ZX - РЕВЮ
N9 (9) 1991

МП "ИНФОРКОМ", 107241, МОСКВА, Б-241, а/я 37

СЕКРЕТЫ ПЗУ

(продолжение)

Процедуры редактора.

Редактор вызывается в двух случаях: из главного вычислительного блока - когда пользователь вводит в систему БЕЙСИК-программу и из процедур, занимающихся обработкой команда INPUT.

0F2C-0F2F EDITOR

Это точка входа. Здесь запоминается на машинном стеке содержимое системной переменной ERR-SP (23613,23614).

0F30-0F37 ED-AGAIN

В ERR-SP устанавливается адрес 107F, где расположена процедура обработки ошибок, возникающих при редактировании строк.

0F38-0F6B ED-LOOP

Организуется цикл, принимающий код нажатой клавиши и выполняется распределение работы по процедурам, в зависимости от того, что это за код.

Опрашивается клавиатура вызовом WAIT-KEY (15D4). Принятый код находится в аккумуляторе. Устанавливается продолжительность звукового сигнала по системной переменной PIP(23609) и его высота. Вызовом BEEPER (03B5) выдается звуковой сигнал, свидетельствующий о том, что клавиша нажата, после чего на стеке выставляется адрес начала этой процедуры (0F38), что и обеспечит работу в цикле.

Далее анализируется принятый от клавиши код.

Если он больше 16H, т.е. это печатный или графический символ или токен ключевого слова - переход на ADD-CHAR (0F81).

Если он равен 6, то это запятая оператора PRINT и выполняется переход туда же.

Если он меньше 10H, но больше 6, то это клавиша редактирования и переход выполняется в ED-KEYS (0F92).

Если код больше или равен 10H, но меньше 16H, то это цветовой код и переход на ED-CONTR (0F60). Обратите внимание, что после цветового кода должен идти еще один код (операнд), определяющий значение данного атрибута цвета.

Последний случай - если код равен 16H (AT) или 17H (TAB), после которых должны следовать два операнда, определяющие координаты позиции печати. Если идет обработка БЕЙСИК-строки (а это определяется по выключенному седьмому биту системной переменной FLAGX), то их не надо исполнять и выполняется переход на процедуру ED-IGNORE (101E).

Если же идет обработка INPUT, то их надо исполнить. Принимается первый операнд вызовом WAIT-KEY (15D4) и работа продолжается в следующей процедуре.

0F5C-0FB0 ED-CONTR

Здесь обрабатываются управляющие коды цвета, а также AT и TAB.

Вызовом WAIT-KEY (15D4) вводится операнд (для кодов управления цветом он первый, а для AT и TAB - уже второй).

Вызовом MAKE-ROOM (1655) выделяется место в области памяти БЕЙСИКа для вставки этих управляющих кодов с операндами в БЕЙСИК-строку. Работа заканчивается переходом на ADD-CH-1 (0F8B)

0F81-0F8A ADD-CHAR

Задача - ввести принятый с клавиатуры символ в редактируемую Бейсик-строку или INPUT-строку. Выключением нулевого бита системной переменной MODE (23617) устанавливается курсор "K". Адрес курсора в экранном файле вводится из K-CUR(23643,23644) в регистровую пару HL и вызовом процедуры ONE-SPACE (1652) выделяется один байт памяти в редактируемой области для ввода туда принятого кода.

0F8B-0F91 ADD-CH-1

В выделенное место вставляется принятый код, запоминается положение курсора и выполняется возврат. Поскольку на стеке был выставлен адрес ED-LOOP (0F38), то это означает повторение цикла.

0F92-0F9F ED-KEYS

Сюда мы попадаем, если была нажата клавиша редактирования. Здесь в зависимости от того, что это была за клавиша, выполняется переход на соответствующую процедуру. Адрес перехода вычисляется с помощью нижележащей таблицы, в которой ведены "смещения" относительно базового адреса.

В качестве базового вводится адрес 0F99. Поиск по таблице исполняется прибавлением базового адреса к коду клавиши редактирования. Из полученного адреса извлекается величина "смещения", которая прибавляется к полученному адресу и дает адрес обрабатывавшей процедуры. Этот адрес выставляется на стеке и командой RET (возврат) осуществляется переход по вычисленному адресу процедуры.

0FA0-0FA8 - таблица клавиш редактирования.

Символ	смещение	адрес процедуры
07 EDIT	09	0FA9
08 курсор влево	66	1007
09 курсор вправо	6A	100C
0A курсор вниз	50	0FF3
0B курсор вверх	B5	1059
0C DELETE	70	1015
0D ENTER	7E	1024
0E SYMBOL SHIFT	CF	1076
0F GRAPHICS	D4	107C

0FA9-0FF2 ED-EDIT

Если Вы находитесь в режиме редактирования, то нажатие клавиши EDIT переводит текущую БЕЙСИК-строку в нижнюю часть экрана для редактирования.

Если Вы находитесь в режиме ввода по INPUT, то нажатие этой клавиши выполняет сброс введенной информации для повторного набора.

В регистровую пару HL вводится содержимое системной переменной E-PPC (23625,23626), т.е. номер текущей БЕЙСИК-строки. Далее, если обрабатывается INPUT-строка (определяется по 5-ому биту FLAGX) - переход на CLEAR-SP (1097) для очистки, а если БЕЙСИК-строка, работа продолжается.

Вызовом LINE-ADRR (1966) вводится начальный адрес этой строки.

Вызовом LINE-NO (1695) по адресу определяется номер строки.

Выполняется проверка номера строки на ноль. Если это так, то следует переход на CLEAR-SP (1097) для очистки области редактирования. (Нулевая строка не редактируется!)

Определяется длина строки и вызовом TEST-ROOM (1F05) выясняется, есть ли место для ее копирования из Бейсик-области в область редактирования.

В регистре HL устанавливается начальный адрес строки, а на стеке - адрес процедуры, обрабатывающей текущий канал ввода-вывода (из системной переменной CURCHL (23633,23634)).

Вызовом CHAN-OPEN (1601) открывается канал "R". В отличие от прочих стандартных каналов "K", "S" и "P", он является внутренним и не связан с внешними периферийными устройствами. Он связывает программную область БЕЙСИКа (где хранится программа) с областью редактирования.

Вызовом OUT-LINE (1855) выполняется "печать" БЕЙСИК-строки. Поскольку открыт канал "K", то эта "печать" представляет из себя ни что иное, как копирование текущей строки из программной области в область редактирования.

Далее выставляется положение курсора - в строке редактирования после ее номера и вызывается CHAN-FLAG (1615), которая восстанавливает предыдущее значение канала ввода/вывода и выставляет в соответствии с ним значения флаговых переменных.

0FF3-1000 ED-DOWN

Если выполняется обработка INPUT-строки - переход на ED-STOP (1001). В противном случае вызовом LN-FETCH (190F) определяется номер следующей строки и переходом на ED-LIST (106E) запускается автоматический листинг.

1001-1006 ED-STOP

В системной переменной ERG NR (23610) выставляется код 10H (что означает STOP IN INPUT) и выполняется переход на ED-ENTER (1024).

1007-100B ED-LEFT

Курсор перемещается влево посредством вызова ED-EDGE (1031). Далее работа продолжается переходом на ED-CHR (1011).

100C-1010 ED-RIGHT

Проверяется текущий символ. Если это CHR13 ("возврат каретки"), то возврат по RET. Если же нет, то увеличивается на единицу адрес положения курсора.

1011-1014 ED-CUR

Адрес курсора запоминается в системной переменной K-CUR (23643,23644) и выполняется возврат по RET.

1015-101D ED-DELETE

Речь идет о стирании символа, находящегося слева от курсора. Курсор передвигается влево вызовом ED-EDGE (1031). Текущий символ удаляется вызовом RECLAIM-2 (19E8).

101E-1023 ED-IGNORE

Процедура вызывается из ED-LOOP (0F38). Двойным вызовом WAIT-KEY (15D4) игнорируются два вводимых кода.

1024-1025 ED-ENTER

Речь идет о завершении ввода строки. Со стека снимаются адреса ED-LOOP (обеспечивающий цикл ввода символов) и ED-ERROR (устанавливающий месторасположение процедуры обработки ошибок, возникающих при редактировании).

1026-1030 ED-END

В системной переменной ERR-SP выставляется исходный адрес процедуры обработки ошибок. Если при редактировании не было ошибок, выполняется возврат в вызывающую программу, а если были, то переход на их обработку.

1031-103D ED-EDGE

Здесь речь идет о перемещении курсора влево. Во внимание должны быть приняты следующие обстоятельства:

- не достигнуто ли уже начало строки;
- не следует выставлять курсор между управляющими кодами и сопровождающими их операндами.

Вызовом SET-DE (1195) инициализируется регистр DE. В нем выставляется адрес системной переменной E-LINE, если идет редактирование БЕЙСИК-строк или адрес WORKSP, если идет обработка команды INPUT.

Сначала проверяется левый край. Если он достигнут - возврат на ED-LOOP (через адрес на стеке).

103E-1050 ED-EDGE-1

Далее обеспечивается неотделение управляющих кодов от их параметров. Вводится код текущего символа. Если это не код от INK до TAB - переход на ED-EDGE-2 (1051). Иначе пропускается один параметр.

1051-1058 ED-EDGE-2

Если есть еще параметры, переход на ED-EDGE-1 (103E), в противном случае возврат.

1059-106D ED-UP

Обслуживается код CHR$ 11 ("курсор вверх").

Если идет обработка INPUT-строки, то этой клавише делать нечего - сразу выполняется возврат. Работа продолжается, если речь идет о строке БЕЙСИК-программы.

Вызовом LINE-ADDR (196E) и далее LINE-NO (1695) определяется номер строки, в которой находился курсор. Вызовом LN-STORE (191C) в системную переменную E-PPC (23625,23626) заносится этот номер.

106E-1075 ED-LIST

Вызовом AUTO-LIST (1795) выполняется перепечатывание листинга на экране. Работа процедуры заканчивается переходом на процедуру, открывающую канал ввода/вывода CHAN-OPEN (1501) при установленном в аккумуляторе номере потока, равном 0, что открывает канал "K".

1076-107B ED-SYMBOL

Сюда мы попадаем, если был нажат SYMBOL SHIFT. Если речь идет о БЕЙСИК строке из программной области, то его быть не должно и переходом на ED-ENTER (1024) выполняется возврат.

Если же речь идет об INPUT-строке, то это какое-то ключевое слово и работа продолжается.

107C-107E ED-GRAPH

Для приема символа (графического или берущегося при нажатой клавише SYMB SHIFT) выполняется переход на ADD-CHAR (0F81).

107A-1096 ED-ERROR

Сюда мы попадаем в случае возникновения какой-либо ошибки при редактировании.

При работе с любым иным каналом, кроме "K" выполняется переход на конец редактирования ED-END (1026). В противном случае вызовом BEEPER (03B5) выдается звуковой сигнал, свидетельствующий об ошибке. Его длительность берется из системной переменной RASP (23606). Далее выполняется переход на ED-AGAIN (0F30).

1097-10A8 CLEAR-SP

Процедура служит для очистки области редактирования, что выполняется вызовом подпрограммы RECLAIM-1 (19E5) после того, как подпрограмма SET-HL (1190) выставит нужным образом регистровые пары HL и DE.

Здесь же устанавливаются в исходное положение системные переменные K-CUR (23643,23644) и MODE (23617).

Подпрограмма "KEYBOARD INPUT".

Эта подпрограмма имеет очень важное значение. Она возвращает в регистре A процессора код каждой нажатой клавиши. Но основное значение этого блока процедур состоит в том, что именно этот блок стандартно привязан к входному каналу "K" (клавиатура), о чем мы будем еще говорить в следующих выпусках.

10AB-10D7 KEY-IBPUT

Сначала проверяется 3-ий бит системной переменной TV-FLAG (23612). Если он включен (включается в WAIT-KEY (15D4)), то это сигнал на копирование строки из области редактирования или из буфера INPUT в нижнюю часть экрана, что и делается вызовом ED-COPY (111D).

По 5-ому биту системной переменной FLAG (23611) определяется, была ли нажата клавиша. Если нет - возврат.

Если была, ее код считывается из системной переменной LAST-K (23560).

По 5-му биту TV-FLAG (23612) устанавливается, подлежит ли переделке нижняя часть экрана. Если да, то это делается вызовом CLS LOWER (0D6E).

Далее работа распределяется по процедурам. Если принятый код является печатным символом - переход на KEY-DONE (111B). Если это управляющий код - переход на KEY-CONTR (10FA).

Если принятый код свидетельствует о включенном режиме CAPS LOCK или несет информацию о режиме, в котором находится клавиатура, выполняется переход на KEY-M&CL (10DB).

Далее обрабатываются коды FLASH, BRIGHT и INVERSE и работа продолжается переходом на KEY-DATA (1105).

10DB-10E5 KEY-M&CL

Не оставляя обработку режима CAPS LOCK процедурам, обслуживающим клавиатуру, здесь при обработке ввода через INPUT выставляются необходимые флаги и выполняется переход на KEY-FLAG (10F4).

10E6-10F3 KEY-MODE

Точно также обрабатывается переключение клавиатуры в другой режим.

10F4-10F9 KEY-FLAG

Заключительные операции по установке флагов и возврат.

10FA-1104 KEY-CONTR

Здесь обрабатываются управляющие коды (кроме FLASH, BRIGHT и INVERSE), т.е. либо INK, либо PAPER. Какой из них конкретно – определяется по тому, была ли нажата при вводе этого кода клавиша SHIFT (по третьему биту кода). И в том и в другом случае переход на KEY-DATA (1105).

1105-110C KEY-DATA

Принятый управляющий код запоминается в системной переменной K-DATA (23565), в регистр DE устанавливается адрес 110D (зачем - будет ясно чуть ниже) и выполняется переход на KEY-CHAN (1113) для смены адреса процедуры, выполнявшей обработку информации, поступавшей из входного канала "K". Там он будет установлен на KEY-NEXT (110D).

110D-1112 KEY-NEXT

Поскольку после управляющего кода должен идти операнд, поэтому и возникла необходимость переключить канал сюда. Здесь этот операнд принимается в аккумулятор и вновь в регистр DE засылается адрес 10A8 (подготовка к восстановлению исходного адреса процедуры, обрабатывающей информацию от канала).

1113-111A KEY-CHAN

Здесь выполняется переключение каналов.

111B-111C KEY-DONE

Финишная процедура. По возвращении из нее флаг CARRY включен, если код от клавиши был успешно принят, а сам этот код находится в аккумуляторе.

Подпрограмма "LOWER SCREEN COPYING"

Эта подпрограмма также представляет из себя пакет процедур. По своему назначению она является вспомогательной и вызывается для работы тогда, когда надо взять редактируемую строку из области редактирования или из буфера INPUT и поместить в нижнюю часть экрана.

111D-1150 ED-COPY

Основная процедура этого блока.

Вызовом TEMPS (0D4D) вводит постоянные цветовые атрибуты.

Устанавливает необходимые флаги.

Запоминает на стеке системные переменные S-POSNL(23690,23691), ERR-SF(23613,24614).

Устанавливает в ERR-SP в качестве адреса процедуры обработки ошибок значение 1167, что соответствует процедуре ED-FULL.

Запоминает на стеке содержимое ECHO-E.

Вызовом SET-HL (1195) инициализирует регистр HL.

Вызовом OUT-LINE2 (187D) печатает строку.

Вызовом OUT-CURS (18E1) печатает курсор.

Вызовом TEMPS (0D4D) вновь вводятся постоянные цветовые атрибуты.

1150-115D ED-BLANK

Если напечатанная строка не равна по длине экранной строке, то это имеет некрасивый вид, поскольку могут различаться по цвету то, что печатается и то, что в этом месте экрана было до печати.

Поэтому необходимо дополнить напечатанную строку до полной посредством печати пробелов в установленном цвете.

Если в этом нет необходимости, то сразу выполняется переход на ED-C-DONE (117C).

Для исполнения печати пробелов - переход на ED-SPACES (115E).

115E-1166 ED-SPACES

Вызовом процедуры PRINT-OUT (09F4) выполняется печать пробела. Далее следует переход на ED-BLANK (1150), что обеспечивает работу в цикле.

1167-117B ED-FULL

Здесь обрабатывается возникшие ошибки. Вызовом BEEPER (03B5) выдается звуковой сигнал с продолжительностью, установленной в системной переменной RASP (23608) и следует переход на финиш ED-C-END (117E).

117C-117D ED-C-DONE

При нормальном завершении работы перед выходом необходимо снять запомненные

на стеке параметры и восстановить их в регистрах процессора, что здесь и делается.

117E - 118F ED-C-END

Восстанавливаются значения системных переменных ERR-SP (23613,24614), ECHO-E (23682, 23663) и др., после чего выполняется возврат в вызывающую процедуру.

Подпрограмма "SET-HL AND SET-DE".

Здесь содержатся всего две вспомогательные процедуры.

1190-1194 SET-HL

В регистровую пару HL засылается содержимое системной переменной WORKSP (23649,23650) минус единица. Т.е. после этого регистр HL указывает на конец рабочего пространства.

1195-11A6 SET-DE

В режиме редактирования здесь в регистровую пару DE устанавливается из системной переменной E-LINE (23641,23642) адрес начала области редактирования и выполняется возврат.

Если Вы не находитесь в режиме редактирования, здесь устанавливается значение взятое из WORKSP (23649,23650).

Подпрограмма REMOVE-FP

Эта подпрограмма имеет сомнительное отношение к процедурам редактора, поскольку в нем нигде не вызывается и наверное осталась в этом блоке процедур по каким-то историческим причинам.

Ее назначение - удалить из БЕЙСИК-строки все числа, записанные в формате с плавающей точкой (в пятибайтной интегральной форме). Кстати, об этом формате Вы можете почитать в нашем трехтомнике, поскольку в рамках материалов "Секреты ПЗУ" мы до этого дойдем еще не скоро.

В двух словах: действительные числа с плавающей точкой хранятся в БЕЙСИК-строках в виде пяти байтов, перед которыми стоит код 0E.

Эта процедура отыскивает такой код в БЕЙСИК-строке и, проходя шесть раз, удаляет шесть ненужных байтов - пять байтов числа и шестой байт - сам код 0E.

Удаление байтов выполняется вызовом процедуры RECLAIM-2 (19E8).

А сейчас мы прощаемся до следующего выпуска, в котором начнем рассматривать наиболее важный раздел ПЗУ - блок исполняющих процедур.

BETA BASIC

Мы продолжаем печать инструктивных материалов по работе с языком программирования BETA BASIC. Начало - см. стр. 134.

20. Команда: PROC

Структура: PROC имя

Клавиша: 2

См. также DEF PROC, END PROC.

PROC - это сокращение от слова процедура.

Процедуры похожи на GO SUB с именем, но имеют то преимущество, что Вам не надо беспокоиться о том, где в программе находится определение процедуры. Компьютер найдет ее где угодно, при условии, что DEF PROC - первое выражение в строке (это ограничение несколько увеличивает скорость вычислений).

Имя процедуры должно удовлетворять тем же условиям, что и имя переменной. Определение процедуры может иметь длину в сколько угодно строк - оно заканчивается оператором END PROC. Так же, как и DEF FN, DEF PROC игнорируется программой, если они не были вызваны через PROC. Программа в своих вычислениях просто перепрыгивает через них. Все переменные, используемые в основной программе, доступны и для процедуры, а все переменные, которые создает или изменяет процедура, доступны для главной программы.

```
10 FOR n=1 TO 20:PROC draw square: NEXT n
200 DEF PROC draw square: LET side-RND*20+20
210 PLOT RND*215,RND*135:DRAW side,0:DRAW 0,side
220 DRAW-side,0:DRAW 0,-side:END PROC
230 PRINT "Finished"
```

Идеальная "структурированная" программа должна содержать серию заданных процедур, каждая из которых выполняет определенную работу и может быть оттестирована независимо. Они вызываются в нужном порядке из основного тела программы, например:

```
100 PROC Set up board (установить игровое поле)
110 PROC Play game (играть в игру)
120 PROC Show Score (показать счет)
130 STOP
```

Если процедура не была задана, то использование PROC вызовет сообщение:

```
W: "Missing DEF FROC" (W: "Отсутствует DEF PROC")
```

Это же произойдет, если Вы используете END PROC, не задав DEF PROC. исключение составляет тот случай, когда в стеке есть доступный адрес возврата. END PROC не знает, что это не адрес PROC и "благополучно" возвратится к указанной строке. Если Вы забудете дать END PROC, то программа выдаст сообщение:

```
X: "No END PROC" (X:"Отсутствует END PROC")
```

Программа будет пытаться "перепрыгнуть" через определение процедуры, но не найдет где она кончается.

21. Команда: RENUM

Структура: RENUM // (начало TO конец) // //LINE новое начало// // STEP шаг//

Клавиша: 4

Просто RENUM используется для перенумерации всей программы таким образом, что номер первой строки равен 10 и интервал между строками также равен 10. Примеры.

RENUM (130 TO 220) - перенумерация указанного блока;

RENUM (130 TO) - перенумерация строк со 130 до конца программы;

RENUM (TO 100) - перенумерация всех строк программы, кроме нулевой, до строки 100 включительно.

Если блок нельзя перенумеровать так, чтобы после его перенумерации строки не "влезли" бы в строки остальной части программы, то Вы получите сообщение:

```
G: NO ROOM for line" (G: "нет места для строки")
```

В данном контексте оно имеет несколько иное значение, чем в обычном стандартной Бейсике.

Если Вам надо задать специфический номер первой строки, то используйте LINE. (Это ключевое слово, т.е. набирается не по буквам).

Шаг между строками задается с помощью STEP. Это тоже ключевое слово.

Примеры

```
RENUM
RENUM LINE 100 STEP 20
RENUM (1540 TO) LINE 2000
RENUM (100 TO 176) LINE 230 STEP 5
```

Вы понимаете, что после перенумерации могут "пострадать" операторы перехода GO TO или GO SUB. Поэтому БЕТА-БЕЙСИК при перенумерации выявляет такие строки и изменяет их. Вместе с тем, могут быть особые случаи, например когда Вы делаете вычисляемый GO TO.

Сначала программа делает поиск выражений, которые могут нуждаться в перенумерации, например GO TO L*100. Такое выражение может указывать на строку, которую Вы собираетесь перенумеровать. Этот случай может оказаться слишком сложным для обычной процедуры перенумерации, поэтому перенумерация прервется с сообщением:

```
Y: "TOO hard" (Y: "слишком сложно")
```

Это дает вам возможность переделать выражение, может быть с использованием оператора ON (см. выше). Можете также на первых порах просто изменить это выражение как-нибудь, чтобы удовлетворить RENUM временно. Например, спрячьте недостойное выражение в кавычки

```
PRINT "GO TO L*100"
```

или поставьте перед ним REM

```
REM GO TO L*100.
```

После того, как все недопустимые выражения будут спрятаны, дайте еще раз команду RENUM. Теперь будут изменены в программе все ссылки на перенумерованные строки, включая: GO TO, GO SUB, RESTORE, RUN, ON, ON ERROR, TRACE, LIST, LLIST, LINE, DELETE.

Функцию CLOCK это не касается. Ее надо переделать самому! (Это потому, что после CLOCK может быть не только номер строки, но и число, показывающее режим настройки будильника).

Примечание. RENUM организует промежуточную таблицу для хранения данных, которая размещается в экранной области памяти. Она чистится после окончании перенумерации, поэтому не пугайтесь, глядя на экран.

22. Команда: ROLL

Структура:ROLL направление; // X координата, Y координата; ширина, длина//

Клавиша: R

См. также SCROLL

ROLL смещает весь экран или заданное окно на один пиксел вверх, вниз, влево или вправо. Все, что выйдет за границы окна, появится из противоположной границы. Другими словами, команда не уничтожает на экране ничего, она просто переорганизует информацию (в отличие от SCROLL). Направление ROLL задается числом 5,6,7 или 8 после команды. Поскольку каждая команда ROLL выполняет незначительное смещение, то лучше всего ее использовать в цикле. Нарисуйте график (DRAW) или дайте LIST, затем попробуйте:

```
100 FOR d=5 TO 8: FOR p=1 TO 100
110 ROLL d: NEXT p: NEXT d: STOP
```

ROLL по диагонали может быть выполнен последовательной подачей ROLL вверх и ROLL вправо.

ROLL для окна экрана вводится подачей после кода направления еще четырех параметров: координат X и Y левого верхнего угла окна (в той же системе координат, что и для PLOT и DRAW), а также ширины окна (в символах) и его длины (в пикселах). Ширина может быть от 1 до 32, а длина от 1 до 176. ROLL может быть очень эффективной командой в играх для организации перемещения объектов или ландшафта. Чрезвычайно интересный, хотя и вызывающий легкое головокружение эффект может быть получен организацией

нескольких взаимопересекающихся окон таким образом, что экран начинает двигаться и дергаться в разных направлениях.

Интересный эффект дают следующие программы:

```
100 LIST:LIST:LIST
110 ROLL 5;0,175;32,88 :ROLL 6;0,175;16,176
120 ROLL 8;0,87;32,88 :ROLL 7;128,175;16,176
130 GO TO 110
```

или

```
200 FOR N=1 TO 7:LIST:NEXT N
210 FOR L=1 TO 175:ROLL 5;0,175;32,L:NEXT L
```

23. Команда SCROLL

Структура: SCROLL // направление//;// X координата, Y координата; ширина, длина//

Клавиша: S

См. также ROLL

SCROLL по синтаксису очень похожа на ROLL. Основное отличие состоит в том, что SCROLL можно использоваться без параметров. В этом случае передвигается целиком весь экран вверх на один символ.

Если после SCROLL дать 5,6,7,8, то весь экран будет смещен на один пиксел в соответствующем направлении. Все, что выйдет за пределы экрана, пропадет совсем.

Новый экран, который "вытягивается" с противоположной стороны - чистый.

Можно передвигать информацию в части экрана - в окне. Для этого после команды должны стоять: код направления движения;

- координаты X и Y левого верхнего угла окна (в той же системе, что и для DRAW и PLOT);

- ширина окна (в единицах символов);

- его длина (в пикселах). SCROLL так же, как и ROLL, широко применяются в играх для создания движущихся изображений. Опробуйте примеры, приведенные для ROLL с командой SCROLL и сравните результат.

А вот пример программы, которая печатает строку и прогоняет ее по экрану:

```
100 LET A$="A NICE LONG STRING.." (красивая длинная строка)
110 FOR C=1 TO LEN A$
120 PRINT AT 10,31; INK 7; A$(C)
130 FOR P=1 TO 8: SCROLL 5:0,95:32,8: NEXT P
140 NEXT C
150 FOR P = 1 TO 255: SCROLL 5;0,95;32,8: NEXT P
```

Текст будет печататься символ за символом в одном и том же месте экрана белым цветом INK. Цвет может быть и другим, но он должен совпадать с цветом PAPER. Буквы будут медленно "выплывать" из квадрата, в котором они "невидимы".

Внутренний цикл FOR-NEXT сдвигает символ на одну позицию влево прежде, чем напечатается второй символ.

Строка 150 перемещает текст по экрану.

Примечание:

Если Вам покажется неудобным смешивать координатные системы "PRINT AT" и PLOT, то Вы можете воспользоваться усовершенствованной командой PLOT Бета-Бейсика и заменять строку 120 на:

```
120 PLOT 248,95; A$ (C)
```

Этот же принцип может быть использован, чтобы сделать текст привлекательным:

```
200 DATA "LONG AGO,IN A DISTANT GALAXY"
210 DATA "LOTS MORE TEXT, LOTS MORE..."
300 FOR L=1 TO 2: READ A$
310 PRINT AT 21,0; INK 7; A$
320 FOR P=1 TO 8: SCROLL 7: NEXT P: NEXT L
330 FOR P=1 TO 176: SCROLL 7 : NEXT P
```

24.Команда: SORT

Структура:

SORT строковый массив или

SORT числовой массив или
SORT строковая переменная
Клавиша: M

Команда SORT организует строки, числа или буквы в восходящем или нисходящем порядке. Сначала обсудим ее применение со строковыми массивами. Вот программа, генерирующая 100 десятибуквенных случайных строк.

```
100 DIM A$(100,10)
110 FOR S=1 TO 100:FOR L=1 TO 10
120 LET A$ (S,L)=CHR$(RND*25+65)
130 NEXT L:NEXT S: GO TO 200
140 SORT A$
200 FOR S=1 TO 100:PRINT A$(S): NEXT S
```

После того, как массив будет сгенерирован, что займет некоторое время, он будет распечатан. Теперь дайте GO TO 140, и он будет отсортирован и вновь распечатан (RUN не давайте). На сортировку уйдет примерно 1/5 секунды. Оно может немного возрасти, если Вы будете использовать более длинные строки. Количество строк более критично. 200 займут 0,7 сек., а 400 - 3 секунды.

Строки сортируются в соответствии с кодами символов. Пробел предшествует "A", а "A", в свою очередь предшествует "a". Если бы в строке 140 было записано

```
SORT INVERSE a$,
```

то порядок мог бы быть противоположным.

Попробуйте. Можно сортировать и отдельные блоки строк, например первые 20:

```
SORT A$(1 TO 20)
```

Команда SORT A$(30 то) выполнит сортировку от 30-ой строки и до конца. Можно выполнять сортировку и не по первому символу. Так, команда

```
SORT A$()(2 TO)
```

выполнит сортировку всего массива по второму и последующим символам.

Команда SORT дает возможность создавать быстроработающие базы данных. В этом контексте назовем весь массив файлом, а каждую отдельную строку - записью. Области каждой строки могут быть использованы для различных типов информации - это поля. Вам для этого могут понадобиться относительно длинные строки. Файл имен, адресов и других данных, например возраста, может быть организован так, что например первые 20 символов в каждой записи (строке) будут содержать имя персоны, следующие 20 - адрес, а последние - возраст. Поскольку любой возраст укладывается в диапазон от 0 до 255, то можно использовать: LET A$ (S,41);CHR$ age, чтобы поместить возраст в запись. Такое хранение данных несложно и занимает мало памяти.

Предположим, что Вам надо хранить еще что-либо более сложное, например, банковский счет. Если Вы используете:

```
LET A$(S,41 то 46)=STR$ balance,
```

то информация будет помещена в строку, но она будет выровнена по левому полю, т.е. если вам надо ввести "9", то число "9" запишется в позиции 41, также как и цифра "9" от числа 900. В результате по этому полю не может правильно работать сортировка.

Поэтому вам нужно аккуратно записывать все данные с тем, чтобы десятичные точки располагались в одну колонку, тогда у каждой строки единицы, десятки и сотни будут записаны в одних и тех же позициях. Для этого можно пользоваться форматирующей функцией FN U$:

```
LET A$(S,41 TO 46)=FN U$("000.000",balance)
```

Полное описание этой функции Вы найдете, когда прочитаете FN U$, USING, FN C$.

Теперь Вы можете сортировать свои записи по возрасту, по счету в банке или в алфавитном порядке.

Обратите внимание на то, что поскольку код "1" меньше, чем код "2", то меньшие числа будут по результатам сортировки размещаться раньше. Можете пользоваться SORT INVERSE, чтобы большие числа шли раньше.

Сортировка работает также с обычными строками и с одномерными массивами.

```
INPUT S$: SORT S:: PRINT S$
```

Если в INPUT Вы дадите "Fred Bloggs", то на печати получите "BF degglors". Это не

выглядит очень полезным, но это дает возможность работы с числовыми данными, которые наиболее экономично хранятся в качестве строковых переменных. Например;

```
LET S$ (position)=CHR$(data)
```

Это может также применяться к обычным одномерным или двумерным числовым массивам, которые используют тот же синтаксис, что и строковые массивы. Двумерный числовой массив можно рассматривать как таблицу, в которой первая размерность - ряд, а вторая - столбец.

Сортировка чисел идет в четыре раза медленнее, чем строк, потому что SORT должна проверить два различных числовых формата, а также учесть возможность наличия как положительного, так и отрицательного числа.

25. Команда: TRACE

Структура: TRACE номер строки.

Клавиша: T

Эта команда позволяет выполнять отладку Бейсик-программ. Она распечатывает текущую строку, текущее выражение в строке, а также избранные переменные, причем выполняет это с уменьшенной скоростью.

TRACE позволяет выполнять немедленный переход, до начала каких либо вычислений, к специальной подпрограмме. Эта подпрограмма может иметь две переменные (не ключевые слова) line и stat.

line - это номер строки.

stat - это номер выражения в строке.

Речь идет о строке, выполнение которой сейчас начнется. TRACE отключается во время работы своей подпрограммы, но снова включается после RETURN. Содержание этой подпрограммы - Ваше дело! Например:

```
9000 PRINT INVERSE 1 ;line; ":"; stat: RETURN
```

Включите эту процедуру в программу и вставьте инструкцию: TRACE 9000

TRACE вставляется с того места, с которого Вы хотите начать отладку. Если бы хотите отключить TRACE в каком-то месте программы, вставьте TRACE 0.

Использование TRACE дает печать выражений по мере их исполнения в формате, который используют системные сообщения. Чтобы отличить их от системных сообщений, воспользуйтесь оператором INVERSE. Трейсирование отключается после RUN, CLEAR и TRACE 0. Если Вы хотите замедлить процесс, вставьте PAUSE или другое средство по Вашему усмотрению. Переменные также могут выводиться на печать. Если Вам это нужно, то объявите их где-нибудь в начале программы, иначе может быть сообщение "Variable not found" ("переменная не найдена"). Чтобы не перепутать результаты печати от TRACE и результаты печати программы, мы Вам советуем использовать PRINT AT, но при этом надо запоминать место текущей печати. Иначе печать из главной программы все равно будет мешать. Это можно сделать например так:

```
9000 LET col=PEEK 23688:LET row = PEEK 23689
9010 PRINT AT 0,0; line; ":"; stat:"      ","A$ =";A$;"      "
9020 POKE 23688,col:POKE 23689,row:RETURN
```

По этой программе печатается номер исполняемой строки и переменная A$. Пробелы необходимы, чтобы исключить наложение печатаемых данных. Подпрограмма запоминает системные переменные, характеризующие позицию печати с тем, чтобы оно изменялось инструкцией PRINT AT только временно.

26. команда: UNTIL

Структура: UNTIL условие

Клавиша: K

См. DO и LOOP.

Задает условия выполнения циклов DO-LOOP.

27. Команда USING

Клавиша: U

Пример: PRIRT USING format строка,число.

См. также FN U$

Оператор USING, как и функция FN U$ (см. далее) задают спецификацию формата, в котором должны печататься числа. По желанию вместо USING можно использовать FN U$. Желаемый формат задается строковой переменной, в которой используются знаки # вместо ведущих пробелов, нули вместо предшествующих нулей и либо то, либо другое вместо цифр, стоящих после десятичной точки.

```
100 FOR n=1 TO 20:LET X=RND*100
110 PRINT X,USING"###,##";X:NEXT n
```

Обратите внимание, насколько аккуратнее получается печать отформатированных чисел, поэкспериментируйте с различными форматобразующими строками. Некоторые возможные форматирующие строки и результат их действия на число 12.3456 показаны ниже:

```
" # #. # "          12.3
" # # #. # "         12.3
" # # # #. # #"      12.35
" 000.00"           012.35
" 00"               12
" R00.00"           R12.35
" 0.00 "            %..3
```

Предпоследний пример показывает, что в форматирующей строке можно использовать не только # и 0, но и другие символы. В этом случае они напрямую передаются на печать.

Последний пример показывает печать знака %, который говорит о переполнении заказанного формата.

FN U$ применяется таким же образом, но:

вместо PRINT USING A$; число

надо: PRINT FN U$ (A$,число).

28. Команда WHILE

Структура: WHILE условие

Клавиша: J

Позволяет вводить условие для завершения циклов DO-LOOP.

29. XOS,XRG,YOS,YRG

Эти четыре слова не являются ключевыми - это специальные переменные, которые позволяют изменять начало отсчета координат и масштаб координатной сетки, используемой командами PLOT, DRAW и CIRCLE.

XOS и YOS задают x-координату и y- координату начала системы отсчета, а XRG и YRG - масштаб по осям X и Y.

Эти переменные устанавливаются через CLEAR и RUN. очень часто изменение начала отсчета системы координат бывает более удобным, чем изменение выражений PLOT, например:

```
LET XOS = 126: LET YOS = 88
```

Эти команды сдвигают начало системы координат в центр экрана.

XRG нормально имеет значение 256 (т.е. Вы можете поместить до 256 точек на оси X). YRG нормально установлено в 176.

```
10 GO SUB 100: REM normal
20 LET XRG = 128: GO SUB 100
30 LET YRG = 88: GO SUB 100
40 LET XRG=256:GO SUB 100:STOP
100 CLS:PLOT 0,0:DRAW 50,10: DRAW 0,50
110 DRAW 50,0:DRAW 0,-50: PAUSE 100: RETURN
```

Сначала изображается нормальный квадрат, затем сжатый по оси X, далее сжатый по обеим осям и, наконец, он сжимается только по оси Y.

Нижеследующий пример показывает изображение графика функции синус.

```
100 LET XRG = 2*PI
110 LET YRG = 2.2
120 LET YOS = 1.1
```

```
130 FOR n=0 TO 2*PI STEP 2*PI/256
140 PLOT n,SIN n:NEXT n
```

Заметьте, что позиция начала координат тоже масштабируется.

При использовании функции CIRCLE величина радиуса не масштабируется, поэтому получить растянутую окружность таким способом не удается.

На этом мы пока прощаемся до следующего выпуска, в котором рассмотрим дополнительные функции Бета-Бейсика.

ЦЕНА ИДЕИ

Сегодня мы продолжаем (и завершаем) несколько необычный для нас раздел "Цена идеи". Начало см. в №6. стр. 131.

Чтобы до конца быть честными, мы должны признаться, что начало статьи вызвало неоднозначную оценку некоторых читателей, около десяти писем пришло от тех, кто счел, что мы уходим в сторону от главной линии и задумали что-то нехорошее.

Это не совсем так. Все гораздо проще. Дело в том, что мы уже несколько месяцев печатаем заметки Стива Тернера "Профессиональный подход", и этот материал не вызывает у читателей сомнений. А одна из ближайших статей его цикла как раз и посвящена вопросам маркетинга собственных разработок. Беда лишь в том, что его материал, как бы хорош он ни был, применим только в Англии и абсолютно неприменим у нас.

Мы много занимаемся маркетингом. Абсолютно все наши разработки проходят маркетинговое исследование у специалистов и нам есть что сказать по этому поводу, тем более что двухлетний опыт кое-чего стоит, вот мы и взяли на себя труд довести то, что хотел, но не мог Вам порекомендовать Стив Тернер.

Кстати, мы не занимаемся маркетингом чужих разработок (т. е. не приобретаем разработки с целью перепродажи) просто потому, что имеем много своих и достаточно загружены перспективными работами. Но мы охотно проводим маркетинговые исследования для других фирм, например можем Вам абсолютно точно сказать по какой цене Вам надо продавать ту или иную разработку, чтобы удовлетворить поставленной задаче. Можем точно предсказать количество покупателей, которое Вы приобретете при заданной Вами цене и избранном плане рекламной кампании, можем дать оптимальный (наиболее дешевый и наиболее эффективный) план рекламной кампании. Такие услуги стоят очень недешево, но заказчик их мгновенно оправдывает, можно ведь только поражаться тому, сколь много теряют разработчики только из-за неправильной оценки предполагаемого объема поставок и связанного с этим неверного определения цены единицы изделия.

Итак, в прошлом выпуске мы подошли к тому, что для того, чтобы подсчитать компенсацию, которую должен получить разработчик идеи от фирмы, принявшей на себя ее реализацию, надо определить ставку роялти, т. е. тот процент от дохода фирмы, который должен получать разработчик (по данному виду товара).

Мы также указали, что этот процент зависит от девяти факторов и находится в пределах от 0. 5% - до 15%.

Первый вопрос от умудренного читателя почему только 15%, а почему не половину? Здесь все очень просто - это мировой опыт с ним приходится считаться. Просто в любой стране есть налоги на прибыль и другие. И при увеличении ставки роялти выше 15% предприниматель-лицензиат попадает в положение, когда ему уже невыгодно заниматься никаким делом.

Сделаем прикидку. Предположим, что нормально преуспевающая фирма работает с уровнем рентабельности 30%-50%. Так оно в большинстве случаев и происходит, выше не дают подняться налоги на сверхрентабельность, а ниже фирма не сможет нормально развиваться (жить сможет, но не сможет расти) и тогда - застой и то, что мы имеем сейчас в экономической сфере.

Примем в среднем - 40%. Это значит, что на 100 рублей продаж себестоимость (т.е. затраты) составляют примерно 70 рублей и прибыль примерно 50 рублей.

30:70*100%=42%.

Проводя Вам отчисления в размере 15% от объема продаж фирма увеличит себестоимость на 15 рублей, т. е. доведет ее до 85 р. и рентабельность составит:

15:85*100%= 17%

Это еще терпимо, хоть и с трудом, если фирме больше просто нечего делать или если от вашей идеи она рассчитывает получить какие-то косвенные преимущества, например

нанести поражение конкуренту или сделать себе имидж.

При отчислении же 20%:

Себестоимость - 90 руб.

Прибыль - 10 руб.

Рентабельность - 11%

Это уже близко к невозможному.

Теперь рассмотрим, от чего зависит процентная ставка роялти и попробуем ее оценить.

1. Важность разработки. Использование новых разработок может принести сравнительно небольшое изменение существующих технологий или продукта, а может привести к технологическому прорыву. Это довольно широкий диапазон. Если, например, оборудование телевизоров стереозвуковой системой находится в нижней части спектра, то разработка волоконной оптики - где-то посередине, а транзисторов и сверхпроводящих керамических материалов - на самом верху.

2. Степень патентной защищенности, возможность соблюдения коммерческой тайны.

Защищенность, предоставляемая патентами или соблюдением коммерческой тайны, состоит из трех отдельных элементов:

2.1 - "открываемость" - относительная легкость или сложность, с которой конкурент может открыть, как функционирует Ваша технология.

2.2 - "повторяемость" - характеризует, насколько легко конкурент сможет повторить технологию после того, как он поймет, в чем заключается ваша идея.

2.3 - "избегаемость" или возможность достижения того же полезного эффекта без необходимости повторения запатентованной разработки. Фотоаппараты "Поляроид", видимо, представляют собой один из наиболее защищенных патентов в истории. Хотя выяснить химический состав пленки и проявителей, используемых фирмой при "мгновенной" съемке, было очень просто, но вот повторить эти составы оказалось не такой уж легкой задачей.

3. Влияние конкурентной ситуации на рынке на возможность использования новых разработок.

Имеются в виду те трудности, с которыми столкнется лицензиат при выходе на рынок с Вашей продукцией. Конечно, чем они выше, тем меньшую сумму он готов Вам заплатить за разработку.

Против Вас здесь работают следующие явления:

3.1 - если в сфере действия вашего нового товара (услуги) при увеличении объема производства однозначно снижаются и удельные за траты. Это не ошибка, действительно этот фактор работает против Вас. Ведь начиная производство Ваш клиент будет находиться в трудных условиях, а когда нарастит обороты и достигнет больших объемов и экономии затрат, то это будет не Ваша заслуга, а его, ведь он это сделает своими трудовыми и материальными затратами.

3.2 - если есть наличие множества различных товаров, удовлетворяющих одну и ту же функциональную потребность;

3.3 - если данная сфера характеризуется приверженностью покупателей одному сорту товара или верностью фирме, конечно в этих условиях труднее выйти на рынок. Кстати, если Ваш лицензиат известная фирма, пользующаяся доверием клиентов, то это тоже не Ваша заслуга, а фирмы;

3.4 - необходимость крупных капитальных вложений при выходе на рынок;

4. Необходимость инвестиций в дальнейшие НИОКР.

Создание прототипов, разработка планов производства и выход на рынок, пробные продажи, а также дизайн и привязка научных разработок к техническим возможностям - все это необходимо для того, чтобы технология получила реальную коммерческую ценность. Но это потребует времени и финансовых ресурсов, поэтому чем больше подобных работ лицензиар уже выполнил, тем большую цену он может запросить.

5. Стадия разработки.

В зависимости от стадии практической готовности разработки риск принятия ее к реализации может быть больше или меньше, снижение риска должно оказывать повышавшее влияние на оценку технологии лицензиатом. Соответственно, цена должна увеличиваться, когда автор патента или идеи берет часть риска на себя, доводя разработку до уровня, максимально приемлемого к запуску в производство.

6. Конкурентные преимущества использования новых разработок.

Они определяются ценой, внешним видом, простотой использования и другими качественными параметрами конечного изделия.

Изделия, предоставляющие отчетливые преимущества по одному или нескольким параметрам, дадут возможность повысить цену на разработку.

7. Возможность рентабельности разработки.

Как отмечалось ранее, цена идеи или разработки - это часть прибыли, которую лицензиат надеется получить в результате ее внедрения. Понятно, что чем больше будет эта прибыль, тем больше будет и цена идеи как в абсолютном, так и относительной выражении, при увеличении рентабельности лицензиаты обычно с большим желанием идут на увеличение цены разработки.

8. Инновационная значимость, технологии и идеи, имеющие широкий спектр использования, обладают большей ценностью. Так, транзистор способствовал развитию нескольких новых отраслей, созданию тысяч новых продуктов, существенному улучшению существующих изделий, микросхемы также революционизировали практически все стороны экономики. Очевидно, что цена технологии, способных породить большое количество практических результатов, будет стоить намного выше, чем технологии и идеи с ограниченными направлениями и будущими возможностями применения.

9. Другие факторы. Рассмотрим три возможных фактора, которые снижают или увеличивают стоимость научной разработки для лицензиата, соответствующим образом изменяя и цену.

9. 1. Исключительность.

Если лицензиат приобретает исключительное право использования разработки или идеи, то получение этого конкурентного преимущества увеличивает цену или ставку роялти.

9. 2. Авансовые платежи.

Владелец новой технологии может потребовать уплату комбинированной цены, состоящей из части, подлежащей немедленной оплате и ставки роялти. Это особенно важно, когда лицензиат - многопродуктовая фирма. Лицензиар рискует, что внимание лицензиата может быть отвлечено необходимостью развития производства других продуктов. Авансовый платеж одновременно стимулирует лицензиата к внедрению и снижает риск, который несет лицензиар, ставка роялти при этом, конечно, снижается.

9. 3 Гарантии использования разработки.

Они предназначены для стимулирования заинтересованности внедряющей фирмы во внедрении Вашей разработки или идеи. Они выполняют схожую с авансовыми платежами функцию. Отсутствие же этих гарантий приводит к увеличению бремени риска, которое несет лицензиар, что увеличивает ставку роялти.

Итак, мы рассмотрели какими факторами определяется размер ставки роялти в

процентах от объема продаж.

Теперь, зная предполагаемый объем продаж и цену единицы продукции, (а это выявляется маркетинговыми исследованиями, хотя это лишь очень малая часть того, что называют маркетинговыми исследованиями) можно рассчитать и величину единовременной выплаты разработчику за коммерческую эксплуатацию его идеи.

$$S = R*C*N/100$$

S - размер паушальной суммы;

R - ставка роялти в %;

C - расчетная цена единицы продукции;

N - объем продаж за период жизни товара (в рассматриваемой нами области этот период надо считать равным одному году - очень быстро меняется материальная база и конъюнктура рынка, кстати, отсюда и проистекает требование высокой рентабельности 30%-50%, т.к. часто приходится переоснащаться.)

По поводу этой формулы надо сделать ряд замечаний. Как Вы видите, размер причитающейся Вам суммы будет зависеть от планового объема продаж и от цены товара, а ни то ни другое от Вас не зависит и определяется только мощностью фирмы, которой Вы предлагаете свою разработку.

То, что фирма занизит цену, чтобы обидеть Вас, маловероятно, т.к. она накажет саму себя.

Возможно обратное - фирма излишне завысит цену так, что упадет объем продаж, зачем она это сделает? А хотя бы потому, что имеет ограниченные мощности и не справится с большим объемом заказов тогда Вам надо просто поискать более мощную фирму и предложить свою разработку ей.

Для нас важно, что Вы располагаете методикой расчета положенной Вам ставки роялти, а фирма - методикой расчета объемов поставок. На переговорах с фирмой Вы перемножаете одно на другое и определяете, что из этого вытекает, а далее можете попробовать переговорить и с другой фирмой.

ПРИМЕРЫ.

Теперь рассмотрим несколько реальных жизненных примеров. Предположим, что несколько человек выполнили несколько разработок в области программного, информационного и аппаратного обеспечения и планируют продать это какой-либо фирме (малому предприятию, кооперативу, заводу).

Разработка А.

Инженер разработал схему и исполнил опытный образец интерфейса локальной сети, который позволяет подключать два и более компьютера для совместной работы. Интерфейс имеет собственное ПЗУ, в котором "зашита" его программная поддержка, оно подключается как "теневое" и не нарушает аппаратно-программную совместимость компьютера.

Образец предназначен для работы с компьютером ABCD.

Перед ним два варианта: А1 - продать схемное решение заводу выпускающему компьютеры ABCD. Понятно, что в этом случае оно останется привязанным только к этой модели.

А2 - продать идею фирме, занимающейся информационным обеспечением. Хоть за идеи платят меньше, чем за готовый образец, но зато потребителей будет больше, поскольку идею можно привязать к любой модели компьютера, а не только к ABCD.

Разработка В.

Программист, исходя из своих личных потребностей адаптировал программу THE LAST WORD 2 таким образом, что этот редактор стал работать не только с русским и латинским шрифтом, но еще и с таджикским.

Удовлетворив свои потребности, он решил еще и продать свою разработку фирме,

распространяющей программное обеспечение, поскольку справедливо полагает, что такая вещь должна послужить еще многим людям.

Разработка C.

Программист разработал программу, обучающую чему-то очень нужному. Пробный маркетинг среди друзей показал, что действительно программа нужна многим, он обращается к фирме с предложением продать право на тиражирование на ленте. Это вариант C1.

Фирма подумала и, понимая нужность и полезность этого предприятия, предложила выпустить эту программу в виде статьи в своем информационном сборнике, а не связываться с кассетой, это вариант C2.

Разработка D.

Ученый в течение двадцати лет занимался тем, что изучал методику лечения разных болезней лекарственными растениями, до этого этим тридцать лет занимался его отец. Собрана уникальная информация. Более 1000 рецептов. В сложных ситуациях предлагаются комплексные решения. И он предлагает свои материалы фирме для подготовки базы данных и распространения ее на дискетах для IBM-совместимых компьютеров. ("Синклер" здесь уже слабоват).

Итак, для рассмотрения влияния факторов на ставку роялти, договоримся, что будем оценивать каждый фактор от 0 до 15 баллов. Всего факторов - 9. Найден коэффициенты K1 ... K9, возьмем среднее значение и пусть оно нам даст прикидочную ставку роялти, конечно это не точное значение, но хотя бы какая-то основа для переговоров с фирмой.

Окончательную же паушальную сумму определим после того, как фирма объявит ориентировочную продажную цену и планируемый объем продаж, поскольку наша методика расчета ставки роялти такова, что она объективно учитывает интересы фирмы, как и Ваши, то можете не сомневаться - занижать цены и объем продаж фирма не будет, ей нет смысла срывать наметившуюся с вами выгодную договоренность. Другое дело, если ее мощности или загруженность текущими работами не позволяют оптимально распорядиться вашей разработкой.

1. Важность разработки.

Все разработки в данном случае безусловно важны. Можно предположить, что разработка D существенно более важна чем остальные, если какая-то из них и предопределяет возможный прорыв, так это она.

Разработка B по-видимому находится в нижней части спектра, а A по-видимому превосходит C, т.к. обучение возможно и без компьютера, а вот соединить два компьютера без интерфейса - удастся вряд ли.

В то же время, вариант A1 направлен на усовершенствование только одной модели, а вариант A2 - любой модели компьютера и потому его важность значительно выше.

Примем в дальнейшую проработку следующий расклад:

Вариант	K1	Вариант	K1
A1	4	C1	3
A2	10	C2	3
B	1	D	11

2. Защищенность.

Здесь рассмотрим все три входящих фактора и возьмем средний между ними.

2.1. Открываемость.

Здесь все просто. Поскольку в вариантах A2, C2 и D речь идет о широком распространении информации, то и открывать уже нечего - этот фактор нулевой.

В остальных случаях он также близок к нулю.

Вариант	K21	Вариант	K21
A1	1	C1	1
A2	0	C2	0
B	2	D	0

2. 2. Повторяемость.

Реально неповторимой можно считать разработку D. Остальные повторимы - разница только в затраченном времени и в профессиональной подготовке. Поскольку варианты A2 и C2 открыты полностью, то здесь этот фактор - нулевой.

Вариант	K22	Вариант	K22
A1	2	C1	1
A2	0	C2	0
B	2	D	15

2. 3. Избегаемость.

Здесь нет нулевых факторов, поскольку даже для открытой информации нужны усилия для обхода. В то же время, вариант D уже не является абсолютно защищенным, поскольку есть и другие методики применения лекарственных растений, да можно лечить и другими методами.

Вариант	K23	Вариант	K23
A1	2	C1	2
A2	2	C2	2
B	3	D	10

В среднем фактор защищенности будет иметь значения:

Вариант	K2	Вариант	K2
A1	1,7	C1	1,3
A2	0,7	C2	0,7
B	2,3	D	8,3

3. Конкурентная ситуация.

Здесь также рассмотрим ряд составляющих факторов.

3.1. Соотношение между постоянными и переменными затратами.

Только в вариантах B и C1 практически все затраты переменные и не снижаются с ростом производства. В вариантах A1 и D есть значительные постоянные предварительные затраты, причем в варианте A1, связанном с переналадкой производственной базы, они особо критичны. В вариантах A2 и C2 они есть, но пренебрежимо малы.

Вариант	K31	Вариант	K31
A1	0	C1	15
A2	15	C2	15
B	15	D	5

3.2. Наличие аналогов.

Выходя на переговоры с Вами, фирма не знает стопроцентно, есть аналоги или нет и определенно, рискует. Ясно только, что в вариантах A2 и C2 наличие аналогов ее не волнует, а наиболее критичен этот фактор для варианта A1, поскольку если она пойдет по неверному пути, и конкурент примет в проработку более удачный вариант, то это ударит не по интерфейсу, а по всему компьютеру в целом и по перспективам развития целого предприятия.

Вариант	K32	Вариант	K32
A1	5	C1	7
A2	15	C2	15
B	7	D	8

3.3. Приверженность фирме.

Очень субъективный фактор, у нас в стране пока нет никаких приверженностей определенному сорту или приверженности определенной фирме.

Тем не менее, можно сказать, что если все равно где и у кого приобретать информацию, то программы приобретать уже сложнее, а перед приобретением компьютера люди очень долго и тщательно взвешивают варианты.

Вариант	K33	Вариант	K33
A1	5	C1	10
A2	15	C2	15
B	10	D	15

3. 4. Необходимость крупных капиталовложении при выходе на рынок.

Варианты A2 и C2 почти не требуют капиталовложений для выхода на рынок, варианты B и C1 требуют небольших капиталовложений в рекламу. Вариант D требует огромных затрат на рекламу, вариант A1 тоже, но меньше, зато там необходимы затраты на предпродажную подготовку (дизайн, оформление, подготовка сопроводительной документации и пр.)

Вариант	K34	Вариант	K34
A1	5	C1	10
A2	15	C2	15
B	10	D	5

В среднем фактор конкуренции будет иметь значение:

Вариант	K3	Вариант	K3
A1	3.7	C1	10.5
A2	15	C2	15.0
B	10	D	8.2

4. Инвестиции в НИР и ОКР.

Варианты B и C1 не требуют инвестиций, программы берутся в чистом виде и размножаются.

Варианты A2 и C2 требуют незначительных затрат на редактирование и подготовку материалов.

Вариант D требует определенных усилий на разработку базы данных.

Наибольших работ требует вариант A, вносящий изменения в технологический цикл изготовления компьютера, но все же это не строительство нового цеха.

Вариант	K4	Вариант	K4
A1	4	C1	15
A2	15	C2	15
B	15	D	12

5. Стадия разработки.

Во всех случаях речь идет о полностью готовой разработке и можно принять K5=15 для всех вариантов, кроме варианта A1, где надо все-таки выполнить привязку к технологическому процессу. Для этого варианта примем K1=7.

6. Конкурентные преимущества. Оценка субъективная, т.к. зависит от положения фирмы на ее рынке. Можно только предположить, что на рынке производителей компьютеров конкуренция более жесткая, чем на смежных и поэтому вариант A1 дает заводу определенные преимущества. Пользующаяся ограниченным спросом и выступавшая вне конкуренции программа B не даст преимуществ, как впрочем и D. Оригинальность открытых идей в разработках A2 и C2 конечно даст преимущества информационной фирме.

С другой стороны, распространение программы B позволяет фирме предлагать в пакете с ней еще и собственную разработку, посвященную смежным вопросам или по крайней мере рассчитывать на то, что ее начнут спрашивать, поэтому вариант B здесь ненулевой.

Вариант	K6	Вариант	K6
A1	7	C1	3
A2	15	C2	15
B	2	D	0

7. Рентабельность.

A1 - рентабельность средняя, ведь речь идет о доработке достаточно дорогой модели компьютера, на которую и так уже падает немало затрат.

У прочих вариантов рентабельность высокая, причем в вариантах B и C1 она немного ниже, чем в вариантах A2, C2, D, т.к. есть затраты на носитель информации (кассету).

Вариант	K7	Вариант	K7
A1	7	C1	12
A2	15	C2	15
B	12	D	15

8. Инновационная значимость.

Некоторые перспективы, что разработка вызовет появление новых есть у интерфейса A1 и у обучающей программы C1. В первом случае возможны разработки специальных программных комплексов для работы в локальной сети, во втором случае возможно развитие идеи в смежные сферы.

То, что появятся новые статьи после публикации A2 и C2 - практически несомненно и здесь этот уровень выше.

Узкая специализация B и D не позволяют надеяться на быстрое развитие направлений.

Вариант	K8	Вариант	K8
A1	3	C1	3
A2	10	C2	10
B	0	D	0

9. Прочие факторы.

9. 1. Исключительность. Везде речь идет об исключительном праве на реализацию идеи. Примем всюду K91=15

9.2. Авансовые платежи. Во всех случаях автор идеи желает получить всю причитавшуюся ему сумму сразу, до начала эксплуатации разработки, это снижает его роялти. K92 = 0.

9. 3. Гарантийные обязательства.

Здесь в них нет нужды, поскольку автор уже получил всю рассчитанную сумму. K93 = 0. Среднее значение K9=5.

Подсчитаем по среднему значению всех факторов размер средней ставки роялти для всех вариантов.

Вариант	R	Вариант	R
A1	4.7%	C1	7.4%
A2	11.1%	C2	12.1%
B	7.0%	D	8.3%

Конечно, результат достаточно условен, но дает Вам хорошую ориентировку перед началом переговоров с фирмой.

Он же позволяет Вам подумать над тем, насколько правильно Вы предполагаете начать распространение своего продукта. Может быть его надо предлагать совсем в другой форме, а не так, как Вы планировали.

Окончательные результаты можно оценить, зная предполагаемый объем продаж и цену единицы продукции. Рассмотрим примеры.

ВСЕ КОЛИЧЕСТВЕННЫЕ ДАННЫЕ, ПРЕДСТАВЛЕННЫЕ В ПРИМЕРАХ - УСЛОВНЫЕ И ИМЕЮТ ТОЛЬКО УЧЕБНО-МЕТОДИЧЕСКОЕ ЗНАЧЕНИЕ

ИХ СОВПАДЕНИЕ С РЕАЛЬНОЙ СИТУАЦИЕЙ ДОЛЖНО РАССМАТРИВАТЬСЯ КАК ЧИСТО СЛУЧАЙНОЕ

Вариант A

Если завод, с которым Вы ведете переговоры, в состоянии поднять цену на свои компьютеры скажем рублей на 50, введя Вашу доработку, и выпускает в год 10000 изделий, то паушальная сумма равна:

S = 10000 * 50 * 0.047=22350 руб.

Если информационная фирма, которой Вы предложите эту разработку имеет 10000 клиентов и в среднем рассчитывает, что 10% из них заинтересуются предлагаемой доработкой по цене 20 руб., то она сможет вам предложить:

S = 1000 * 20 * 0.111 = 2220 руб.

Вариант B

Маркетинговые исследования показывают, что такая программа может иметь спрос в ограниченном регионе, в котором фирма имеет всего 200 клиентов. Предполагается, что 10% смогут приобретать ее по цене 30 руб. за экземпляр.

S = 20 * 30 * 0.07 = 42 руб.

Не очень много, но ведь программа разрабатывалась не для коммерческих целей.

Вариант C

Как показали исследования, можно рассчитывать, что из 10 000 клиентов эту программу готовы приобрести 10% при цене программы 7 рублей.

S = 1000 * 7 * 0.074 = 520 руб.

Рассмотрим встречный вариант, предложенный фирмой - включение ее в сборник. В этом случае фирма удовлетворит все 10 000 клиентов и в среднем каждому программа обойдется в 50 копеек.

S= 10000 * 0.5 * 0.121 =605 руб.

Как видите, вариант, предложенный фирмой, интереснее и менее хлопотный.

Вариант D

Возможность приобретения такой программной разработки ограничена. База данных по лекарственным растениям безусловно интересна тысячам граждан, но они не имеют дома IBM-совместимых машин. Организаций, работающих в этом направлении, и оснащенных требуемой техникой - немного, и для того, чтобы выйти на них, требуются серьезные усилия по рекламе, что конечно поднимет цену готового продукта.

Предположим, что в покрытие этих затрат установлена цена в размере 4 000 рублей и рекламная кампания привела к заключению 20 договоров (очень оптимистическая оценка).

S = 4000 * 20 * 0.083 = 6640 руб.

Конечно, это недостаточная оценка полувекового труда авторов.

Причина скорее всего в том, что рынок еще не созрел для приобретения этой разработки в той форме, в которой она предлагается (в виде базы данных). И концепция маркетинга с самого начала была порочной.

Имеет смысл подумать о заключении договора с крупным издательством о публикации книги массовым тиражом.

При цене книги 20 руб. можно рассчитывать, что она разойдется в количестве 50000 экземпляров.

S = 5000*20*0.063 = 83 000 руб.

Данная оценка ближе к истине. Правда, в этом случае стоило бы пересчитать ставку роялти. По всей видимости она будет несколько иной.

ПРОФЕССИОНАЛЬНЫЙ ПОДХОД

Мы продолжаем печать заметок известного британского программиста Стива Тернера и сегодня мы поговорим о самом щепетильном вопросе, связанном с программированием - об ошибках в программах. (Начало см. на стр. 49, 70, 117, 143)

Многие из Вас, возможно, видели отрывки из американского фильма "Звездные войны". Наиболее захватывающие сцены космических сражений в этом фильме моделировались с помощью компьютеров и, надо сказать, что суммарные трудозатраты программистов на подготовку всего обширного комплекса программного обеспечения составили 3 тысячи человеко-лет.

Спрашивается - как же можно разрабатывать такие грандиозные проекты и создавать такие обширные системы без ошибок, способных свести на нет огромные трудозатраты.

Да, конечно, судьба планеты не зависит от правильности работы Вашей программы, но в принципе те же проблемы стоят и перед Вами. Почему в программах всегда есть "жучки"? Почему программисты так много ошибаются? Можно ли избежать ошибок или, по крайней мере, постараться удалить их из готового продукта?

Ошибки делают все. Одна медсестра сказала, что если бы она делала столько ошибок, сколько делают программисты, никто из ее пациентов не выжил бы.

Я думаю, что она на самом деле тоже делает немало ошибок. Все дело в том, что они незаметны и не являются очень критическими, а к тем действиям, которые могут быть критическими, ее ведь просто не допускают - их делают дипломированные врачи. В нашей повседневной жизни мы совершаем массу действий, не являющихся критическими и поэтому не замечаем ошибок. И действительно, в любом деле между полным успехом и полным провалом есть еще огромный спектр возможных результатов, включающий в себя все виды нашей естественной человеческой неаккуратности.

Если Вы во время ходьбы споткнетесь, то Ваш мозг автоматически даст команду на корректировку - он ведет постоянный контроль за положением Вашего тела. И только когда Вы ведете работу, не осуществляя такого постоянного контроля - тогда и начинаются все неприятности. Хотя даже и в этом случае рядом может оказаться кто-то другой, кто заметит ошибку и исправит Вас.

Совсем по-другому обстоят дела в программировании, ведь компьютеру все равно правильную или неправильную команду он встречает. Он все равно старается сделать все, что может, чтобы ее исполнить и не заботится о последствиях. Мы по-человечески закладываем в программы массу естественных ошибок, а компьютер слепо все это обрабатывает.

Как правило, ошибки в программировании имеют отдаленные последствия. Время между совершением ошибки и ее проявлением измеряется часами, днями и даже годами, но суть та же. Все мы делаем ошибки постоянно. Разница в том, что ошибки программиста имеют отложенное действие, как бомба с часовым механизмом.

Можно ли перестать делать ошибки? Нет, быть совершенством невозможно. Единственное, что можно сделать - постараться свести их количество к минимуму.

Путь к уменьшению количества ошибок лежит через дисциплину программирования и самоконтроль, а дисциплина и свободное творчество находятся в постоянном конфликте.

Вопрос исключения ошибок, таким образом, сводится к самоконтролю, а самый лучший способ - это расставить ловушки на всем пути. Причем если Вы думаете, что надо исключить ошибки только в программировании, то заблуждаетесь. Их надо исключить еще раньше - уже на этапе проектирования программы. Согласитесь, что нет ничего обиднее, чем узнать, что вся Ваша концепция порочна и неверна. Конечно, мы не можем охватить все случаи, в которых могут возникнуть ошибки, но на наиболее ответственные этапы мы все-таки укажем.

Ошибки в проекте.

1. Размер программы.

Подготавливая проект, оцените размеры всех входящих в него данных: файлов, таблиц, буферов и т.п. Оценить размер программного блока - сложнее, но надо опираться на собственный предыдущий опыт и постоянно контролировать процесс. Попробуйте составить список подпрограмм, которые войдут в Вашу программу. Обязательно оцените средний размер памяти, отпущенный Вам на среднюю подпрограмму.

Когда будете программировать, то постоянно сверяйтесь - укладываетесь ли Вы в этот размер.

2. Быстродействие программы.

Часто скорость является критическим фактором, особенно это относится к играм, имеющим движущиеся объекты. Оцените время, необходимое для выполнения наиболее критических с этой точки зрения операций, например операции по перестроению изображения на экране. Прикиньте, сколько кадров в секунду Вы можете достичь, проверьте себя на простейшем тесте.

Если что-то не получается - подумайте о том, чтобы использовать другие команды процессора или другие методы адресации. Если и это не помогает - может быть можно уменьшить экранную область? Вы же ведь видели не раз, как в программах например используется для динамической графики только верхняя треть экрана, а остальное поле занято статическим, не изменяющимся рисунком.

3. Концепция программы.

Проверяйте концепцию программы на простом примере на БЕЙСИКе, можно это делать и вообще на листе бумаги. Попробуйте объяснить суть задуманной Вами программы кому-нибудь и подумайте над высказанными Вам соображениями. Представьте себя на месте будущего пользователя Вашей программы и задайте себе вопрос: "А что бы я сделал, если бы захотел сыграть в эту игру не по правилам, заложенным программистом?".

4. Цели программы.

Очень часто (особенно если это не игровая, а деловая или прикладная программа) проработка концепции еще не ведет к успеху. Надо абсолютно точно знать, чего же Вы хотите достичь. Прежде, чем начинать проработку проекта, надо составить список целей, для которых создается программа, и, по мере развития проекта, постоянно с этим списком сверяться.

5. Интерфейс "человек-компьютер".

Если Вы предполагаете использовать в программе новый для Вас метод управления, его необходимо проверить на модели. А не слишком ли он сложен? Сможет ли его освоить и применять с удовольствием средний пользователь?

6. Структура программы.

Проверьте всю структуру программы, пройдясь по ней с карандашом в руках вдоль и поперек. Расскажите ее кому-нибудь из знакомых. Объясните, как все работает и что за чем следует. Обязательно постарайтесь получить вопрос типа: "А что будет, если... ?", - и ответьте на них. Проиграйте всю программу в уме, следя за блоками структурной диаграммы. Постарайтесь в этот момент стать глупой машиной.

Наметьте для себя точки в программе, в которых устанавливаются основные переменные и в которых они изменяются.

Ошибки в программировании.

Я выписал 10 наиболее часто встречающихся ошибок в программах, написанных в машинном коде. Они расписаны по убыванию вероятности их появления. Есть прямой смысл иметь этот список перед глазами. Если что-то у Вас идет не так, 90%, что Ваша ошибка есть в этом списке.

1. Переменная или регистр процессора не установлены так, как это надо перед

началом операции.

2. Перепутаны условия перехода и т.п., например, написано JR C вместо JR NC.

3. В операциях с регистрами процессора перепутаны операнды, например: LD A,B вместо LD B,A

4. Перепутаны данные и адреса. Например LD HL,1000 вместо LD HL,(1000)

5. Неверно обрабатывается счетчик цикла. Всякий счетчик должен быть:

- установлен;
- вызван;
- изменен;
- сохранен.

6. Не рассмотрен "нулевой" вариант. Не учтено, что будет, если переменная или счетчик будут равны нулю. А если пользователь по запросу введет нуль?

7. Регистр процессора или переменная используются для какой-то второй цели, а их содержимое не сохранено. Наиболее часто это происходит, когда подпрограмма вызывает другие подпрограммы и т. д. или когда Вы вставляете внутрь своей программы новые строки.

8. Арифметические ошибки. Особенно часто возникают, когда в операции участвуют 16-битные и 8 битные числа.

9. Взаимодействие между процедурами. Должно быть строго определено что и откуда процедура принимает и что и куда она возвращает.

10. Перепутаны шестнадцатеричные и десятеричные числа.

Пути сокращения количества ошибок.

1. Сначала проектирование - программирование потом.

2. Структурируйте программу.

3. Документируйте программу (процедуры и переменные).

4. Старайтесь писать как можно меньше машинного кода. Максимально используйте свои старые процедуры, копируйте похожие куски программы.

5. Пишите в одной и той же манере. Например, организуйте циклы одним и тем же способен и изменяйте счетчик цикла в одном и том же месте.

6. Будьте проще. Пишите простой код. Не пользуетесь без особой нужды головоломными приемами.

7. Без нужды не трогайте стек.

8. Новые процедуры сначала расписывайте на бумаге.

Одним из неудобств является то, что на экране Вы можете одновременно видеть лишь очень малый кусок программы. Я считаю, что программисту, работавшему над крупным проектом, принтер необходим. После написания новой процедуры я ее распечатываю и делаю ее "прогон" в уме. Конечно, все возможные варианты ее работы отследить невозможно, но по крайней мере первый прогон по ней и последний я анализирую обязательно.

В вычислительных операциях я прикидываю минимальное и максимальное возможные значения.

В операциях сдвига и в операциях с битами приходится на бумаге рисовать карту битов в байте. Конечно это я делаю не всегда, а когда компьютер не желает исполнять свеженаписанную процедуру.

Как-то раз я написал процедуру, которая сразу заработала. Я был ошеломлен настолько, что не поверил сам себе. Два часа я разбирал ее на части в уверенности, что так не бывает. И нашел в ней такое ... - самую настоящую мину с часовым механизмом, которая только ждала своего часа.

Для тех, кто пишет на БЕЙСИКе.

Как найти ошибку в БЕЙСИК-программе? Что делать, если программа отказывается работать.

1. Начните искать ошибку в строках, которые Вы изменяли последними. Обычно

ошибка в них.

2. Постарайтесь локализовать место ошибки путем запуска не всей программы, а только ее части. Ее начало определит команда RUN n или GO TO, а конец можете обеспечить вставкой STOP.

Начинайте это делать с достаточно большого блока и уменьшайте его постепенно.

3. Если в программе есть или создаются массивы данных, то можете написать маленькую программку для проверки этих массивов. То ли в них содержится, что должно быть.

4. Оговорите суть проблемы со своим другом. Обычно, когда Вы умеете хорошо объяснить суть проблемы, то и решение уже рядом.

5. Попробуйте вжиться в проблему настолько, чтобы засыпать с нею в уме. Нередко утром Вас осеняет простое решение. Конечно это стоит делать только для особо важных и сложных проблем.

6. Распечатайте листинг проблемной области. Прокрутите работу этого блока всухую (в уме). Наметьте точки, в которых Вы вставите STOP (точки прерывания). Когда программа встанет в точке прерывания, прямой командой PRINT проверьте значения важнейших переменных.

А что делать, если программа не Ваша, если Вы набрали ее из распечатки в журнале, и она не работает?

Надо сразу сказать, что проблема эта довольно типичная. Большинство ошибок возникает при наборе, многие вызваны непропечаткой символов, а есть и такие, которые заложены уже в приведенный листинг. Для всех категорий ошибок есть масса причин, вызывающих их появление, и это конечно проблема, но решение ее - довольно полезная задача. Все программисты знают, что настоящий профессионализм приобретается не тогда, когда находишь собственные ошибки, а тогда, когда чужие.

1. Запускайте программу по частям, используя оператор STOP и проверяя переменные после остановки. Если все в порядке, запускайте программу дальше оператором CONT.

2. Найдя строки, в которых происходит ошибка, проверьте правильность своего набора. Обратите внимание на соответствие прописных и строчных букв, на похожесть буквы I и цифры 1, а также буквы B и цифры 8. Постарайтесь понять, что делает каждый оператор и проследить из каких других частей программы могла прийти ошибка вместе с какой-либо переменной. Если Вы поймете, что делают строки программы, то сможете обнаружить даже и те ошибки, которые содержатся в приведенном листинге.

3. Особое внимание обратите на ошибки, связанные с DATA и с POKE - их наиболее трудно отыскивать, часто здесь приходится не просто проверять правильно ли Вы набрали текст программы, но и разбираться с тем, что и для чего они вводят.

4. Внутри циклов используйте оператор PRINT, чтобы следить за ходом изменения параметра цикла и основных переменных.

5. Ищите улики. Пробуйте разные значения в INPUT и следите за тем, какие подпрограммы их обрабатывают. Старайтесь точно определить условия, в которых они начинают неправильно работать.

Пробный запуск.

Несмотря ни на какие предпринимаемые меры предосторожности, некоторые ошибки тем не менее все же доживают до тестового запуска процедуры. Тестирование должно быть систематический. Проводите его малыми порциями. Когда компьютер "зависнет", Вы будете знать, где собака зарыта.

Постепенно наращивайте размер программы путем "прикрепления" новых блоков к уже проверенным и надежно работающим. Если какой то блок не зависит от работы прочих - испытывайте его в первую очереди и только потом проверяйте его в составе всей программы.

Точно так же я работаю и на БЕЙСИКе, только там я запускаю группы строк, проверяя содержимое переменных.

Пользовательская проверка.

Этот этап служит не столько для отыскания ошибок, сколько для проверки соответствия ее конечным целям. Кроме того, очень часто другие пользователи делают такие вводы в программы, которых программист и не предполагал и не предвидел. И это позволяет вскрыть еще немало ошибок.

На этом этапе я провожу еще и окончательную настройку программы, чтобы достичь наилучшей "играбельности".

Последний барьер.

Когда моя игра готова к выпуску, я провожу еще одну - последнюю проверку.

Я читаю всю программу в распечатке, процедуру за процедурой. Это чтение на понимание. Я должен убедить себя в том, что понимаю, что делает каждая команда. При этом обычно вскрывается и еще какая-нибудь шальная ошибка. Далее уже слово за публикой.

План тестирования.

Чтобы правильно выполнить тестирование написанных процедур, нужна систематичность. Лучше всего для начала подготовить план.

Каждому тесту дается свой справочный номер, рядом с которым выписываемся список данных, подлежащих проверке. Очень часто эти данные - это то, что вводит пользователь при работе с программой, например команды, поступающие от джойстика.

Можно быть еще более систематичным и расписать возможные условия, в которых будет работать та или иная процедура. При этом можно использовать распечатку программы, пометив из ней все группы команд, находящиеся между условными конструкциями типа JR Z, DJNZ, JP C и т.п.

Чтобы тест был всеобъемлющим, определите условия, при которых те или иные блоки должны работать. Когда же дело дойдет до реального теста, большинство возможных тестов Вы вычеркнете из своего листинга без необходимости специально задавать условия и прогонять блоки. Останутся самые необходимые пункты, которые нельзя упустить.

Убедитесь, что каждый из намеченных к тестированию блоков реально выполняет свои функции. Это не значит, что достаточно посидеть и посмотреть, как он работает. Решите заранее для себя, в каких точках Вы будете прерывать работу процедуры и проверять то ли содержится в регистрах процессора, что там должно быть и те ли значения имеют переменные, какие положено. Конечно, не каждую группу команд стоит так проверять. Это можно оставить для наиболее головоломных. А вот конец каждой процедуры - это вполне удобное место, в котором стоит это сделать.

В следующем выпуске я приведу образец плана тестирования программы, а также приложу распечатку собственной программы MONITR (Real Time Monitor - монитор, работающий в режиме реального времени), с помощью которой очень удобно делать многочисленные проверки.

(Продолжение следует)

Слово эксперту

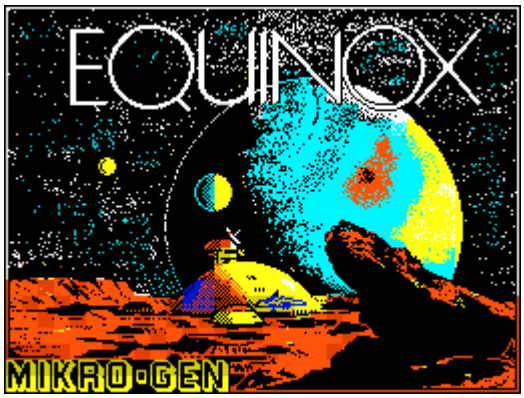
С большой радостью мы отмечаем, что не успели мы объявить атаку на программу EQUINOX, как получили ответ от эксперта, сделавшего на наш взгляд великолепную проработку.

Так держать, эксперты!

EQUINOX

(РАВНОДЕНСТВИЕ)

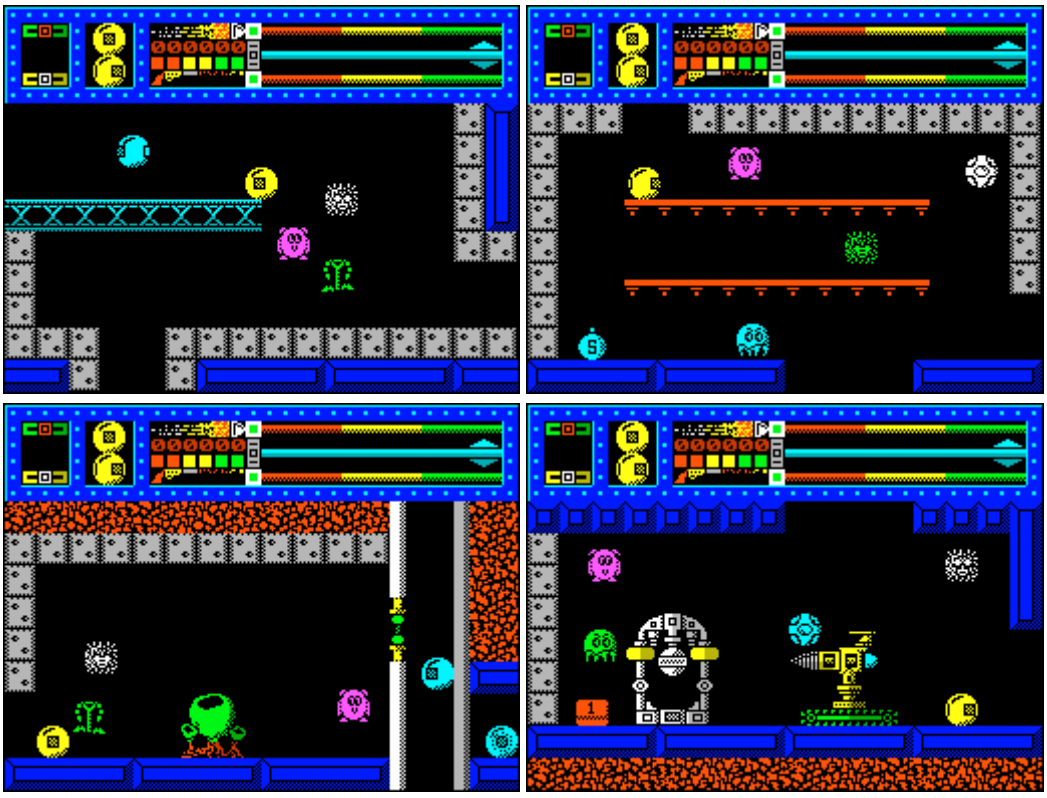
Mikro-Gen 1986г.



Эксперт Кочнев С. В.

г. Москва.

Программа, строго говоря, относится к жанру аркадных адвентюр, но учитывая необходимость постоянно отбиваться от противника, справедливо будет определить ее жанр как ARCADE/ADVENTURE/ACTION. Недюжинная реакция и умение метко стрелять здесь понадобятся не менее, чем способность мыслить логически, поэтому эту игру можно рекомендовать и любителям боевиков.



Программа оснащена очень высококачественной графикой и, хотя общее число локаций всего 128 экранов, не считая комнат межуровневой телепортации, что по сравнению с некоторыми программами не так уж велико, отличная графика с лихвой

компенсирует этот недостаток. В программе задействовано немалое число технических устройств. Одни работают бесплатно, другие - только после уплаты соответствующей суммы. Такое разнообразие делает программу одинаково привлекательной для любителей разных жанров.

Действие происходит в "далекой-далекой" Галактике, на астероиде, ставшем луной обитаемой планеты. На спутнике велась добыча жизненно важной для обитателей этой планеты руды, но внезапно, в день равноденствия, наступающего здесь раз в несколько тысяч лет, в недрах астероида проснулось страшное животное, занесенное вместе с астероидом из соседней звездной системы.

Чудовище выпустило на поверхность разработок свои органы чувств и оказалось способным силой своего воображения создавать фантомов, способных убивать. Оставшиеся в живых рабочие бежали, бросив шахты. Вооруженные Силы планеты в состоянии уничтожить чудовище, применив ультразвуковое оружие, но бежавшие рабочие оставили в спешке канистры с ядерным топливом, применявшимся в силовых агрегатах машин, неубранными. Поэтому применение оружия грозит взрывом всему астероиду.

Положение осложняется еще и тем, что ядерные канистры сами по себе неустойчивы и взорвутся, если их вовремя не поместить в специальную камеру.

Ваша миссия: ликвидировать опасность взрыва. Управляя дроидом, Вы должны найти на каждом уровне шахты ядерную канистру и в установленное время по пневмолиниям отправить ее в специальную камеру хранения.

Настройка программы.

Сразу после загрузки программы Вы получаете главное меню:

1. DEFINE KEYS - задание клавиш.
2. INSTRUCTION - инструкция.
3. START GAME - запуск игры.
4. JOYSTICKS - джойстики.

Если Вы будете работать на клавиатуре, то для выбора удобных для Вас клавиш вызовите опцию главного меню "DEFINE KEYS".

Получите следующее меню:
SELECT KEYS FOR:
(выберите клавиши для:)
LEFT - движение влево
RIGHT - движение вправо
THRUST - включение двигателя
USE - пользование предметами
FIRE – огонь
PAUSE - пауза

Если Вы используете джойстик, то вызвав опцию "JOYSTICK", попадете в меню выбора органа управления:

1. KEYBOARD - клавиатура
2. KEMPSTON - кемпстон-джойстик
3. INTERFACE 2 - интерфейс-2, т.е. синклер-джойстик (правый).

Функции "USE" на джойстике соответствует нажатие рукоятки "вниз".

Следует иметь в виду, что если поиграв на джойстике, Вы решили вернуться на клавиатуру, то клавиши Вам придется задавать заново. Если же Вы сразу решили использовать клавиатуру, то выбирать ее опцией "JOYSTICK" не надо.

Знающие английский могут вызвать на экран краткую инструкцию к программе. Для возврата к главному меню нажмите любую клавишу.

Подождите пока запускать игру, лучше послушайте музыку (если не хотите, нажмите любую клавишу, кроме управляющих главным меню). По ее окончании программа

последовательно выдает на экран внешний вид и краткое описание основных видов технических устройств, с которыми Вам предстоит столкнуться. Здесь мы даем инструкции к этой технике в порядке вывода их компьютером на экран.

TELEPORTER TERMINAL

Система местной (т.е. в пределах данного уровня) телепортации, состоящая из пары устройств, бесплатно работать не будет. Поэтому сначала следует отыскать и опустить в телепортер монету, для чего следует поместить дроида внутрь устройства и нажать клавишу "USE". Звуковой сигнал свидетельствует о приеме монеты.

Для телепортации необходимо совместить дроида с шаром, для чего следует включить двигатель. Вы окажитесь у второго телепортера. За одну монету Вы можете дважды поменять свое местоположение (локацию). Однако имеется одна тонкость: системе безразлично, будете ли Вы телепортироваться "туда-обратно" или два раза "туда" (есть и такие хитрые лабиринты). В любом случае за одну монету она обслужит Вас дважды. Вы можете опустить в один из терминалов и большее число монет. При этом соответственно возрастет и количество возможных телепортаций. Во время местной телепортации Вы можете брать с собой любые предметы.

Примечание: не следует пытаться опускать в терминал батарейки, ключи, динамитные шашки и прочие несвойственные монетоприемнику предметы - устройство крепкое, выдерживало и не такое.

TRANS-LEVEL TELEPORTER

Устройство межуровневой телепортации. Вход только при наличии специальной магнитной карточки. В комнате телепортера слева вверху находится индикатор уровня шахты, на котором Вы в данный момент находитесь. В центре, на стене, расположены клавиши с цифрами, обозначающими номера уровней. Для телепортации на нужный Вам уровень необходимо подлететь к соответствующей клавише и выполнить функцию "USE". Раздастся звуковой сигнал и индикатор укажет, на каком уровне Вы теперь находитесь. Можно выходить.

Следует, однако, иметь в виду, что если номер выбранного Вами уровня окажется больше числа, написанного на карточке, то устройство работать откажется. Таким образом, если Вы на первом уровне войдете внутрь с карточкой N1, то телепортироваться вообще никуда не сможете.

Т.к. Вы можете иметь при себе только один предмет, а для входа в TRANS LEVEL TELEPORTER необходимо иметь при себе магнитную карточку, то ничего, кроме нее перенести на другой уровень Вы не сможете.

Топливо во время перемещения в комнате телепортера не расходуется, время не останавливается. Лазер во избежание несчастного случая или умышленного вредительства блокируется.

CREDIT DISPENSER

Дословно - распорядитель кредитов.

К делу не относится, но любопытно: в игровой технике (мы не имеем в виду "Морской бой" и прочие шедевры, сделанные видимо специально, чтобы не возбудить в советском человеке жажду крови) количество опущенных в автомат монет индицируется как "CREDIT". Есть, правда, в зависимости от достоинства монеты и другие соотношения, скажем 1 COIN - 1/2 CREDIT и другие. В любом случае, CREDIT - это число игр, оплаченных заранее. У нас это вполне прижилось и дожидающиеся своей очереди "профи", скажем у имитатора мотогонок "SUPER HANG-ON" (кстати его автор - фирма "SEGA" отлично конвертировала для SPECTRUMa самую длинную и сложную из трасс - PART AMERICA) спросит у играющего: "Сколько кредитов поставил?" В нашем случае, будь у системы TELEPORTER TERMINAL дисплей, то опустив в ее монетоприемник пару монет, мы бы увидели сообщение:"CREDIT:4", что соответствует четырем телепортациям.

Не верьте названию этого странного устройства! Просто так "раздавать кредиты" он не будет. CREDIT DISPENSER меняет монету на жизнь вашего дроида! Вообще, на каждом уровне находится только минимально необходимое число монет для выполнения данной миссии. Поэтому если Вы совершите ошибку, Вам будет просто нечем платить системе телепортации. В этом случае ничего другого не остается, кроме как обменять одного из дроидов на монету. Для этого надо опустить дроида на горловину аппарата (при этом у Вас не должно быть с собой никаких предметов) и нажать клавишу "USE". Если это была последняя попытка, игра прекратится. Старайтесь не делать ошибок в ходе игры, тогда и пользоваться CREDIT DISPENSER'ом не придется. Кроме того, он имеется не на всех уровнях. Если Вы попадете в такую ситуацию, Вам останется только выйти из игры, нажав "CAPS SHIFT" + "BREAK".

DISPOSAL CHUTE

Распределительное устройство вакуумной системы сообщения, пронизывает все уровни шахты. В нашем случае может служить для отправки ядерных канистр в хранилище. Работает бесплатно. Чтобы воспользоваться DISPOSAL CHUTE необходимо ввести Вашего дроида (разумеется с канистрой на борту) в приемник устройства и выполнить функцию "USE". Вы увидите прибытие контейнера в камеру хранения. Вход в приемник возможен как слева, так и справа, поэтому если труба пневмолинии преградила Вам путь, пересекайте ее в месте приемного устройства.

MAGNETIC LIFT

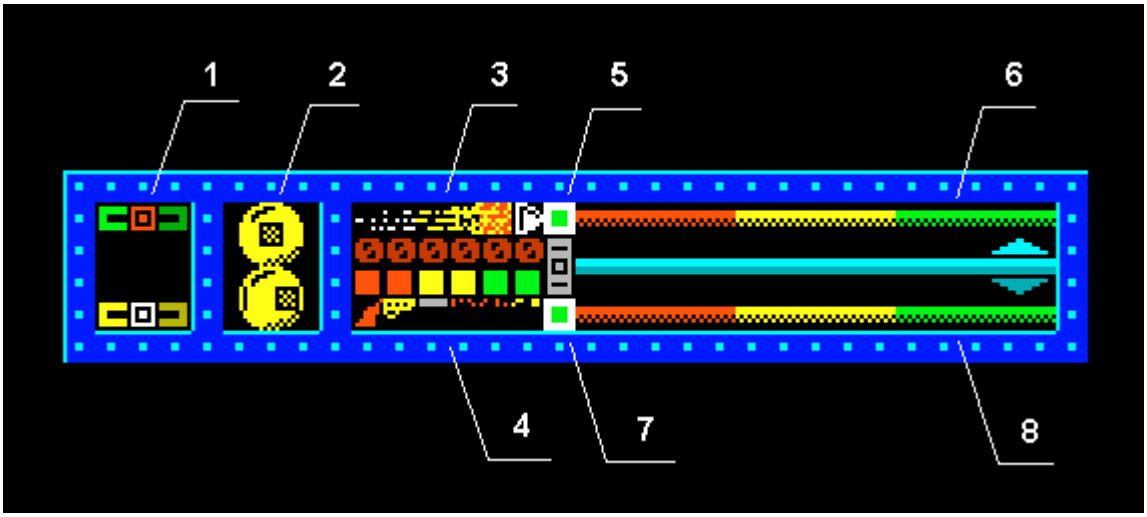
Остроумно придуманная система подъема металлических тел, состоящая из постоянного магнита и металлического контура, образующего шахту лифта и предназначенного для поддержания силовых линий магнитного поля вдали от его источника, а также экранизации магнита (без этого он бы действовал на все металлические тела, в том числе и на не находящиеся в шахте лифта). В контуре на каждом из этажей, которые пересекают лифт, сделаны разрезы для входа и выхода. Силовые линии, пронизывающие контур, настолько мощны, что в этих местах наблюдается непрерывный электрический разряд. Магнитный лифт начинает свое действие сразу после входа дроида внутрь шахты. Вам остается только не прозевать нужный этаж. Кроме несомненных положительных свойств - экономия топлива и времени, у лифта есть и очень серьезный недостаток: работать он может, разумеется, только в одну сторону. Это не имеет значения на первом уровне, но на остальных уровнях лабиринт построен так, что если Вы по ошибке неправильно им воспользуетесь, например забудете захватить нужный предмет, то за обратный путь Вы можете заплатить не только временем, топливом и энергией лазера, но и деньгами. Отметим также, что пересечь шахту лифта в горизонтальном направлении, кроме ее верхней точки, возможно только в одном направлении, так как подъем начинается сразу после входа в шахту и поэтому входы в MAGNETIC LIFT расположены на разной высоте. Иначе говоря, магнитный лифт обладает односторонней проходимостью как в вертикальном, так и в горизонтальном направлениях.

Если эти объяснения показались Вам слишком запутанными, посмотрите на данный ниже план второго уровня, и Вы сразу все поймете.

Работу технических устройств, а также работу с предметами мы рассмотрим ниже, параллельно с разбором методов прохождения некоторых уровней.

Экран программы.

После запуска программы Вы увидите сверху на экране таблицу, отражающую основную текущую информацию.

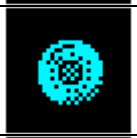


где:
1 - содержимое грузового отсека;
2 - количество оставшихся у Вас дроидов;
3 - очки;
4 - время;
5 - индикатор включения двигателя;
6 - указатель уровня топлива;
7 - индикатор включения лазера;
8 - указатель состояния энергетического отсека.

Стрелки на индикаторах топлива и энергии, по мере их расхода, движутся справа налево, т.е. крайне правое положение соответствует их максимальному запасу.

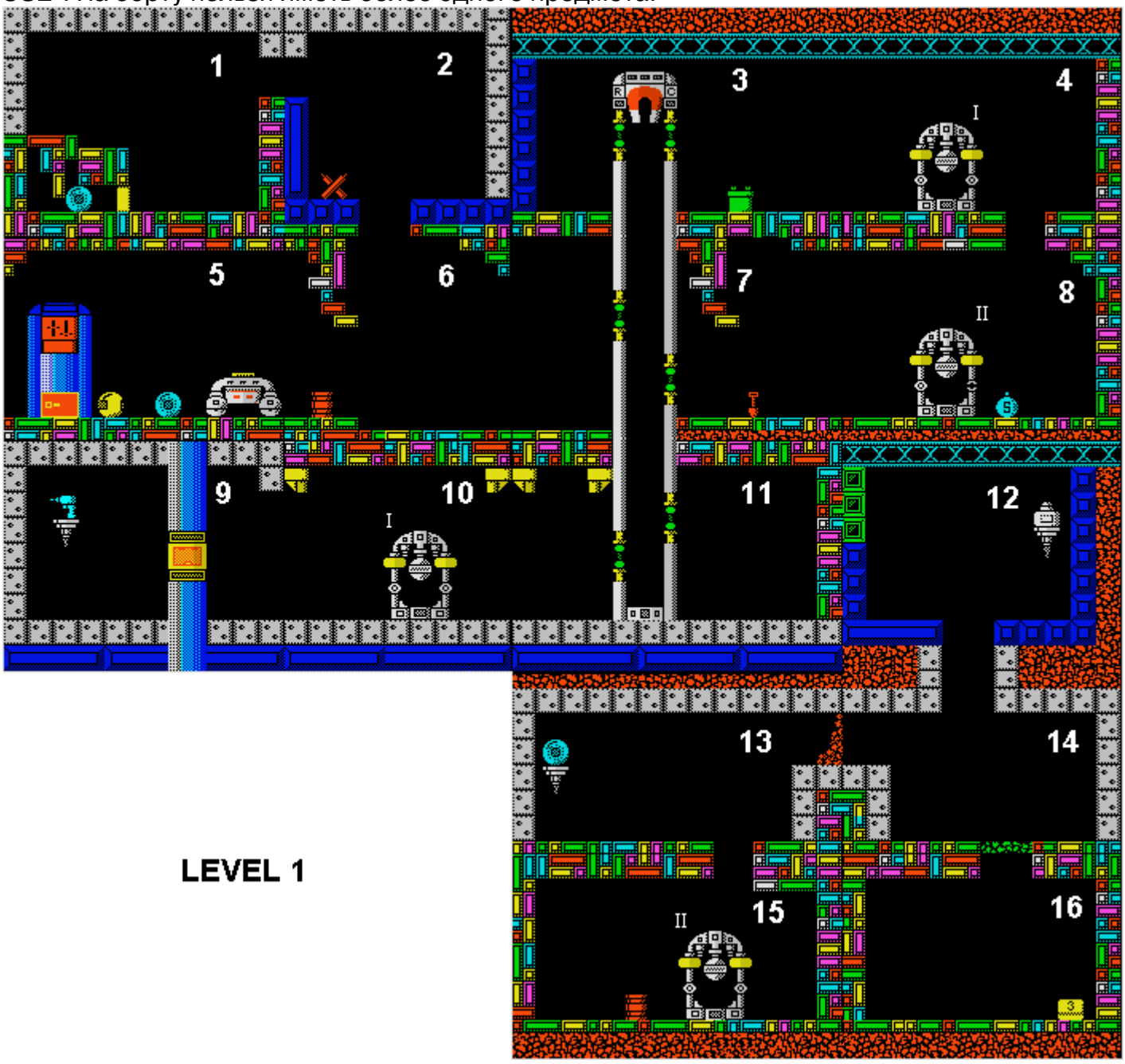
Работа с предметами и описание некоторых уровней.

Сначала рассмотрим основные предметы, с которыми Вам предстоит работать на всех уровнях, а также приведем их условное обозначение, т.е. дадим легенду к нашим картам.

Обозначение	Название	Назначение
	Ядерная канистра (ЯК)	Опустить в приемник DISPOSAL CHUTE
	Магнитная карточка (МК)	Вход в межуровневый телепортер. Номер указывает на максимальный уровень телепортации.
	Монета (М)	Оплата работы TELEPORTER TERMINAL'а
	Батарейка	Подзарядка энергетического отсека
	Канистра с топливом	Заправка двигателя
	Дрель (Д)	Вскрытие сейфов

	Динамитная шашка (ДШ)	Расчистка проходов
	Граната	Уничтожает всех фантомов в комнате
	Ключ (К)	Отпирание дверей
	Предохранитель (ПР)	Отключение систем электронной защиты

Примечание: Для захвата, обмена и использования всех предметов служит клавиша "USE". На борту нельзя иметь более одного предмета.



LEVEL 1

Мы умышленно не даем план первого уровня, так как он достаточно прост для того, чтобы Вы потренировались на нем сами.

В правом верхнем углу каждой комнаты находится число, соответствующее ее

порядковому номеру. Нумерация, разумеется, чисто условная и служит для упрощения записи плана прохождения уровня. Символы, обозначающие технические устройства, достаточно похожи на их внешний вид, поэтому мы не стали делать для них пояснительную таблицу.

Выйдя из TRANS-LEVEL TELEPORTER'а в комнате 5, возьмите монету, двигайтесь 6-7, далее на лифте -3-4, опустите монету и телепортируйтесь -10-9, возьмите дрель, -10-11-7 (выход налево, не прозевайте, иначе придется возвращаться /3-4-8-7/) -6-2-1, использовав дрель, достаньте монету, -2-6-7-3-4-8. Опустите монету, но не телепортируйтесь (кстати, обратите внимание на невозможность 6-7-8, это и есть односторонняя проходимость) 7-6 (а так можно!) -2, возьмите динамитные шашки, -6-7-3-4-8,Т,-15-13, расчистите взрывчаткой завал, возьмите монету, -15, опустите монету, -13-14-12, возьмите канистру, -14-13-15 (Внимательно следящие наверное уже отметили, что на данный момент на системе TELEPORTER TERMINAL (I) можно произвести 1Т, а на системе TELEPORTER TERMINAL (II) – 3Т). Т, -8-7-3-4, Т, -10-9, опустите канистру в DISPOSAL CHUTE. На этом Ваша миссия на втором уровне считается выполненной. На шкале времени появляется сплошная полоса, т.е., здесь опасность взрыва ликвидирована. Но Вам нужно идти дальше, для этого нужна магнитная карточка с номером следующего уровня. -10-11-7, возьмите ключ, -8,Т,-15-13-14. Здесь мы встречаемся с нестандартным применением ключа - в качестве ершика для прочистки горизонтального прохода. -16, возьмите карточку, -14,-13-15,Т,-8. На этом Ваш кредит на обеих системах телепортации иссяк (можете проверить), но они нам больше не нужны. Возвращайтесь к TRANS-LEVEL TELEPORTER'у: -7-6-5 и входите.

Как видите, пройти уровень не так уж и просто, есть над чем поломать голову. Здесь мы не давали указаний по поводу использования батареек и бочек с топливом: их, а также гранату, применяйте по собственному усмотрению.

Уровень 3

Здесь мы встречаемся с тремя новыми устройствами - транспортером (7-8-9), односторонним проходом (1-4) и системой силовой защиты (1). Для снятия последней в ее щит необходимо вставить предохранитель (ПР). Т.к. подробное описание метода прохождения занимает довольно много места, а Вы уже имеете довольно полное представление о работе технических устройств, то в подробных комментариях больше нужды нет. Перейдем к сокращенной записи. Символ ""[1] будет обозначать "взять", "" - "использовать" и "" - "обменять". Отметим теперь касательно транспортера: он является горизонтальным аналогом лифта, ехать на нем можно только в одном направлении, перейти в соседнюю комнату можно только по ходу движения.

Способ прохождения:

12-11-10-7-4 .. К-7-10-11-14-.. К 15 .. Пр 14-11-10-7-4-5-2-1..Пр .. М-4-7-10-11-14-13-16-.. М; Т-3.. ЯК 6-9-8-7-10-11-14-13 .. ЯК-16 Т-3-6 .. МК-9-8-7-10-11-12

Как видите, запись получилась достаточно компактной. Постарайтесь предложить свой алгоритм, хотя бы не намного отличающийся от данного. Сделайте то же для второго уровня.

Уровень 4

Здесь Вам встретятся прессы (комнаты 1 и 8). От Вас им ничего не нужно, постарайтесь только под них не попасть.

Разобрав предыдущие примеры, Вы можете найти алгоритм прохождения этого уровня самостоятельно. Сделайте это и сравните с алгоритмом данным нами. Не спутайтесь, если внешне они будут отличаться. Принцип всегда будет одинаков.

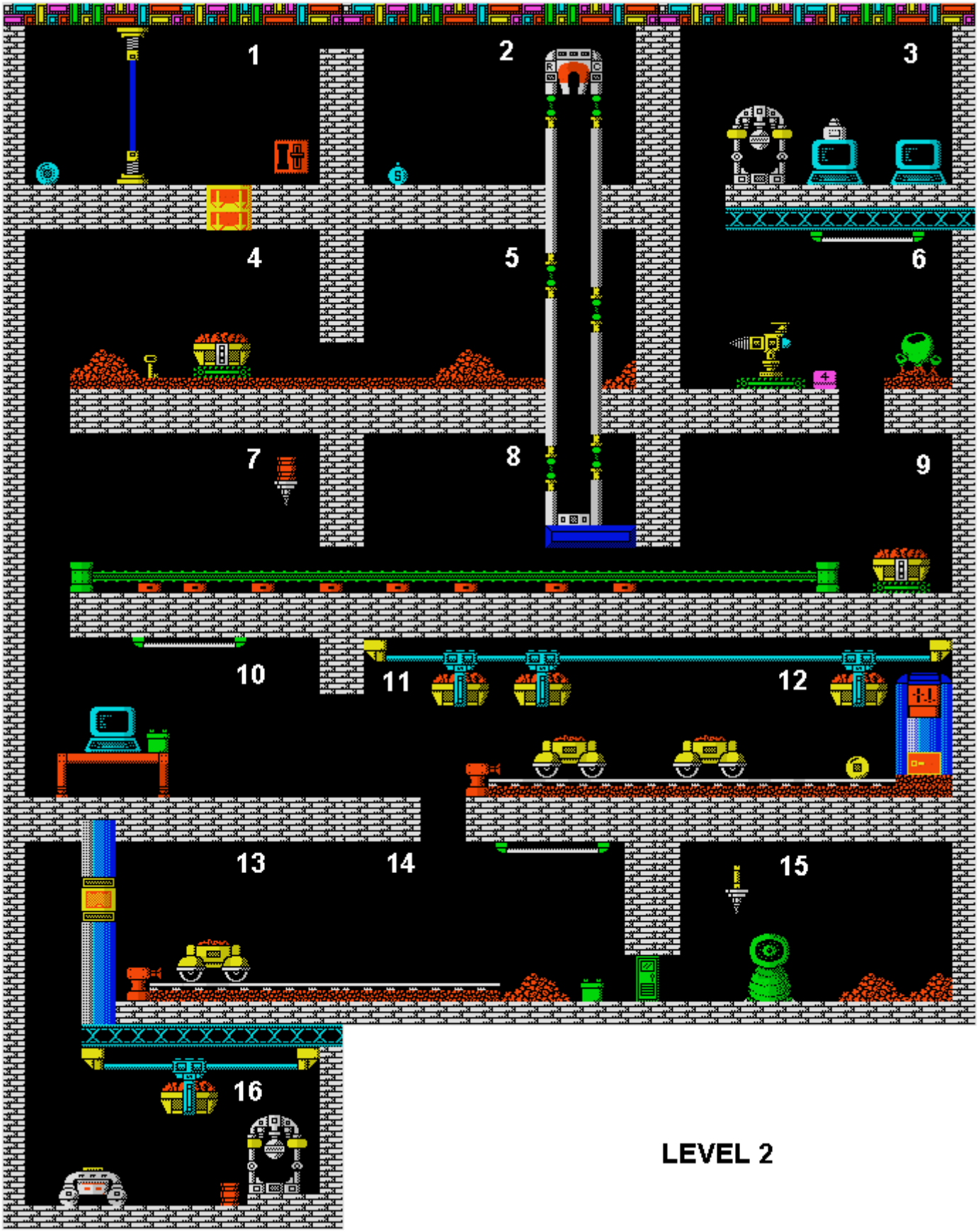
Способ прохождения:

6 ... М-5-4-1-2 ... М-3-6-5 ... М-4-1-2 .. М-3-6-5-1 ... Пр-1-2 Т-7-8 Пр.. Д-7 Т-2-3 ..

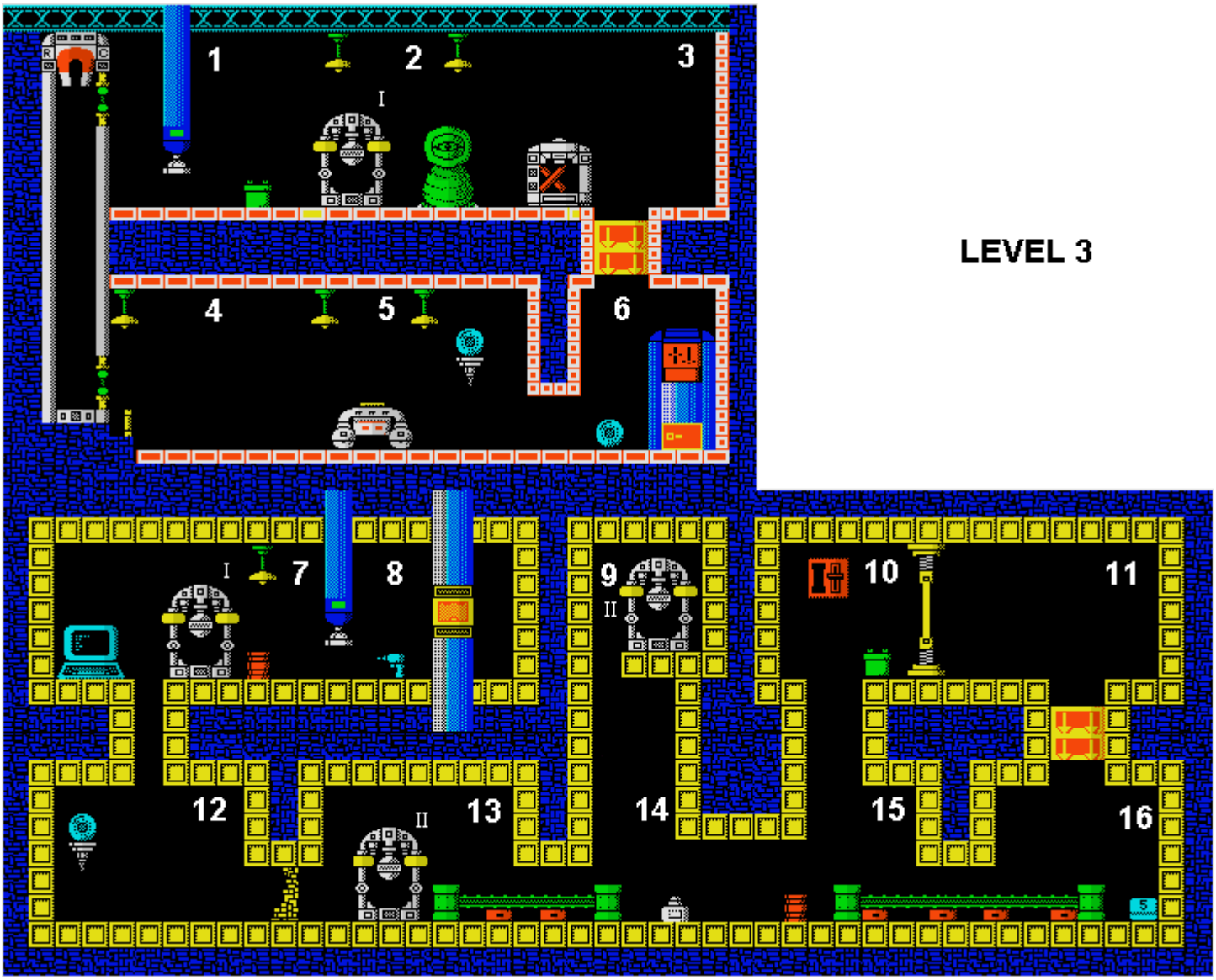
[1] В оригинале пробелы. (Прим. NUK)

Д; .. ДШ-2 Т-712 .. ДШ; .. М-4 13 .. М; Т-9-14 .. ЯК-13-12-7-8 .. ЯК; .. Пр-7-12-13 Т-9-14-15-10 .. Пр-11-16- .. МК-15-14-13-12-7 Т-2-3-6

Не имея желания лишать Вас удовольствия самостоятельно пройти оставшиеся уровни (а половину мы уже разобрали), рассматривать мы их не будем.



LEVEL 2



Полезные советы

У игры EQUINOX есть очень интересное свойство. Впрямую мы о нем не сообщали, но если Вы сами догадались о сказанном ниже, значит не зря потратили время на изучение этого описания и нестандартное мышление Вам не чуждо. Дело в том, что в этой игре возможен переход на следующий уровень без выполнения предыдущего, ведь для этого нужна только карточка с соответствующим номером.

Это очень облегчает работу, ведь, как Вы уже успели понять, сразу программу не пройти, прежде надо разработать план действий. Мы рекомендуем Вам сначала не выполнять миссии ни на одном из уровней, этим Вы сэкономите свое время, а возможно и попытки. На первоначальном этапе Вы должны составить точные планы всех уровней, здесь Вам поможет клавиша "PAUSE". Следующий этап - чисто аналитический - разработка алгоритма прохождения. Кстати, Вам понадобятся два алгоритма: для простого перехода на следующий уровень и для выполнения миссии. После этого можете возвращаться к игре и спасать астероид.

Топлива и энергии, как правило, хватает, но все же старайтесь их использовать рационально. Например, если Вы видите, что Вам скоро потребуется заправка, на борту у Вас ничего нет, то взятую бочку не надо использовать сразу! Продолжайте движение, клавишу "USE" нажимайте только тогда, когда кончится топливо или потребуется место в грузовом отсеке, ведь содержимого канистры хватает, чтобы полностью заправить топливный отсек, и его избыток будет просто выброшен. Это справедливо и для батареек. Если у Вас на борту что-то есть и необходима дозаправка, то подлетите к канистре, обменяйте ее на Ваш предмет, отлетите от этого места и используйте канистру, после чего заберите предмет.

Выбросить ни один из предметов нельзя, можно либо обменять его на другой, либо использовать. Во избежание тесного контакта с призраками, комнату старайтесь "проскакивать" побыстрее, гранату используйте там, где это затруднительно. И последнее: не пугайтесь, увидев незнакомый предмет или устройство, может быть это только графическое решение?!

GOOD LUCK!

ADVENTURE PROJECT

Расчет характеристик персонажей.

Задание начальных атрибутов Ваших персонажей - это только начало трудного пути установления правил, по которым они будут взаимодействовать друг с другом.

В предыдущей статье (№6, с.122) мы начали рассматривать проект, связанный с похищением экспериментального образца "Часов Вселенной" и остановились там, где Инф, Ора и Ком обнаружили затонувшую яхту, на которой налетчики вывезли уникальный прибор. Инф и Ора ныряют в море и находят, что яхта охраняется двумя чудовищами - Крамом и Боргом. Дело идет к тому, что столкновения не миновать.

Сражение всегда было важнейшим элементом адвентюрных программ, но редко когда удается создать напряженный дух схватки. Помочь приблизить ситуацию к реальной могут те принципы искусственного интеллекта, о которых мы и говорим.

Общий подход в программе состоит из двух этапов:

1. На основе характеристик персонажей рассчитывается желательность схватки. Либо та, либо другая сторона могут в последний момент отказаться от боя.

2. Задаются начальные условия боя:

- в программных переменных устанавливаются имена тех, кто участвует в битве и какими преимуществами они располагают (наличие оружия или магических объектов).

Эти этапы взаимосвязаны. Компьютер рассчитывает желательность схватки, а затем задает ее начальные условия. Например, Инф не станет наносить удара по Оре. Имеет смысл вообще исключить такую возможность. Вы можете, скажем, встроить в программу блок, который будет игнорировать те команды пользователя, которые не согласуются с личными характеристиками персонажей и выдавать на экран какое-то сообщение.

В нашем примере сражение, скорее всего, неминуемо, поскольку проникнуть на яхту герои все-таки должны для успешного выполнения цели игры.

Итак, программа, занимавшаяся имитацией схватки, должна решать следующие 6 задач:

1. Сколько героев вступают в схватку?
2. Сколько противников им противостоят?
3. Соотношения в силе между персонажами.
4. Имеется ли оружие и как оно учитывается.
5. Пути выхода из битвы.
6. Условия победы.

С первыми двумя проблемами из перечисленных мы с Вами разобрались в прошлом выпуске, когда персонажи решали, стоит ли им вступать в бой.

Соотношения боевых характеристик персонажей зависят от их атрибутов, о которых мы тоже говорили в шестом выпуске. Нам необходимы формулы для того, чтобы из исходных атрибутов (эмоции, энергия, сила, интеллект, совесть, образованность) рассчитать производные характеристики, необходимые для битвы, такие, как гнев, мужество, решительность, трусость, самолюбие. Физическая сила может не рассчитываться, а браться из атрибутов.

Гнев, например, может формироваться из эмоций, силы и интеллекта. Мужество из энергии, силы и совести.

Если первичные атрибуты установлены в диапазоне (-10... +10), Инф, Ора, и чудовища могут иметь факторы, приведенные на рис. 1.

Инф не имеет отрицательных параметров и потому очевидно готов к схватке. Оре не так повезло, у нее может быть самолюбие как у львицы, но мужества и силы у нее, как у овечки.

Тем не менее, недостатки Оры могут быть парированы, если мы рассмотрим их противников. Брог и Крам - это пара хоть и могучих, но трусливых и неповоротливых монстров, мало способных выиграть хоть какое-то сражение.

Программа, приведенная на рис. 2 - это упрошенная процедура, описывающая схватку. Входящие формулы, описывающие такие факторы, как гнев или мужество, должны быть рассчитаны в другой подпрограмме, до битвы. Так гнев, например, может рассчитываться по формуле:

LET гнев = (INT(эмоции*интеллект+ + сила)/NOATT) + 1

Формула рассчитывает среднее значение атрибутов, заключенных в скобках. NOATT - количество атрибутов, в нашем случае это 3.

Функция INT служит для округления результата. Тот недостаток, что "Спектрум" всегда при вычислении целой части числа выполняет округление вниз, можно парировать за счет прибавления единицы к полученному результату.

Когда все атрибуты рассчитаны. Вы можете перейти к сценарию битвы. Листинг, приведенный на рис.2, соответствует атрибутам, приведенным на рис.1, которые передаются через массив С, организуемый в строках 1...90.

Группа операторов IF...THEN в строках 110-120 занимается расчетом, имеют ли Инф и Ора достаточно сил для продолжения битвы.

Брог и Крам не имеют такой возможности последнего выбора, но Вы всегда можете изменить программу и поставить столько проверок, сколько хотите.

Переменная N$ в строках 110 и 120 выполняет роль статусной переменной. Она выставляется в строке 2 и до строки 110 является пустой.

Если кто-то из героев ранен и его сила падает ниже –9, его имя помещается в переменную N$ и передается в подпрограмму 2000. Здесь чудовища могут его убить, если их атрибуты им это позволяют.

Теперь рассмотрим действия, выполняемые в строках 160...190. Строка 160 сравнивает гнев, силу и мужество Инфа и Крама. Если показатели Инфа лучше, то он выигрывает раунд и оператор LET в конце строки исполняется таким образом, что его гнев возрастает, а сила уменьшаются, далее управление передается строке 180 для оценки второй пары.

Если Крам выигрывает раунд, то управление переходит на строку 170. где изменяются его атрибуты.

Строка 190 изменяет атрибуты Брога, если Ора проигрывает в схватке.

Когда эти расчеты и изменения произведены, программа возвращается на строку 100, откуда весь процесс вновь повторяется.

Подпрограмма в строке 2000 вычисляет, может ли противник убить героя, сила которого упала ниже -9. Если это произошло, то к имени героя, хранящегося в переменной N$, добавляется символ #.

Атрибут	Инф	Ора	Кран	Брог
Ловкость	5	9	-3	-3
Гнев	3	2	7	7
Самолюбие	5	9	9	6
Интеллект	7	2	4	4
Мужество	7	-4	-2	-3
Сила	4	-3	7	5

Рис. 1

```
1 REM Подпрограмма "СРАЖЕНИЕ"
2 LET N$ =""
5 RESTORE 60
10 DIM C(4,6)
20 FOR k = 1 то 4
30 FOR m = 1 TO 6
40 READ C(k, m)
50 NEXT m: NEXT k
60 DATA 5,3,5,7,7,4:REM ИНФ
70 DATA 9,2,9,2,-4,-3:REM ОРА
80 DATA -3,7,9,4,-2,7:REM КРАМ
```

```
90 DATA -3,7,6,4,-3,5:REM БРОГ
110 IF C(1,6)<-9 THEN PRINT "Инф не может больше сражаться": LET N$="ИНФ": GO SUB 2000
120 IF C(2,6)<-9 THEN PRINT "У Оры нет больше сил": LET N$="Ора": GO SUB 2000
125 LET G$=""
130 IF C(3,6)<-9 OR C(4,6)<-9 THEN GO SUB 3000
140 IF G$="STOP" THEN STOP
160 IF (C(1,1)+C(1,2)+C(1,6)) > (C(3,1)+C(3,2)+C(3,6)) THEN LET C(1,2)=C(1,2)+1: LET
      C(3,6)=C(3,6)-1: GO TO 200
170 LET C(3,2)=C(3,2)+1: LET C(1,6)=C(3,6)-1
180 IF (C(2,1)+C(2,2)+C(2,6))(C(4,1)+C(4,2)+C(4,6)) THEN LET C(2,2)=C(2,2)+1: LET
      C(4,6)=C(4,6)-1: GO TO 180
190 LET C(4,2)=C(4,2)+1: LET C(2,6)=C(2,6)-1
200 GO TO 100
2000 REM Гибель героя.
2010 IF C(3,1)>4 AND C(3,5)>5 AND C(3,6)>3 THEN PRINT "Крам убивает Инфа": LET N$ = N$+"#"
2020 IF C(4,1)>4 AND C(4,5)>5 AND C(4,6)>3 THEH PRINT "Брог убивает Ору ": LET N$=N$+"#"
2030 IF N$="Инф#" THEN LET C(2,1)=C(2,1)-2: LET C(2,2)=C(2,2)+1: LET C(2,5)=C(2,5)-1
2040 IF N$="Ора#" THEN LET C(1,1)=C(1,1)-2: LET C(1,2)=C(1,2)+1: LET C(1,5)=C(1,5)-1
2050 RETURN
3000 REM Гибель монстра
3010 IF C(3,6)<-9 THEN PRINT "Крам убит": LET G$="STOP"
3020 IF C(4,6)<-9 THEN PRINT "Брог убит": LET G$="STOP"
3030 RETURN
```

Строки 2030 и 2040 используют новое значение N$, чтобы изменить атрибуты оставшегося в живых персонажа. В приведенном примере его гнев увеличивается, а самолюбие убывает, такое взаимовлияние довольно просто, но наглядно демонстрирует возможности системы, в которой один персонаж оказывает влияние на других.

Последняя подпрограмма, начинавшаяся в строке 3000, проверяет, не пора ли погибать монстрам. То же уменьшение сил, которое убивает положительного героя, может убить и отрицательного.

Переменная G$, введенная в строке 125, участвует в этой процедуре. G$ принимает значение "STOP", если Брог умирает и после возвращения в главную часть программы заканчивается ее работа в строке 140. Когда монстры уйдут, Вы можете провести своих героев на яхту.

Приведенная выше распечатка текста программы может быть еще и еще более усовершенствована.

Так, в строках 110 и 120 персонаж выбывает из борьбы только если его силы падают ниже -9. Вы можете принять здесь в рассмотрение и фактор мужества, используя для этого логический оператор OR.

Еще одним дополнением может быть, например, кинжал, лежащий у двери капитанской рубки. Здесь возможны варианты. Может быть, один их героев подберет кинжал, пока другой ведет схватку, а может быть Вы решите, что его подбирать надо только когда оба противника подавлены.

Добавив несколько новых строк и строк PRINT. Вы придадите программе элемент неожиданности, совершенно очевидный дух искусственного интеллекта, а Инф и Ора получает дополнительный шанс остаться в живых.

Такие объекты, как кинжал, легко хранить в памяти программы и несложно организуется их перемещение их вместе с персонажами. В распечатке на рис. 3 показан один из методов хранения объектов в локациях до того, как персонажи их подобрали. Каждая локация имеет группу предметов и хранит их в переменной, скажем B$. Как только изменяется локация, изменяется и содержимое переменной. Если в данной локации предметов нет, то возвращается пустая строка. Размещение многих объектов в одной строковой переменной вместо массива позволяет значительно снизить расход памяти компьютера.

Формат, в котором названия объектов хранятся в b$, довольно очевиден и может обслуживаться парой строк.

Переменная начинается с астерикса (*), им же отделяются объекты друг от друга.

Первая буква в названии объекта указывает на его тип.

О – оружие

П - обычный предмет

В - враждебное существо.

Если бы в Вашей программе участвовали товарно-денежные отношения, можно было бы ввести и еще один класс ценностей, отнеся к ним драгоценные камни, золото, монеты, украшения и пр. Очевидный класс составляют магические предметы.

Следующие символы содержат информацию о названии объекта.

```
10 LET b$="*пяхта*окинжал*вБрог*вКран"
20 INPUT a$
25 LET c$=a$(1 TO 3)
30 LET k=1
50 IF b$(k)="*" THEN GO TO 90
60 IF k=LEN (b$) THEN PRINT "Объект не найден": GO TO 20
70 LET k=k+1
80 GO TO 50
90 LET m=k+2: LET c=m+2
100 IF c$=b$ (m TO c) THEN GO TO 120
110 GO TO 70
120 IF b$(k+1)="о" THEN PRINT a$; ":"; "Вы нашли оружие"
130 IF b$(k+1)="п" THEN PRINT a$; ":"; "Вы нашли предмет"
140 IF b$(k+1)="в" THEN PRINT a$: ":": "Перед Вами появился противник"
150 GO TO 20
```

Переменная b$ задается в строке 10.

В строке 20 вводится название предмета и в строке 25 оно "урезается" до трех символов. Цикл в строках 50...80 ищет появление астерикса. Когда он найден, выполняется переход на строку 90. где символ за символом сканируется символьная переменная.

В строках 120. .. 140 определяется тип объекта и информация об этом доводится до играющего.

Если же этот объект не найден в b$, то сообщение об этом генерируется в строке 60.

А теперь продолжим наш проект. Когда Инф и Ора обнаружили кинжал, подняли его, победили чудовищ и прошли в дверь, их ослепил яркий свет. Они оказались в подземном городе короля Кельроса. Их захватили в плен и повели на допрос к королю.

Самое первое средство, которое стоит попробовать для выхода из такой ситуации - попытаться договориться. В следующей части мы рассмотрим технику общения между персонажами.

(Продолжение следует).

FORUM

В нашей почте сентября лежит сразу несколько писем, которые могли бы претендовать на приз "Лучший вопрос месяца", если бы мы осмелились устроить такой конкурс. Но наши возможности имеют вполне осязаемый предел и не позволяют так рисковать.

Привидения в ОЗУ

Во-первых, поступило обращение от военнослужащего из г. Мурманска т. Новикова В. П. сомневающегося в надежности своей машины. Вот в двух словах суть его проблемы.

Экспериментируя с оператором PEEK, он обнаружил, что значительная часть верхнего пространства памяти компьютера содержит нули, что и не удивительно - память девственно чиста, но на самом верху, в районе адреса 65300 содержатся какие-то числа.

Попытка их "очистить", т.е. принудительно обнулить, в одних случаях ни на что не влияет, в других выводит компьютер из строя (конечно не физически, а только программно). Более того, многие из них просто нестабильны. Если в ОЗУ уже размещена какая-то Бейсик-программа, то эти числа меняются без видимой закономерности, что затрудняет ему борьбу с ними за очищение памяти от "мусора".

Прежде всего, мы должны успокоить нашего читателя - с компьютером все в порядке, просто он провел эксперименты с одним из разделов оперативной памяти компьютера. Те, кто имеет нашу разработку "Большие возможности Вашего Спектрума", могут найти в ней карту памяти и увидеть, что в исходном (после включения) состоянии этот участок памяти отводится для хранения символов графики пользователя (UDG), а также для организации в нем вспомогательных служебных структур - стеков. Это стек GO SUB и машинный стек. Содержимое этой области находится под управлением ПЗУ компьютера и потому может изменяться, даже если пользователь об этом и не догадывается.

Напомним для начинающих карту памяти "Спектрума". Нижние 16K занимает ПЗУ (содержащее необходимые процедуры операционной системы, о чей мы достаточно подробно пишем в разделе "Секреты ПЗУ"). Далее примерно 7K занимает экранная область памяти, потом идет буфер принтера, системные переменные и пр. И наконец - свободное пространство (более 2/3 общей памяти). Это "свободное пространство" подвержено постоянным изменениям. Это не чистый лист бумаги, на котором можно что-то написать и стереть. Оно непрерывно "пульсирует", то увеличиваясь, то уменьшаясь, выделяя свои просторы для временного использования по мере необходимости для различных нужд. Контроль же за всеми этими процессами ведется в установках значений системных переменных и в данных на стеках, находящихся на вершине памяти (вот почему они могут меняться и почему их нельзя "портить").

Возьмем простой пример, Вы нажали клавишу EDIT и видите как строка из листинга Вашей программы на экране копируется в нижнюю часть экрана, где Вы можете вносить в нее свои изменения и дополнения. Вам и невдомек, что в это же время в "свободной" области памяти происходят бурные события. Выделяется место под область редактирования, передвигаются вверх буфер INPUT и стек калькулятора, копируется содержимое Вашей строки из программной области в область редактирования и прочее и прочее.

Вы нажали ENTER и отредактированная строка заняла свое место в программе. И это опять "раздвигание" программной области, чтобы втиснуть измененную строку, смещение всего, что было выше ее вверх, копирование, сворачивание области редактирования и, наоборот, сдвиг вниз буфера INPUT и стека калькулятора.

Кстати, обратите внимание на то, что после такого сдвига вниз освободившиеся ячейки станут "свободными", но не нулевыми, как после включения компьютера. Компьютер не тратит драгоценное время на очистку (если обнуление это очистка) освободившихся областей памяти. Ему достаточно указать в нужных адресах, что то, что там находится - "чисто и свободно", не важно, равно оно нулю или нет.

То же происходит и со стеками, находящимися в верхних адресах компьютера. Они могут расти (растут они вниз) или сжиматься, но по мере их сжатия освобождающиеся ячейки памяти присовокупляются к свободному пространству, но не обнуляются. Там остается "эхо" предшествующих событий и держится оно до тех пор, пока стек, разрастаясь сверху вниз вновь не проутюжит эти адреса или пока снизу не подступят рабочие области по мере развития Вашей Бейсик-программы.

Что такое стек?

Продолжая эту тему, ответим также на вопрос В. Сивцева из Воронежа о том, что такое стеки GO SUB и машинный стек, о которых так часто упоминается.

Большинство наших читателей конечно знает, что делает команда GO SUB - вызывает исполнение подпрограммы. Когда ПЗУ встречает в Вашей программе команду GO SUB, то сначала запоминается то место, в котором она встретилась - номер строки и номер оператора в строке.

Когда же встретится команда RETURN, то эти запомненные данные и послужат для точного возврата именно туда, откуда подпрограмма вызывалась.

Стек GO SUB - и есть тот участок памяти, в которой это все и запоминается. Он растет сверху вниз и работает по принципу "последним пришел - первым уйдешь". Т.е. при команде RETURN со стека снимаются данные, поступившие в последней GO SUB. Легко представить себе, что вызвав через GO SUB подпрограмму, Вы в ней опять применяете GO SUB для вызова другой и так далее. При этом стек будет расти все больше и больше. Совершенно очевидно, что возвращаясь из этих подпрограмм через RETURN, Вы снимете все заложенные на стек данные, он сожмется и придет в исходное состояние. Кстати, если Вы из подпрограмм будете возвращаться через GO TO, то такой нежный баланс на стеке будет нарушен и работоспособность Вашей программы будет висеть на волоске.

Машинный стек находится чуть ниже стека GO SUB и непосредственно граничит с верхней границей свободного пространства. Соответственно, когда стек GO SUB расширяется и сжимается, то машинный стек копируется то вниз, то вверх.

Машинный стек аналогичен стеку GO SUB по природе, но с ним работает не Бейсик-программа, а сам процессор Z-80. Там процессор запоминает адреса процедур, когда они вызывают другие процедуры и наоборот, снимает эти адреса при возврате. Вторая его функция - временное место хранения содержимого регистров процессора или каких-либо ячеек памяти, когда их нужно освободить для каких то срочных дел, но нельзя их потерять. Для этого в системе команд Z-80 есть даже две группы команд - PUSH (команды этой группы отправляют на стек данные на хранение), и команды POP (они восстанавливает данные, снимая их со стека).

Так, например, PUSH BC отправит на стек содержимое регистров B и C, а команда POP HL снимет то, что последних помещалось на стек (2 байта) и зашлет их в регистры H и L.

Обратите внимание еще и на тот факт, что если Вы никогда в машинном коде (на Ассемблере) не программировали и не собираетесь, то и машинный стек Вам вроде бы не нужен? - Ничего подобного!

Вполне достаточно того, что любое Ваше действие на компьютере (а также и бездействие) сопровождается исполнением сотен процедур, находящихся в ПЗУ, которые работают в машинном коде и без этого стека жить никак не смогут.

Зачем нужны прерывания?

Еще один вопрос, интересующий десятки наших читателей в последнее время - это использование прерываний в "Спектруме". Что такое прерывания? Зачем они нужны?

Надо сказать, что кое-что об этом есть в нашем трехтомнике по программированию в машинных кодах, но видимо недостаточно. В нашем портфеле сейчас заготовлен обширный материал на эту тему, выполненный Баяновым К. К. из Томска по нашему заказу, но на финише года мы должны в первую очередь завершить начатые статьи и циклы, поэтому этот материал дадим в следующем году, а пока ограничимся несколькими словами о прерываниях.

Система прерываний - это средство, с помощью которого процессор может прекратить исполнение одной задачи и перейти к другой, а потоп опять вернуться туда, где прервал работу.

Все компьютеры имеют систему прерываний. На самых совершенных машинах работа может быть организована например так, что процессор исполняет свою главную задачу, но иногда обращается к вспомогательным, которые в свою очередь распределены по приоритетам (по степени важности) и более приоритетные выполняются в первую очередь. По мере исполнения каждой работы компьютер возвращается к предыдущей незаконченной задаче.

Процессор Z-80 может исполнять прерывания двух основных типов.

Первый тип - немаскированное прерывание (NMI). Оно так названо потому, что программист не может его предотвратить или направить на исполнение своей задачи (маскировать). Процессор неотвратимо исполняет это прерывание, когда оно происходит.

Маскируемое прерывание более интересно, т.к. может быть использовано и потому, что оно имеет три разных режима (IM0, IM1 и IM2).

В нулевом режиме процессор просто ждет команды от внешнего устройства на то, чтобы прервать свою работу и обратиться к исполнению программы, заложенной в известном месте памяти. В фирменной машине и в большинстве самодельных аналогов нет средств для работы в этом режиме.

Первый режим (IM1) - основной. "Спектрум" конструктивно исполнен так, чтобы всегда работать в нем. В этом режиме процессор прерывает работу, запоминая в регистре PC адрес, в котором он прервался, и обращается в ПЗУ по адресу 0038H. Процедура ПЗУ, расположенная в этом месте обеспечивает "тикание" внутренних часов "Спектрума" и сканирование клавиатуры в поисках нажатой клавиши. Если клавиша была нажата, тут же изменяются значения флагов в системных переменных (см. раздел "Секреты ПЗУ") и далее работа продолжается.

Такое "тикание" происходит 50 раз в секунду (это связано с тем, что в нашей электрической сети частота тока - 50 Гц). Если бы Вы жили в Англии или в США, Ваш компьютер сканировал бы клавиатуру чаще - 60 раз в секунду.

Самый серьезный, хотя и реже используемый режим - второй (IM2). В нем после прерывания процессор обращается не в одну и ту же точку ПЗУ, как в IM1, а туда, куда задаст сам программист. Адрес перехода по этому прерыванию вычисляется следующим образом.

Он состоит из двух байтов. Старший байт процессор берет из регистра I (который так и называется - вектор прерываний). Программист может задать его так, как ему надо. Второй байт адреса (младший) процессор принимает от внешнего периферийного устройства, выдавшего этот сигнал прерывания.

Адрес = 256*(содержимое I)+то, что поступило по шине данных. Далее компьютер смотрит, что содержится по этому адресу (и в следующем) и определяет, куда ему перейти на исполнение новой задачи.

Новый адрес = содержимому адреса+256*содержимое (адрес+1).

Пример:

Регистр I содержит число 143. От периферии поступило число 27, расчет адреса дает:

256*143+27 = 36635 (и 36636)

Если в этих ячейках содержатся числа 137 и 93, то будет сделан переход на адрес:

137+256*93=23945

Таким образом, программист в нужном ему месте может организовать таблицу переходов, содержащую 126 адресов. Длина этой таблицы будет 256 байтов. Далее он выставит регистр I так, чтобы он всегда указывал на начало этой таблицы, и положение адреса перехода по прерыванию будет зависеть от внешнего устройства. Так, можно обеспечить работу компьютера со 128-ью разными внешними устройствами, каждое из которых обслуживается своей личной программой (драйвером).

Пока мы на этом остановимся, а в будущем году дадим развернутый материал на эту тему, причем приложим к нему несколько образцов программ-драйверов.

К вопросу о совместимости.

Нам было очень приятно получить письмо из Елгавы (Латвийская республика) от нашего читателя Масолова Ю. В.

Вы помните, что в начале года на наших страницах проводилось исследование почему не работает на многих самодельных машинах программа "Арканоид". Было установлено, что процессор в фирменной машине при чтении внешнего порта "подхватывает" байт атрибутов и в этом вся проблема и заключалась.

Нашему читателю удалось таким же образом запустить TOP GUN, DUET и многие другие программы.

Конечно же, нам очень приятно, что проблемы, поднятые на наших страницах, приводят в итоге к практическим решениям.

Вместе с тем, как указано в письме, есть еще проблемы с прерываниями INT. В разных схемах этот сигнал формируется по разному. В результате опытов удалось установить, что этот сигнал должен появляться в НАЧАЛЕ КАДРОВОГО СИНХРОИМПУЛЬСА и сбрасывается через M1 и MREQ, а если прерывания запрещены, то через 10 мксек. Если же сигнал начинается в другое время, имеются следующие проблемы:

1. Дрожат спрайты на экране (ARKANOID и др.).
2. Исчезает курсор в экране (ARTSTUDIO).
3. Линии в бордюре не совпадают с фоном экрана.

О необходимости разобраться с сигналом INT говорит и наш читатель Хвиш А. А. из г. Луцка. Им также установлено, что этот сигнал оказывает большое влияние на программную совместимость с родительской машиной. Так, на его "Балтике", как и на машинах его близких друзей, отказывались работать программы "Samantha Fox", "Turbo Comp" и др.

Идею по доработке компьютера он нашел в минском журнале "Радиолюбитель" и, как пишет наш читатель, "Теперь компьютер работает отлично".

В этом журнале есть и немало прочей ценной информации для любителей нашей системы. Так, например, сообщается о том, что минский кооператив "Сонет" поставляет желающим ПЗУ 27256 с записью в старшей 16-килобайтной области системы "Disk-monitor", копировщика ZX-COPY, программы "Assembler" и базовой системы ввода/вывода (BIOS) CP/M.

Данный пакет дает возможность не только останавливать любые программы, вносить в них изменения и запускать с места останова, но и писать свои. Выгрузка программ на ленту или диск выполняется несложно с помощью контроллера. Загрузчик BIOS делает возможной работу в среде CP/M, причем без каких-либо существенных аппаратных изменений.

Принцип работы заключается в переписывании верхних 16K в область теневого ОЗУ.

Наш читатель опробовал этот пакет на своем "Балтике" и остался очень и очень доволен. К сожалению, у нас нет информации от кооператива "Сонет" по этому вопросу, но если кто-то из приближенных к нему читателей может содействовать установлению такой связи, мы с радостью опубликуем более широкую информацию об их разработках, ведь судя по нашей почте, очень многих интересует работа в среде CP/M.

Но есть и новые вопросы, новые проблемы.

Во-первых, у нашего постоянного читателя из г. Киева Довженко В. П. не идет программа PSI-CHESS на Львовском варианте компьютера. Что это - случайность или за этим что-то стоит?

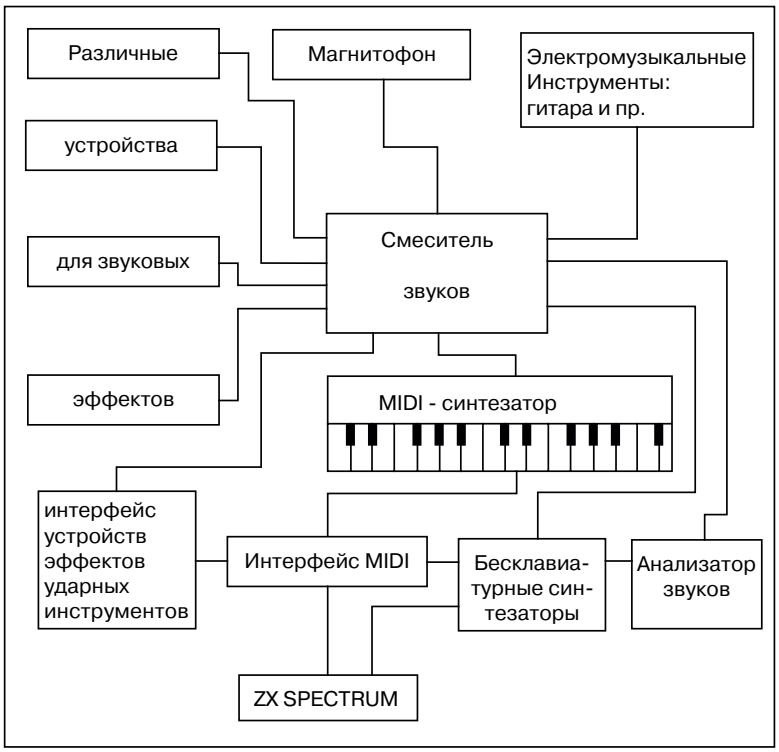
Есть также сообщения и о том, что скорость игр, запущенных на "Ленинградской" версии заметно ниже, чем на других версиях, например на "Балтике". И это скорее всего не случайность.

Интерфейс MIDI

Еще один часто встречающийся вопрос касается интерфейса MIDI, который служит для подключения к компьютеру музыкальных инструментов, синтезаторов и прочих музыкальных устройств.

Читателей интересует все: описания, схемы, прошивка ПЗУ, программная поддержка.

К сожалению, мы не располагаем по этому вопросу никакой информацией. Может быть в стране просто еще нет разработок, привязанных к отечественной элементной базе, но если все же что-то есть, то мы с радостью приобрели бы любую информацию, столь нужную нашим читателям.



Для тех, кто пока просто не задумывался над этим вопросом, мы дадим блок-схему подключения музыкальной системы к "Спектруму", в основу которой положен интерфейс "MIDI" и синтезатор звуков, например YAMAHA DX-100. Надеемся, это зрелище впечатлит тех, кто может заняться такой разработкой. Из схемы наглядно видно, к каким системам открывается доступ, если есть в наличии MIDI-интерфейс. Схема взята из журнала Sinclair User.

Заканчивая сегодня раздел "Форум", мы хотим порекомендовать Вам достать и прочитать журнал "Компьютер" (N1 за 1991 год). Этот совместный советско-польский журнал формируется по материалам польского ежемесячного журнала "Kompjuter" и выходит раз в квартал. Журнал известен среди пользователей компьютеров разных систем, в основном он посвящен IBM-совместимым машинам, но данный номер был почти полностью предоставлен системе "СИНКЛЕР" и содержит весьма интересные и поучительные материалы.

Мы не знаем, как дело обстоит во всей стране в целом, но в Москве и Ленинграде он еще есть в продаже и приобретается без особых проблем.

Мы очень часто получаем письма с просьбой посоветовать над чем бы поработать нашим читателям, умеющим программировать и желающим приложить свои усилия с пользой для себя и общества.

Рецепт очень прост - в нем всего пять слов: "Найдите потребность и удовлетворите ее". (Рецепт взят из недавно вышедшей книги "Как стать предприимчивым и богатым. Из американских рецептов." М., "Молодая гвардия", 1991)

Нужна помощь.

Требуются обучающие программы для ПК "Спектрум". 183060, Мурманск-60, ул. Мира, д. 5/4, кв. 221, Гершковичу М. Б.

Для применения в медицинской практике требуется тех. документация и

запрограммированные ПЗУ с ДИСК МОНИТОРОМ ОС. 270111, Одесса-111, а/я 63, Карасевичу В. В.

Компьютер в моем городе.

Сегодня мы обращаемся ко всем нашим читателям с предложением принять участие в крупной акции по созданию летописи развития движения самодеятельного компьютерного творчества в нашей стране.

То явление, которое мы сейчас переживаем, связанное с огромным проникновением компьютерных знаний в самые широкие круги, уникально. Это подлинно народная компьютерная революция. И все Вы ее творцы и активные участники.

Народ сам, не спросясь никого, и не дожидаясь милости от государственных комитетов по информатике и по народному образованию оснастил себя колоссальным парком машин.

Сегодня в стране по нашим оценкам 1.5 - 2 миллиона пользователей "Спектрума". Можно сколько угодно восхищаться гением сэра Клайва Синклера, создателя этого компьютера, но мы то абсолютно точно знаем сколько у нас своих гениев. Тех, кто "на коленке", разрабатывал и из немыслимых деталей собирал советские "Спектрумы", когда официальные журналы упивались достоинствами "Микроши".

А через что прошли заводы, которые начали выпускать первые "Спектрумы"! Каким мужеством обладали их руководители, не побоявшиеся срыва планов основного производства и взявшиеся за совершенно новое, рискованное и не пользующееся официальной поддержкой дело!

Нет в мире другой страны, в которой были бы еще самодельные "Спектрумы". А ведь сколько разных версии разработано и освоено Вами!

Можно только поражаться, какой же волей должен обладать народ, который без малейшей поддержки и даже вопреки ограничениям на ввоз процессоров Z-80 (и такое было!) поднял в стране уровень компьютерной грамотности и довел ее до миллионов.

Мы обращаемся с просьбой ко всем. Пока все это живо в памяти, именно в эти трудные для страны дни давайте запишем все, что мы знаем и помним об этом пути.

Пройдут годы и мы уверены: об этом периоде будут говорить с должным уважением. Давайте соберем наши общие знания в единую летопись.

Пишите все, что знаете. Когда и как в Вашем городе появились первые компьютеры? Какие модели? Чем сейчас живет город? Кто выпускает машины в Вашем регионе? Какие программы пользуются наибольшим авторитетом и какие модели машин? Кто и как их разрабатывал, кто что предложил нового? Где Вы собираетесь и как? Сколько Вас? Сколько в городе машин и каких? Есть ли учебные классы и школы, оснащенные компьютерами?

Нам нужны и адреса производителей "Спектрумов". Самых крупных мы, конечно знаем, но ведь их сейчас сотни.

Если Вы живете не в городе, а в поселке или селе - это не имеет никакого значения - нам важно знать все.

Если Вы лишь недавно приобрели компьютер и не имеете информации о прошлом - напишите, как Вы видите день сегодняшний. Ведь завтра и он станет историей.

Нам обычно пишут, когда с компьютером что-то плохо. Почему бы не написать, когда с ним все хорошо (и сказать, что это за модель и какой завод ее сделал).

Особо ценными были бы для нас заметки и воспоминания тех, кто были первыми и разрабатывали первые версии.

Одним словом, нас интересует ВСЕ! Мы будем скрупулезно накапливать и хранить всю информацию. Как знать, ведь если этого не сделать сейчас, она может быть безвозвратно утрачена! Много ли сейчас узнаешь о тех людях, которые делали первый российский (украинский, белорусский…) автомобиль?

У нас есть мечта - когда-нибудь создать музей бытовых персональных компьютеров. Чтобы лет через двадцать все могли прийти и увидеть, как это начиналось.

Пока это только мечта, но если Вы, уважаемый читатель, внимательно следите за нашим развитием, то знаете, что мы хоть и медленно, но неуклонно идем своим путем. И мы

обязательно создадим такой музей, когда наберем для этого достаточно сил.

Обращаясь к Вам с такой просьбой мы преследуем еще одну цель.

Мы хотим выявить среди Вас тех, кто умеет вести маркетинговые исследования, обладает для этого достаточной коммуникабельностью, контактностью, способностью собирать и анализировать информацию.

Крупные разработчики и изготовители компьютеров, периферийных устройств, поставщики программного и информационного обеспечения нуждаются в надежных маркетинговых исследованиях, прежде чем предпринимать какие-либо шаги. До сих пор мы неплохо удовлетворяли их потребности, но количество заказов растет и мы будем формировать коллектив внештатных исследователей, которые по нашему заказу смогут проводить такие оплачиваемые работы.

Наш каталог.

Вышел из печати и сейчас рассылается подписчикам наш каталог игровых и прикладных программ для "Спектрума". Это получился солидный труд объемом 150 страниц, включающий более 5000 программ.

Мы высылаем его тем, кто в свое время (три месяца назад) подписался на предложенных нами условиях. Мы полагаем, что он обрадует всех и они не пожалеют о том, что перечислили средства "ИНФОРКОМу".

Для тех, кто в свое время не оформил подписку на него, сообщаем, что установлена цена в размере 45 рублей. Заказы принимаются.

Мы надеемся на то, что этот каталог станет общепринятым средством для общения любителей между собой при обменах, переписке, составлении собственных каталогов и т. п.

Мы также планируем один раз в год дополнять его новым содержанием и держать его актуальность на должном уровне.

ИНФОРКОМ ПРЕДЛАГАЕТ...

Наши читатели привыкли ежемесячно вместе с очередным выпуском "ZX-РЕВЮ" получать информационный листок, начинающийся с этих слов.

В этом месяце Вы не получите такого листа. Причина в том, что мы ведем пересмотр цен в связи с общей экономической ситуацией и должны какое-то время подождать, пока пройдут заказы по отправленным ранее листкам с устаревшими ценами.

Просим откликнуться.

Мы убедительно просим откликнуться нашего читателя, приславшего для нас на дискете 5,25" рисунки CLOTH, ROBOT и SUNRISE. При передаче между подразделениями утерян конверт с Вашим обратным адресом, а нами для Вас кое-что подготовлено.

Что бы это значило?

Любителям поломать голову предлагаем набрать нехитрую программу и попытаться объяснить как же она работает.

```
10 FOR k = 72 TO 79
20 POKE 23681,k
30 LPRINT "GOOD LUCK TO YOU"
40 NEXT k
```

Тем, кому этот пример доставил удовольствие, рекомендуем в 1992 году читать в "ZX-РЕВЮ" сериал В. С. Михайленко (Минск), посвященный методам защиты программ от копирования и борьбы с ними. Пример взят оттуда.

Содержание