробная часть числа переводится в двоичный код – мантисса готова. 201 (восьмеричное) плюс (или минус, если производилось умножение) количество делений или умножений на 2 – получим порядок числа.

Осталось "скомпоновать" число: последовательно слева направо записываются знак (0 - для знака "+", 1 - для знака "-"), порядок, мантисса. Остальные биты мантиссы заполняются нулями или мантисса укорачивается справа так, чтобы общая длина числа равнялась 64 битам.

Рассмотрим несколько примеров перевода десятичных чисел в двоичные числа двойной точности:

Пример 1: Исходное число: 9.5 . Делим на 2:

9.5

4.75 - результат после 1-го деления:

2.375 - после 2-го деления;

1.1875 - после 3-го деления.

0.1875 - умножается на 2;
0.375 - умножается на 2;
0.75 - умножается на 2:
1.5 - отбрасывается 1 и умножается на 2:
1 - все.

`0 10000100 0011 00.`

Адреса:	Содержимое:
X	41030 (восьмеричное)
X+2	0
X+4	0
X+6	0

$$\begin{array}{r} 13.25 \\ 6.625 \\ 3.3125 \\ 1.65625 \end{array}$$

0.65625
1.3125
0.625
1.25
0.5
1

```
11000010010101000000000000000000000000000000000000000000000000
```

0.0234375
0.046875
0.09375
0.1875
0.375
0.75

1.5

Переводим дробную часть числа 1.5 равную 0.5 в двоичный код: это будет 1. Порядок равен $201-6=173$ (числа восьмеричные). Знак - 0 (" "). Вид числа в двоичном коде:

[illegible]

или четыре слова: 36700,0,0,0 в восьмеричном виде.

Как же теперь использовать наши умения ? Сначала надо поближе познакомиться с тем, как Бейсик транслирует свою программу.

Исходная программа на Бейсике после ее запуска командой RUN преобразуется в последовательность адресов подпрограмм, реализующих соответствующие операторы Бейсика, и исходных данных: констант, адресов переменных и так далее [8]. Эти подпрограммы, как и сам транслятор, делающий такое преобразование, защиты (записаны) в ПЗУ.

После того, как указанная последовательность (ее называют "шитым кодом") получена в ОЗУ, в регистр R4 помещается адрес начала последовательности, и командой "JMP @ (R4) + " начинается обращение к подпрограмме, адрес начала которой записан в указанной последовательности первым. Передача управления от одной подпрограммы к другой осуществляется также по команде "JMP @ (R4) + " , содержащейся в конце каждой подпрограммы.

Если в программе на языке ассемблера программисту необходимо получить значение какой-либо функции (или арифметической операции), то он может организовать обращение к готовым подпрограммам, зашитым в ПЗУ. Далее под словом "операция" подразумевается арифметическая операция или функция, а под словом "операнд" - операнд арифметической операции или аргумент функции.

В таблице 19 приводится список адресов подпрограмм, при обращении к которым вычисляется результат соответствующей функции (или арифметической операции).

Операнды и результаты операций – числа двойной точности.

Таблица 19. Адреса подпрограмм Бейсика

Арифметические операции и функции	Адреса подпрограмм	Количество операндов
+	167144	2
-	167124	2
/	170776	2
*	170210	2
SQR	171350	1
SIN	173614	1
COS	173566	1
TAN	174306	1
ATN	174434	1
EXP	171762	1
LOG	173052	1
FIX	176212	1
INT	176340	1
RND	175176	1
подпрограмма преобразо- вания результата в целый тип	166760	1

Необходимо помнить, что для нормальной работы подпрограмм, указанных в таблице 19, область памяти с адресами от 2000 до 4000

должна быть свободной (эта область - область системных переменных Бейсика).

Предлагаем Вашему вниманию небольшую подпрограмму, которая может облегчить Вам вызов нужных подпрограмм Бейсика.

Для вызова нужной подпрограммы в регистрах должны быть подготовлены следующие данные:

R0 - число операндов (1 или 2);

R1 - адрес первого операнда;

R2 - адрес второго операнда (для операций с двумя операндами.

При операциях с одним операндом значение несущественно);

R3 - адрес подпрограммы операции (из табл.19);

R5 - адрес результата.

Эта вспомогательная подпрограмма работает аналогично транслятору с Бейсика после команды RUN - формирует требуемый для вызова нужной подпрограммы шитый код и передает ему управление.

Последовательность адресов подпрограмм и данных (шитый код) начинается с адреса A.

```

FUN: MOV    R0, -(SP)          ; сохранение
      MOV    R1, -(SP)          ; содержимого
      MOV    R2, -(SP)          ; регистров
      MOV    R3, -(SP)          ; в стеке;
      MOV    R4, -(SP)          ;
      MOV    R5, -(SP)          ;
      MOV    #A, R4             ; последовательность адресов
                                ; начинается с адреса A;

      MOV    #156170, (R4)+      ; адрес подпрограммы пересылки в стек;
      MOV    R1, (R4)+          ; адрес первого операнда;
      CMP    R0, #1             ; если операция с одним операндом,
      BEQ    C                  ; то идти на метку C;
      MOV    #156170, (R4)+      ; засылка в стек второго операнда;
      MOV    R2, (R4)+          ; адрес второго операнда;
C:     MOV    R3, (R4)+          ; адрес нужной подпрограммы записыва-
                                ; ется в последовательность адресов;

      CMP    R3, #166760        ; если это не подпрограмма перевода
      BNE    G                  ; вещественного в целое, идти на G;
      MOV    #156350, (R4)+      ; подпрограмма пересылки слова из стека
      BR     D                  ; (результат целого типа);

G:     MOV    #156334, (R4)+      ; либо пересылка вещественного числа
D:     MOV    R5, (R4)+          ; из стека по адресу, указанному в R5;
      MOV    #B, (R4)+          ; организовать выход из шитого кода;
      MOV    #A, R4             ; запустить шитый код на выполнение;
      JMP    @ (R4)+            ;

B:     MOV    (SP)+, R5          ; восстановление
      MOV    (SP)+, R4          ; содержимого
      MOV    (SP)+, R3          ; регистров
      MOV    (SP)+, R2          ; из стека
      MOV    (SP)+, R1          ;
      MOV    (SP)+, R0          ;
      RTS    PC                 ; возврат из подпрограммы
A:     .+22

```

По команде "MOV #156170, (R4)+" в эту последовательность записывается адрес подпрограммы, производящей пересылку значения переменной в стек, а по команде "MOV R1, (R4)+" - адрес самой переменной, указанный в R1.

Аналогично засылается в стек второй операнд, если нужен.

После выполнения любой арифметической подпрограммы результат получается в стеке. Его необходимо оттуда извлечь и переслать по адресу, заданному в R5. Для этого используется подпрограмма 156334, пересылающая вещественное число (4 слова), либо 156350, пересылающая

одно слово, если результат целый.

Пример цепочечных вычислений:

```
; Вычисляем выражение (4+7)*3.0625-1.25
MOV    #2,R0          ;двухоперандная команда
MOV    #AR1,R1         ;первое слагаемое
MOV    #AR2,R2         ;второе слагаемое
MOV    #167144,R3      ;адрес подпрограммы сложения
MOV    #REZ,R5         ;адрес результата
JSR    PC,FUN          ; складываем
MOV    R5,R1           ;результат используем
                        ;как первый сомножитель
MOV    #AR3,R2         ;второй сомножитель
MOV    #170210,R3      ;адрес подпрограммы умножения
JSR    PC,FUN          ; умножаем (результат там же - в REZ)
MOV    R5,R1           ;результат используем как уменьшаемое
MOV    #AR4,R2         ;вычитаемое
MOV    #167124,R3      ;адрес подпрограммы вычитания
JSR    PC,FUN          ; вычитаем
; Печатаем полученный результат на дисплей
ADD    #10,R5          ;засылаем результат в стек;
MOV    -(R5),-(SP)
MOV    -(R5),-(SP)
MOV    -(R5),-(SP)
MOV    -(R5),-(SP)
JSR    PC,@#164710     ;вызываем подпрограмму перевода
                        ;числа в символьную строку;
ADD    #4,SP           ;восстанавливаем стек;
MOV    #3027,R1        ;адрес символьной строки
MOV    #20000,R2       ;ограничитель строки - пробел
EMT    20              ;печатаем сформированную строку
; Аргументы:
AR1:   .#40600.#0.#0.#0 ;4.0
AR2:   .#40740.#0.#0.#0 ;7.0
AR3:   .#40504.#0.#0.#0 ;3.0625
AR4:   .#40240.#0.#0.#0 ;1.25
REZ:   .#0.#0.#0.#0     ;результат
```

6.3. Использование системных переменных МОНИТОРа БК

Область ОЗУ в диапазоне адресов от 0 до 377 МОНИТОР БК использует для хранения своих рабочих переменных (используются те адреса, которые не заняты векторами прерывания, см.п.4.4.3). В этих переменных указываются текущие режимы работы устройств, например, режимы работы дисплея. Зная о назначении тех или иных переменных, можно добиться таких эффектов, которые невозможны при "штатном" использовании системного программного обеспечения.

Рассмотрим несколько примеров использования системных ячеек. Примеры даны на Бейсике и на языке ассемблера.

Таблица 20. Системные переменные

Адрес	Длина байт	Назначение
040	1	Режим 32/64 (0 - режим "64", 377 - режим "32")
041	1	Инверсия экрана (0 - выкл., 377 - вкл.)
042	1	Режим расширенной памяти (0 - выкл., 377 - вкл.)
043	1	Регистр (0 - LAT, 200 - РУС)
044	1	Подчеркивание (0 - выкл., 377 - вкл.)
045	1	Инверсия символа (0 - выкл., 377 - вкл.)

046	1	Режим ИСУ (0 - выкл., 377 - вкл.)
047	1	Режим БЛР (0 - выкл., 377 - вкл.)
050	1	Режим ГРАФ (0 - выкл., 377 - вкл.)
051	1	Режим ЗАП (0 - выкл., 377 - вкл.)
052	1	Режим СТИР (0 - выкл., 377 - вкл.)
053	1	Режим 32/64 в служебной строке (0-"64", 377-"32")
054	1	Подчеркивание в служ. строке (0-выкл., 377-вкл.)
055	1	Инверсия служ. строки (0 - выкл., 377 - вкл.)
056	1	Гашение курсора (0 - выкл., 377 - вкл.)
104	1	Код последнего введенного символа
112	10	Положения табуляторов (побитно при режиме "64")
126	2	Адрес буфера программируемого ключа K10
130	2	Адрес буфера программируемого ключа K1
132	2	Адрес буфера программируемого ключа K2
134	2	Адрес буфера программируемого ключа K3
136	2	Адрес буфера программируемого ключа K4
140	2	Адрес буфера программируемого ключа K5
142	2	Адрес буфера программируемого ключа K6
144	2	Адрес буфера программируемого ключа K7
146	2	Адрес буфера программируемого ключа K8
150	2	Адрес буфера программируемого ключа K9
153	1	Тип операции в режиме ГРАФ (0 - СТИР, 1 - ЗАП)
154	2	Унитарный код положения графического курсора
160	2	Абсолютный адрес курсора в ОЗУ экрана
162	2	Константа смещения одного алфавитно-цифрового символа относительно другого
164	2	Длина ОЗУ экрана в символах
176	2	Координата X последней графической точки
200	2	Координата Y последней графической точки
202	2	Начальный адрес ОЗУ экрана
204	2	Адрес начала информационной части экрана минус 40000
206	2	Длина ОЗУ экрана в байтах
210	2	Длина информационной части экрана
212	2	Код цвета фона на экране
214	2	Код цвета символа на экране
216	2	Код цвета фона в служебной строке
220	2	Код цвета символа в служебной строке
260	2	Адрес подпрограммы, выполняемой при прерывании по вектору 60. Если 0, то ничего не выполняется
262	2	Если 0, то код клавиши "BK" равен 12, иначе 15
264	2	Начальный адрес загруженной программы
266	2	Длина загруженной программы

Примечание: В буфере программируемого ключа записана длина соответствующего ключа (в символах), далее - сами символы.

Пример 1. Печать красными буквами на синем фоне:

```

10 РОКЕ &0214,-1      MOV    #177777,@#214    ;красный цвет;
20 РОКЕ &0212,&052525  MOV    #52525,@#212     ;синий фон;
30 ?"Приветик!"       MOV    #TXT,R1          ;печатаем
                        CLR     R2              ;текст
                        EMT     20              ;
                        HALT
                        TXT: .A:Приветик!

```

Пример 2. Запись символов в служебную строку:

```

10 РОКЕ &0160,РЕЕК(&0204)+&036100      MOV    @#204,@#160
                                           ADD    #36100,@#160

```

20 ? "текст"

MOV #TXT,R1

CLR R2

EMT 20

HALT

TXT: .A:текст

Примечание: эта последовательность проходит только в программе; если попытаться проделать ее в непосредственном режиме, то после оператора POKE ... Бейсик напишет "Ок" в служебной строке и курсор вернется на свое прежнее место.

Пример 3. Печать "в столбик":

10 X=PEEK(&O162)

MOV @#162,R3 ;запомнить старый режим

20 POKE &O162,&O100

MOV #100,@#162

30 ?"текст"

MOV #TXT,R1

CLR R2

EMT 20

40 POKE &O162,X

MOV R3,@#162 ;восстановить режим

HALT

TXT: .A:текст

ПРИЛОЖЕНИЕ. КОДЫ СИМВОЛОВ БК-0010-01

Таблица 1. Коды символов

Код символа		Символ (или клавиша)	Примечание
десятичный	восьмеричный		
3	3	КТ	Отказ от вводимой строки
7	7	СУ/G	Звонок (одновременное нажатие клавиш "СУ" и "G")
8	10		Курсор влево
10	12	┘	Клавиша "BK"
12	14	СБР	Очистка экрана
13	15	СУ/М	Установка позиции табуляции
14	16	РУС	Включение русского шрифта
15	17	ЛАТ	Включение латинского шрифта
16	20	СУ/Р	Сброс позиции табуляции
18	22	СУ/R	Исходная установка курсора
19	23	ВС	Возврат строки
20	24	СУ/Т	Перевод курсора на 8 позиций вправо
21	25	СУ/U	Перевод курсора в начало следующей строки
22	26	├	Сдвигка в строке
23	27	└	Раздвигка в строке
24	30		Удаление символа, стоящего перед курсором ("забой")
25	31		
26	32		
27	33		
28	34	СУ/Э (\)	Клавиши перемещения курсора
29	35	СУ/Щ (/)	
30	36	СУ/Ч (\)	
31	37	СУ/ (/)	
32	40		Пробел
33	41	!	Восклицательный знак
34	42	"	Кавычки