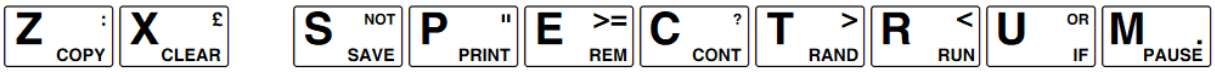

КРАСНЫЙ-СИНИЙ-ЗЕЛЁНЫЙ
ПРЕДСТАВЛЯЕТ:
Z : COPY
X £ CLEAR
- S NOT SAVE
P " PRINT
E >= REM
C ? CONT
T > RAND
R < RUN
U OR IF
M . PAUSE
BASIC
С Нооооооля
2025 GOD MODE Edition

КРАСНЫЙ-СИНИЙ-ЗЕЛЁНЫЙ

ПРЕДСТАВЛЯЕТ:

Цикл книг «Программирование глазами купчинского гопника»



BASIC

➡ С НУЛЯ ⬅

Внимание: книга может содержать терминологию, схожую со словами ненормативной лексики и предназначена для лиц, старше 18 лет.

2025 GOD MODE Edition

КРАСНЫЙ-СИНИЙ-ЗЕЛЁНЫЙ

И корпорация «СЧАСТЛИВЫЕ ДЕВЯНОСТЫЕ»

ПРЕДЛАГАЮТ:

Широкий выбор разнообразной стебно-обучающей, исторической и научно-популярной укуренной литературы как для начинающих, так и для профессиональных приколистов со стажем. Вот краткий перечень, того, что мы предлагаем сейчас.

♣ Сказки и мемуары о жизни 1980-2001 годов:
– жизнь квартиры и двора у домов 86 корпус 1 и 2 по Бухарестской улице (1983-2000)
– появление в квартире компьютеров ДУБНА-48К и ZX-Spectrum (1992-1996)
– летние приключения в посёлках Солнечное-2 и Солнечное-3 (1993-1997)
– райский отдых в посёлке Тарасово и семейной базе отдыха ЛОМО (1989-1991)
– обломная покупка участка СНТ «Электрон» (1998)
– освоение уличной и промышленной светотехники (1989-2000)
– прогулки по Купчинским дворам и железным дорогам (1984-2000)
– и многое другое…

♥ Исторические исследования:
– История развития домов серии 1-ЛГ602-В (1966-2010)
– История уличного освещения Ленинграда (1934-2015)
– История появления помоек в садоводческом массиве «Крутая Гора»
– «Компьютерные» и «Речные» поля у совхоза Раздолье
– Болото Сумрицы. Прошлое и настоящее (2000-2020)
– Крутогорское море
– «Электрон» и Электричество (1995-2020)

♠ Обучающая компьютерная литература и статьи:
– Яркий мир «Волшебного DOS» **НОВИНКА 2024 ГОДА!**
– Четыре сезона с Mapster32 или eDUKE ДолBuilding По-Купчински (Красная часть)
– Четыре сезона с Mapster32 или eDUKE ДолBuilding По-Купчински (Синяя часть)
– Четыре сезона с Mapster32 или eDUKE ДолBuilding По-Купчински (Зелёная часть)
– Четыре сезона с Mapster32 или eDUKE ДолBuilding По-Купчински (Жёлтая часть)
– Dune-2 Building Crazy Scenarios
– IMPACT тайна 80-го уровня
– IMPACT Как создать свои уровни вместо базовых
– и прочие…

♦ Прочие сувениры и сокровища из старых добрых времён:
– плёночные фотографии домов, улиц и дворов (1997-2004)
– видеофильмы с VHS кассет (1998-2007)
– забойная музыка (Acid, EuroDance, Dream, Trance, House, Hardcore, Rave (1992-2003))

Планируется к выходу эта книга, а также другая крышесносящая литература. Мы поможем вам войти в охрененно богатый духовный мир автора этой книги.

По вопросам оптового приобретения продукции
обращаться в головной офис корпорации по адресу:

123456, мир Абстрактных Иллюзий, город Мечты, улица Счастья дом 90, тел: 987-65-43

ZX- Spectrum BASIC с н□□□□□ля

Глава 1

Введение

По законам эпистолярного жанра, любая серьёзная книга о Спектруме должна начинаться с длинной телеги об истории появления компьютера, его создателе и запоздалой компьютеризации СССР. Этих историй тут не будет, ибо давно неактуальны. Я начну свою книжку иначе.

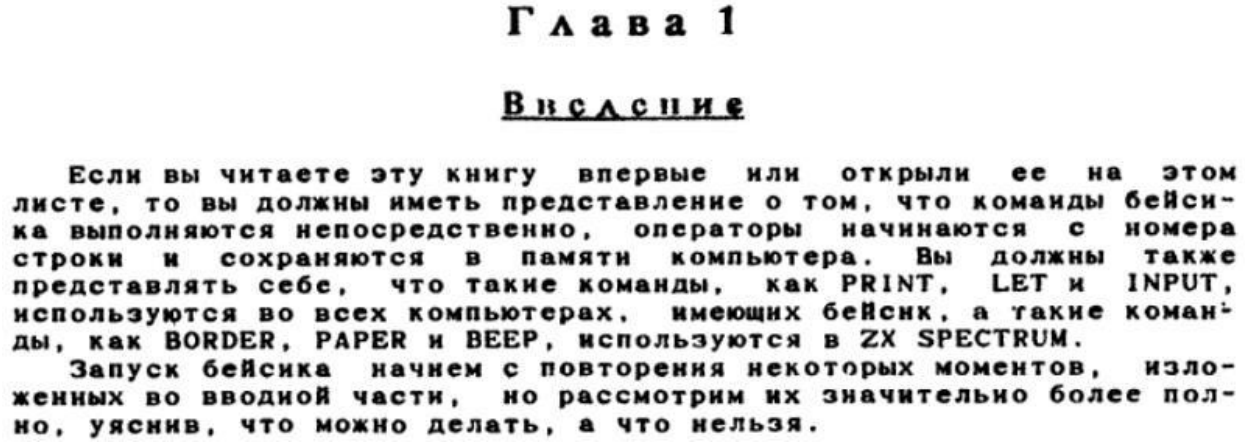
Весна 2024 года подходила к концу. Я доводил до ума мемуары о своей жизни в Ленинграде-Петербурге на рубеже восьмидесятых и девяностых годов. В какой-то момент добрался до историй о первых попытках программирования, которые предпринимались в январе-марте 1993 года.

Я пытался вспомнить, какой же затык у меня вызвала игра «*Animated Strip Poker*», и почему по подобию игры «Dominoes» не удалось записать свою BASIC программу блоком «*Bytes:*». Я планировал поверхностно чиркануть пару строчек об этой проблеме, и идти дальше. Но рассказ не клеился, потому что к текущему моменту я напрочь забыл не только ассемблер, но и BASIC. Нужно было найти время покопаться в старом архиве, отыскать эту игру, но пока было не до нее.

И вот как-то раз днем я залез в давно забытый и заархивированный цифровой скарб по Спектруму, который накопил за всё существования домашних компьютеров с 1992 года и привёз с собой за рубеж. Отыскивая игру «*Strip Poker*», я параллельно решил сделать ревизию папки, чтобы хлам не занимал лишнего места. Сортируя архив материалов по Спектруму, я решил пересмотреть подборку литературы и выкинуть потерявшее актуальность.

«На кой хрен они мне нужны?» – подумал я, посмотрев на старые самоучители по BASIC.

Просто так стереть жалко, нужно хоть глянуть разок. Открываю, я значит, первую страницу и читаю свежим взглядом:



Глава 1

Введение

Если вы читаете эту книгу впервые или открыли ее на этом листе, то вы должны иметь представление о том, что команды бейсика выполняются непосредственно, операторы начинаются с номера строки и сохраняются в памяти компьютера. Вы должны также представлять себе, что такие команды, как PRINT, LET и INPUT, используются во всех компьютерах, имеющих бейсик, а такие команды, как BORDER, PAPER и BEEP, используются в ZX SPECTRUM.

Запуск бейсика начнем с повторения некоторых моментов, изложенных во вводной части, но рассмотрим их значительно более полно, уяснив, что можно делать, а что нельзя.

Рис. 2. Типичное введение из самоучителя по BASIC 1987-1993 годов.

С такого оптимистичного описания начинался любой самоучитель по BASIC первой половины 1990-х годов.

И действительно, если вы читаете эту книгу впервые, то конечно должны не только иметь представление о том, как работает BASIC, но и наизусть знать всю прошивку ПЗУ с дизассемблером в художественном переводе. Это же логично! Представьте себе подростка, которого до 18-ти летия никак не готовят к взрослой жизни. Естественно, что в день совершеннолетия он моментально прозревает, становится великим учёным с мировым именем, женится, одномоментно рожает восемь совершеннолетних детей и живёт долго и счастливо.

Конечно же, я знал, что команды LET, PRINT и INPUT используются во всех компьютерах, а BORDER, PAPER и BEEP только в ZX-Spectrum. Базара нет! Эту истину знает каждый с рождения. Конец восьмидесятых, начало девяностых. Неимоверными усилиями человек накопил себе денег на дефицитный компьютер, купил его и принёс домой. Вот он лежит в коробочке, еще даже не распакованный. Чувак даже не знает, как его толком подключать. Смотрит он на эту коробку, и вдруг к нему приходит озарение, или озаряет приход. Встаёт он посреди комнаты и изрекает, как Ленин с броневика:

«Ёптыть, а ведь и правда, команды PRINT и INPUT используются во всех каркуляторах и кампутерах!».

«*Запуск Бейсика начнём с повторения некоторых моментов, изложенных ВО ВВОДНОЙ ЧАСТИ, но рассмотрим их значительно более полно…*»

Хорошо завернули, ни прибавить, не отнять. Только есть пара нюансов: где эта, с.ка, вводная часть!? С какого адреса собрались производить запуск BASIC? И наконец, насколько подробно там что-то разбирали, если следующей главой идёт обучение вводу выражения «LET A=10» и вертикальному передвижению курсора по строкам? Вводная часть о том, как принести компьютер домой, протереть для него стол и извлечь из упаковочной тары?

За прочтением предисловий поднялось настроение. От этого описания меня попёрло, как травы на газоне или сока из берёзовых шишек. Из памяти выныривали какие-то яркие детско-юношеские впечатления, и вскоре накрыла ностальгия. Книжки моментально были помилованы. Мало того, впереди собственной тени, я побежал на сайт Virtual TR-DOS докачивать всю остальную литературу, относящуюся к языку BASIC, да и вообще любую спектрумскую прессу, попавшую под курсор мышки.

Бегло пролистав все найденные самоучители, я сразу вспомнил легендарные фразы «*Гоблины хоронят рыцаря*», «*В машинной программе вы не должны использовать регистры IY и I*» и остальные шедевры, смысл которых я не понимал, но с годами они стали мемами.

Копнув глубже, я обратил внимание, что изнутри все эти бессмертные творения выглядели как братья близнецы. Авторы разных книг меняли местами слова в предложениях, но структура повествования, последовательность глав, опечатки в программах, и типичные ошибки кочевали из издания в издание. В порядке вещей давались таблицы с кучей технической информации вообще по другому компьютеру и это никого это волновало. Складывалось впечатление, что авторы даже не проверяли работоспособность своих трудов.

Да, тридцать с лишним лет назад, на волне популярности, было наштамповано много книг, но почитав эти экземпляры сейчас, я задумался: а откуда авторы этих шедевров брали вдохновение? Логично предположить, что был какой-то один первоисточник, с которого бездумно нашлёпали копий.

В азарте я стал искать международные ресурсы по ZX-Spectrum и выкачивать оттуда всё, что выкачивалось, начиная от журналов «*CRASH*», «*Your Spectrum*», «*ZX-Computing*» и заканчивая машинными кодами, дизассемблером ПЗУ с документацией Z80 на английском, итальянском и чешском языках.

И таки, нашёлся виновник торжества! Большинство русскоязычных книжек оказались перепечаткой корявого перевода английского самоучителя Стивена Виккерса 1983 года с дополнениями последних глав от 1987-го. По всей видимости, перевели только основную часть книги, а эта самая мифическая «вводная часть» по пути пролюбилась. Как вариант, под вводной главой подразумевалось «Приложение С», что тоже логично.

Освежив память пролистыванием данной литературы, я наконец-то дошёл до игры «*Strip Poker*» и сразу понял, в чем произошёл затык 31 год назад. Да даже сейчас, позабыв

ассемблер и утратив навыки работы с отладчиками, я призадумался. Как аккуратно вскрыть эту хрень, чтобы увидеть BASIC? А впрочем, фиг с ней, я же не вскрывать её полез, а поверхностно вспомнить события для летописи.

Следом, неимоверными усилиями, я отыскал ту самую версию «Dominoes», которая была в детстве и запустил. Результат меня еще больше озадачил. Запись и запуск BASIC блока производился с помощью хитрых вычислений системных переменных, значения которых я не понимал и сейчас.

К стыду своему, я даже в эпоху эмуляторов не понимал, как работает машинный стек, не говоря уже о том, как функционирует BASIC, поэтому подбор «программы мечты» (*загрузка BASIC программы блоком Bytes с адреса 23755*) не увенчался успехом.

Запуск BASIC… Мощное и ёмкое выражение. А ведь и правда, при включении компьютера он и как-то запускается. Интересно, как? А что если начать изучение и анализ с самого нуля, с адреса «00000»? Может, внимательно проанализировав всю последовательность загрузки, удастся поймать суть? А немного разобравшись в структуре BASIC, есть шанс приблизиться к разгадке «вечного двигателя» и запустить блок данных с адреса 23755, над которым бился с начала 1993 года.

«Умеют же люди пользоваться этими переменными, вызывать программы из ПЗУ и стеки всякие двигать» – подумал я.

А может, дать ещё одну попытку и попробовать нормально изучить BASIC? Естественно, не для того, чтобы лепить на нём бестолковые и никому не нужные программы, а чтобы понять, как работает эта неведомая хрень. Просто закрыть недостижимую мечту детства и тему Спектрума. Тогда следующий вопрос: с чего лучше начать книгу? Конечно, с вводной главы, где рассмотреть язык BASIC с самого нуля, а вернее, с «н00000ля» и это логично.

Раз этой вводной книги нигде нет, значит, нужно её найти или создать. Ведь BASIC то как-то нужно запустить. Я пока нихрена не знаю, как подступиться. На литературу особой надежды нет. Примеров, которые я мечтаю создать, нигде не нашёл. Все хвалят книгу «*The Complete SPECTRUM ROM DISASSEMBLY*» 1983 года. Скажу честно, полистав и первоисточник и некачественный перевод от 1992 года, я не сильно вдохновился:

```
ПОДПРОГРАММА 'SYNTAX-Z'
Подпрограмма 'SYNTAX-Z' является интерполируемой. Она вызывается 32 раза, с записью
только одного байта на каждом вызове. Простой тест седьмого разряда FLAGS будет
сбрасывать признак нуля во время выполнения и устанавливать его во время проверки
синтаксиса.
Таким образом, SYNTAX задает множество Z.

2530 SYNTAX-Z     BIT   7,(FLAGS)          Проверка 7 разряда FLAGS.
                  RET                     Окончание.
```

Рис. 3. Пример из машинописного перевода дизассемблера ПЗУ.

«*Множество Z*». Бл… неужели трудно было описать процессы нормальным и простым языком с бытовыми аналогиями? Почему до сих пор не появился вольный перевод для обычных чайников, как я? И таких примеров куча не только в этой книге, но и другой относительно серьёзной литературе. «*Спасти флаги*»… Я так понимаю, что спасать их нужно от флагеляции. Ну не от Первомайской демонстрации же? Смысл комментировать вычислительную работу команд, которая и так очевидна по любому справочнику? Полезнее писать какую функцию несёт, для чего нужно задание данного значения и какой эффект получится, если его сменить.

Посмотрев на всю эту печальную картину, я решил действовать иначе. Нет, кое-что, полезного в этих книгах есть, но моя методика будет другой. Я просто открою несколько копий эмуляторов, установлю курсоры-стрелочки на адрес «0» и буду водить их по дорожкам, записывая происходящие события своими словами. Гоняя курсор по фрагментам программ десятки раз, я буду заменять значения, и смотреть, как ведёт себя

система в разных ситуациях, пока нужные куски не станут полностью понятны. Параллельно с изучением, я планирую писать этот самоучитель.

На страницах книги я введу свою компьютерную терминологию, которая может отличаться от общепринятой. Кроме того, повествование уже наклёвывается в свободном художественном стиле со стёбом в стиле блоггеров-экстремалов или раннего «Lurkmore». Для программ задумывается свой алгоритмический язык, который будет являться смесью машинных кодов Z80 с действиями в программе Spectaculator. А что так можно было? Конечно! Книга – личное творческое пространство автора и никто не ограничит полёт фантазии.

ЧАСТЬ I. КРАСНАЯ
ОСНОВЫ ПРОГРАММИРОВАНИЯ НЕ НА ЯЗЫКЕ BASIC

Ну что могу сказать? Это будет изучение основ языка BASIC несколько специфичным способом. Вводную главу I части предлагаю начать со сказки для детей о возвращении отважной Жёлтой Стрелочки в волшебный город BASIC.

Глава 2
Возвращение Жёлтой Стрелочки в город BASIC

В одной волшебной, но очень специфичной стране, родилась милая и очаровательная Жёлтая Стрелочка. Не спрашивайте как, я и сам не знаю. Включилась программа и она появилась. И не успела малышка ничего понять и подумать, как неведомая волшебная сила подхватила и куда-то потащила, прямо как в сказке «Волшебник Изумрудного Города».

Стихия исчезла также внезапно, как началась и маленькая Стрелочка очутилась в огромной Волшебной стране. Теряясь за горизонтом, вдаль простиралась ровная, как струна, дорожка. Оглянувшись по сторонам, она заметила, что стоит в самом начале пути, у клеточки пятью нулями. Вдоль дорожки тянулся забор из чисел, и в каждом его фрагменте было зашифровано какоё-то задание с головоломкой:

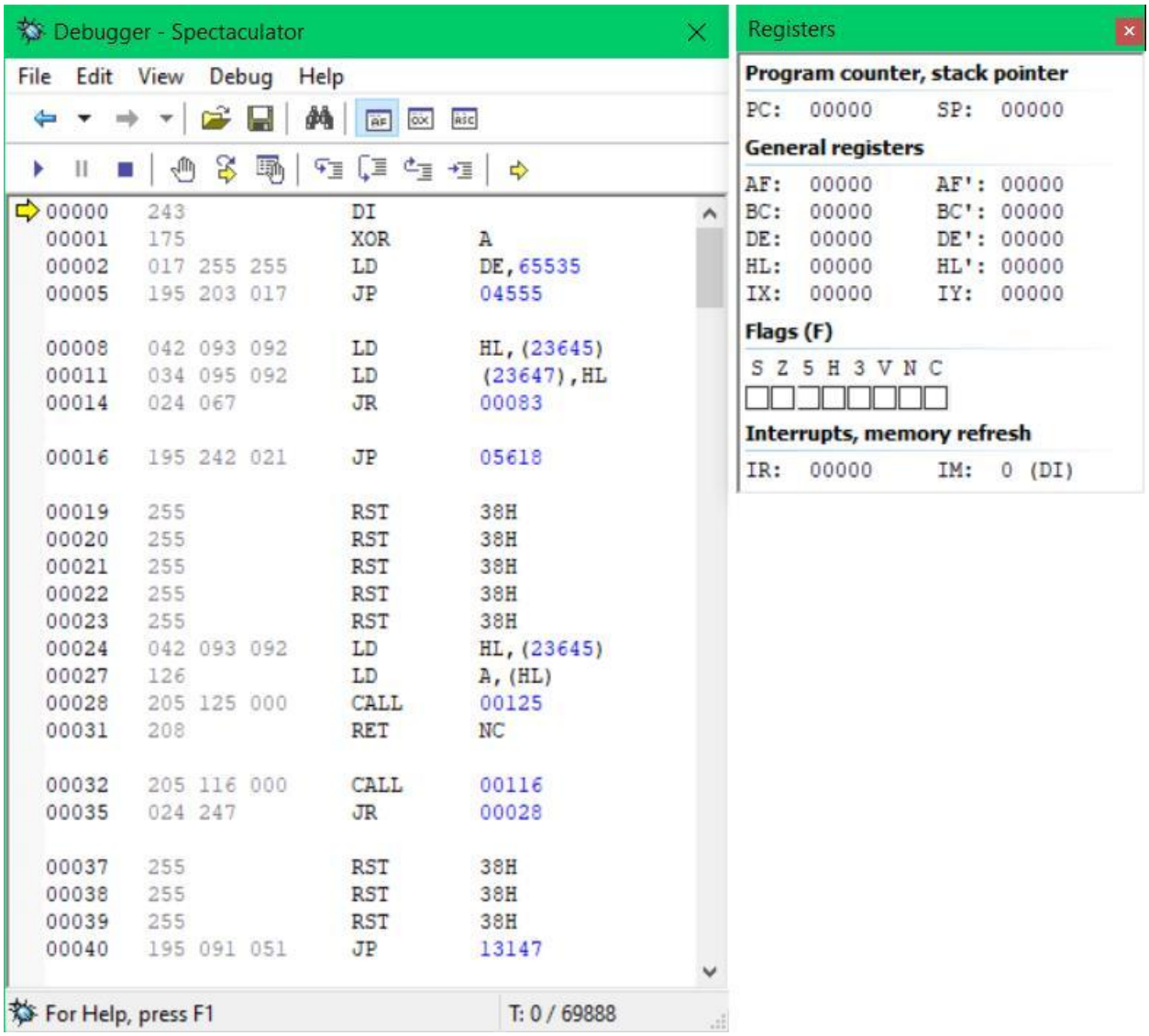


Рис. 4. Желтая Стрелочка перед адресом «00000» в начале нелёгкого пути.

Собравшись с мыслями, Стрелочка робко двинулась вперёд, по серой асфальтовой дорожке, полной загадок и неожиданностей. Едва она успела сделать шаг, и встать напротив номера 00001, как значения в окошечках «PC» с «IR» покраснели, и стали по «00001». Часики «T» в главном окошечке показали, что свой первый шаг по дорожке она совершала целых 4 волшебных секунды, а может минуты. Ясно одно: год в этой стране состоит из 69888 таких «НеМинутоСекунд»:

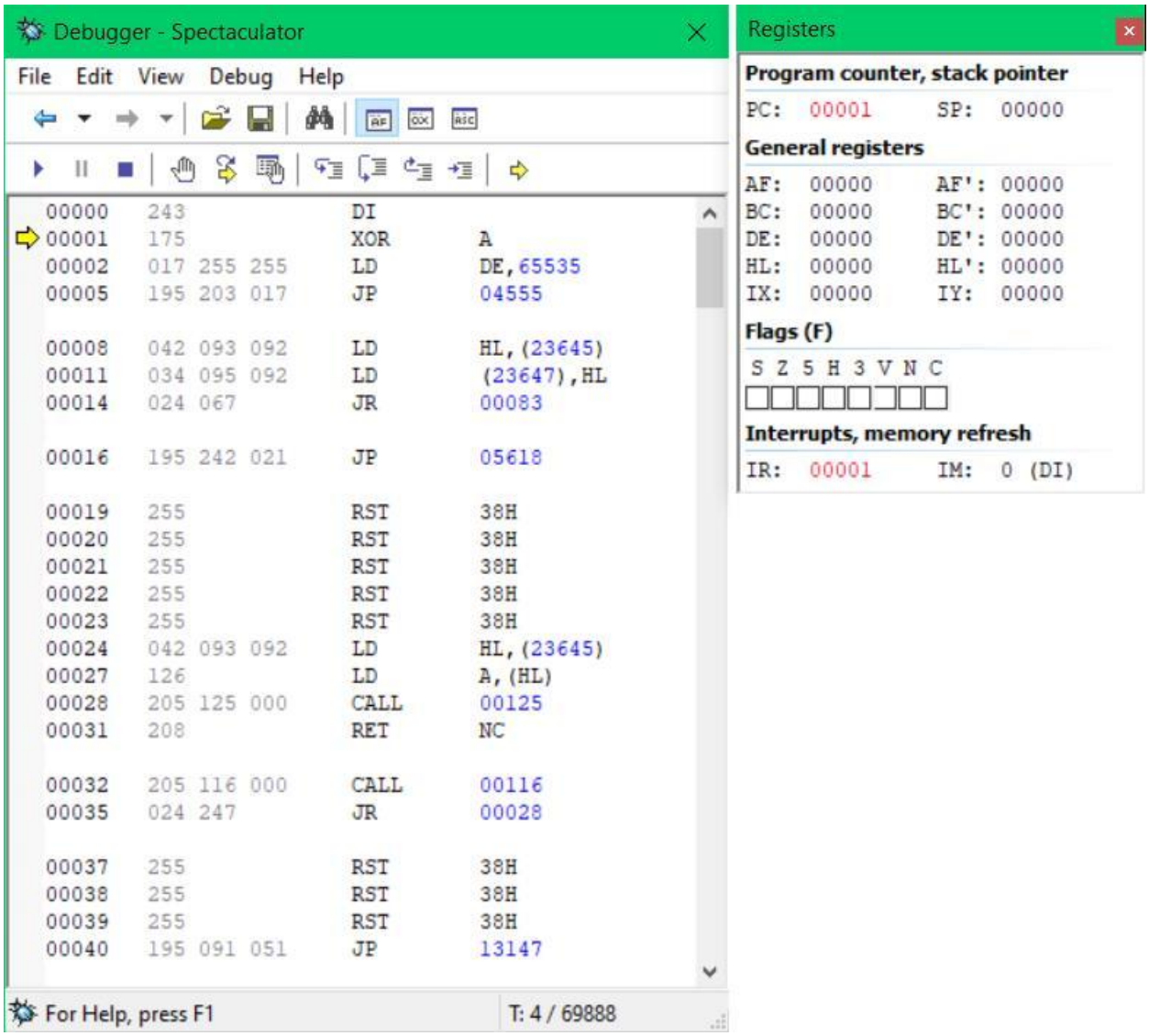


Рис. 5. Команда DI. Первый шаг Стрелочки по волшебной стране.

«Как интересно! Это, я, получается, могу управлять всеми тайнами этой волшебной страны! – подумала Стрелочка, посмотрев на изменившиеся циферки.

И вдруг её осенило: «Я и есть Царица этой бескрайней загадочной земли!»

Осознав свою сущность, первой командой Стрелочка запретила нафиг все параллельные процессы с опросом клавиатуры, выключив прерывания. Для чего, а хрен знает. Видимо потому что почувствовала, что в недалёком будущем все 49152 ячейки будут стёрты к известной матери, а прерывания могут внезапно подпортить грандиозные планы по очищению памяти.

На всякий случай обнулив «A», Стрелочка как бы намекает, что это нормальный ядрёный и конкретный сброс, а не жалкая пародия от команды «NEW». На дорожку неплохо запомнить в «HL» самую последнюю ячейку памяти 65535, которая является концом волшебной страны. Кроме того, это первый шаг подготовки к разрушительному шоу, под названием «Сброс».

На этом отведённый лимит ячеек района «RST 0» закончился. Следом начинается территория «RST 8», поэтому для развития идеи «сброса» пора сбегать с задворок подальше в самый культурный центр, по адресу 4555 и искать там свободное место:

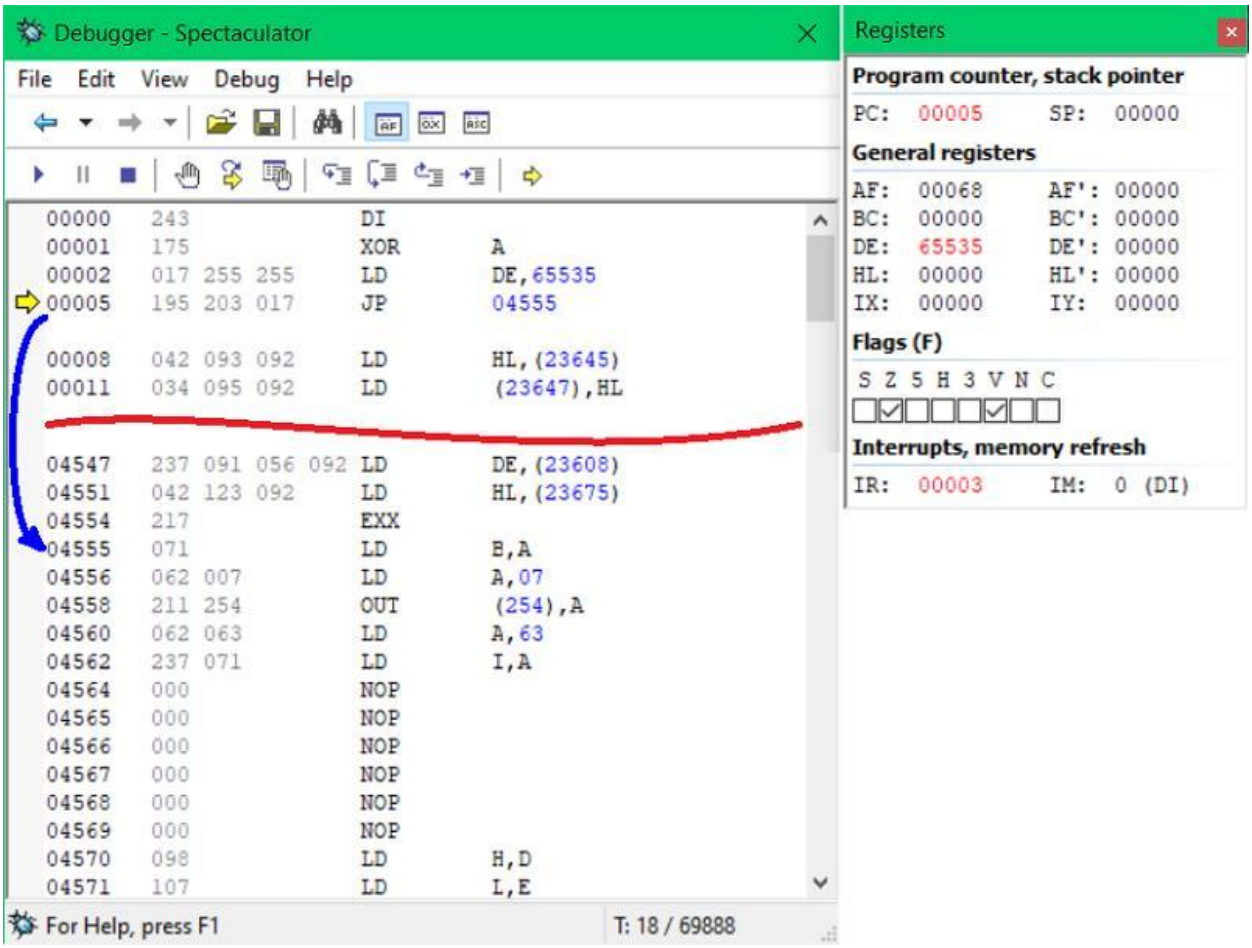


Рис. 6. Великое перемещение Стрелочки по команде JP.

Обосновавшись на новом месте, она немедленно перетаскивает в регистр «B» тип перезагрузки, в данном случае нолик, и напоминает, что это не хухры-мухры, а конкретный термоядерный сброс. Освободившейся от такой ответственности, регистр «A» сразу привлекается к выводу милой белой рамочки.

Сразу за рамочкой, «A» в качестве посредника передаёт «I» значение 63 для подготовки к параллельной работе программы обработки нажатия клавиш. Преодолевая пустырь из ноликов в 24 «Несекунды», Стрелочка берёт время подумать и осознать всю важность своей работы, чтобы сгоряча не отменить своё решение.

А дальше, с адреса 4570, Стрелочка начинает воплощать в жизнь ту самую загадочную, но знакомую всем программу...

– Блиц вопрос. Время на размышление пять секунд: назовите фамилию поэта и любой фрукт.

Пи-и-и-и-и. Время пошло.

– Пять секунд истекло. Ваш ответ?

– Пушкин! Яблоко!

А теперь переиначим на спектрумский лад:

– Что такое сброс на компьютере ZX-Spectrum? Время на раздумье 5 секунд. Ваш ответ?

– Чёрный экран с пробегающими вертикальными полосочками!

Думаю, большинство представило себе эту картину. Так оно и есть. Это тот самый «Сброс» собственной персоной.

Итак, вернёмся к путешествию отважной ➪ Стрелочки, которая идёт уже целых 90 несекунд времени в своём неведомом процессорном исчислении.

Шла она и решила скопировать размер всего сказочного мира в «HL», чтобы пользоваться копией, а оригинал положила в запасники, чтобы не повредился. Теперь Стрелочка уже понимает, что для воплощения идеи предстоит сложнейшая работа. Нужно заполнить все 49192 ячейки памяти с 16384 по 65535 циферками «2». Чтобы было не так скучно, Стрелочка начинает это делать с «конца света», то есть с 65535 и постепенно продвигаться вверх к родному заселённому ПЗУ.

...В ОЗУ разыгралась непогода. Шквал двоечек, волной накатывает снизу вверх, поглощая всё живое, и в какой-то момент доходит до видимой экранной области. От числа «02» область атрибутов закрашивается в чёрный цвет, а на экране появляются тёмно-красные вертикальные полосочки, в точку шириной:

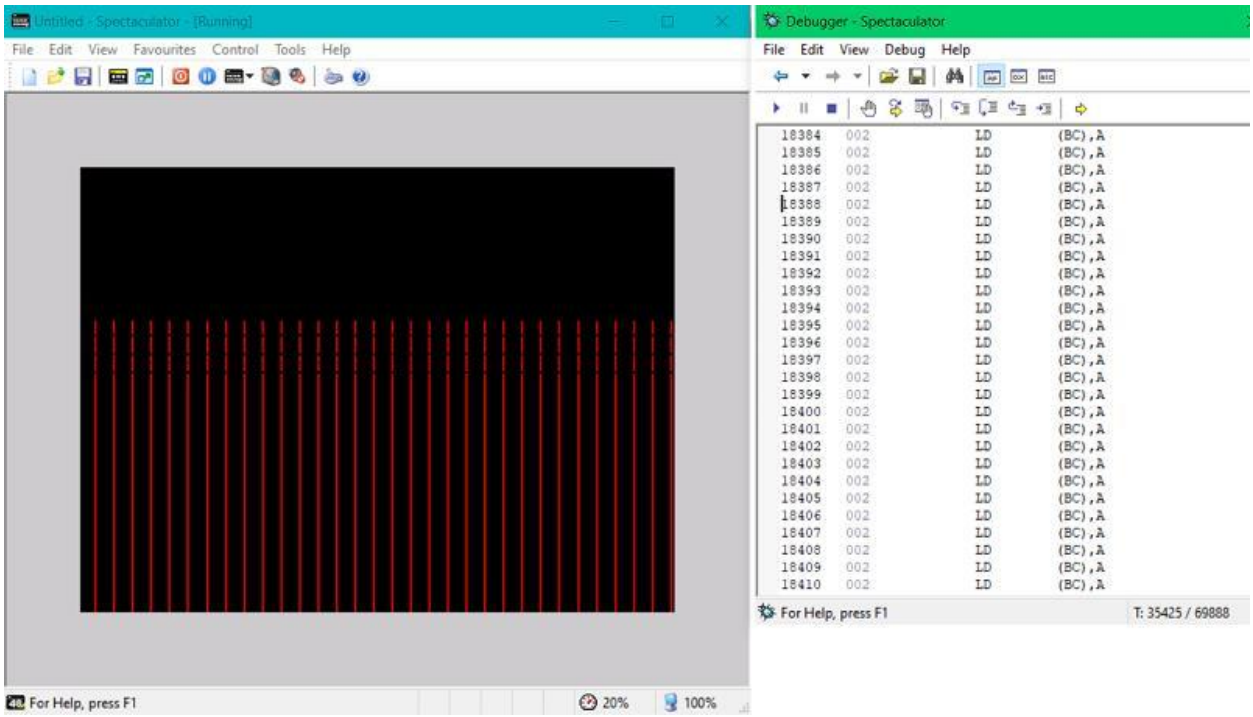


Рис. 7. Шквал сброса. Фаза-1. Сплошное заполнение памяти числом «2» снизу вверх.

Шла 35425-я сказочная секунда очередного года процессорного летоисчисления. Закончив заполнение всего мира двоечками, Стрелочка приступает ко второму акту перфоманса.

Чтобы не возвращаться назад, теперь Стрелочка решает двигаться в прямом направлении: с адреса 16384 до конца своего волшебного ячеечного мира. Проверив и убедившись, что ячейка в нужном диапазоне, она действует осторожно с чувством и толком. Сначала Стрелочка аккуратно вычитает единичку из заданной ячейки. Убедившись, что ячейка исправна и там осталось значение «1» (*при неисправности туда ничего не запишется и будет «0», а вычитая из нуля единицу, получится 255*), она повторно вычитает единичку и оставляет ячейку в покое, после чего переходит к следующей. На этом этапе с экрана начинают сходить тёмно красные полоски, но уже сверху вниз, как при стандартной загрузке картинки.

Перезаписав все возможные ячейки памяти нулями, ураган под названием «Сброс» утихает, однако экран продолжает оставаться чёрным. Ведь нули в области цветовых атрибутов экрана, дают сплошной чёрный цвет:

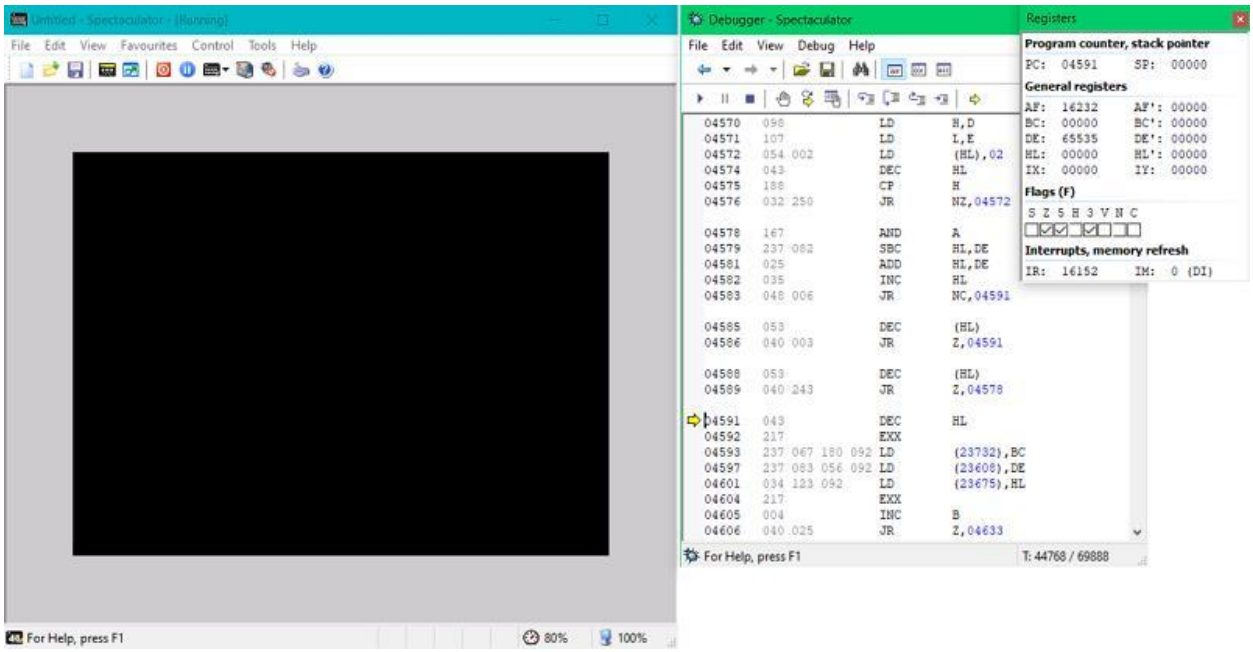


Рис. 8. Окончание «двоечного» урагана. Нулевая гладь и последствия на экране.

Волна цунами отступила, превратив область ОЗУ в безжизненную нулевую пустыню. А Стрелочка уже подводит итоги и осмысливает результаты разрушительной стихии от своего сумасшедшего эксперимента. Если по дороге попалась хоть одна битая ячейка, которая не прошла тест, то в HL запоминается самый последний адрес памяти, который прошел проверку без разрывов с 16384.

Вернёмся в цивилизованные цветущие края ПЗУ. К адресу 4592 вся волшебная страна начищена до блеска. Кругом сияют одни нули. Пора готовить инфраструктуру для жизнеобеспечения главного города страны – BASIC системы. Предстоит еще столько дел: нужно расставить системные переменные, перенести шрифт в буквенные символы графического режима, создать столбик машинного стека…. В общем, работы куча и нужно приступать к приятным подготовительным хлопотам.

Но не всё так просто. Стрелочка, конечно, пытается сохранить три первых системных переменных `P-RAMT`, `RASP/PIP` и `UDG`, но на проверку все они оказываются пустышками. Эти переменные сохраняются лишь в том случае, если путешествие начинается с лёгкого сброса по команде «`NEW`»:

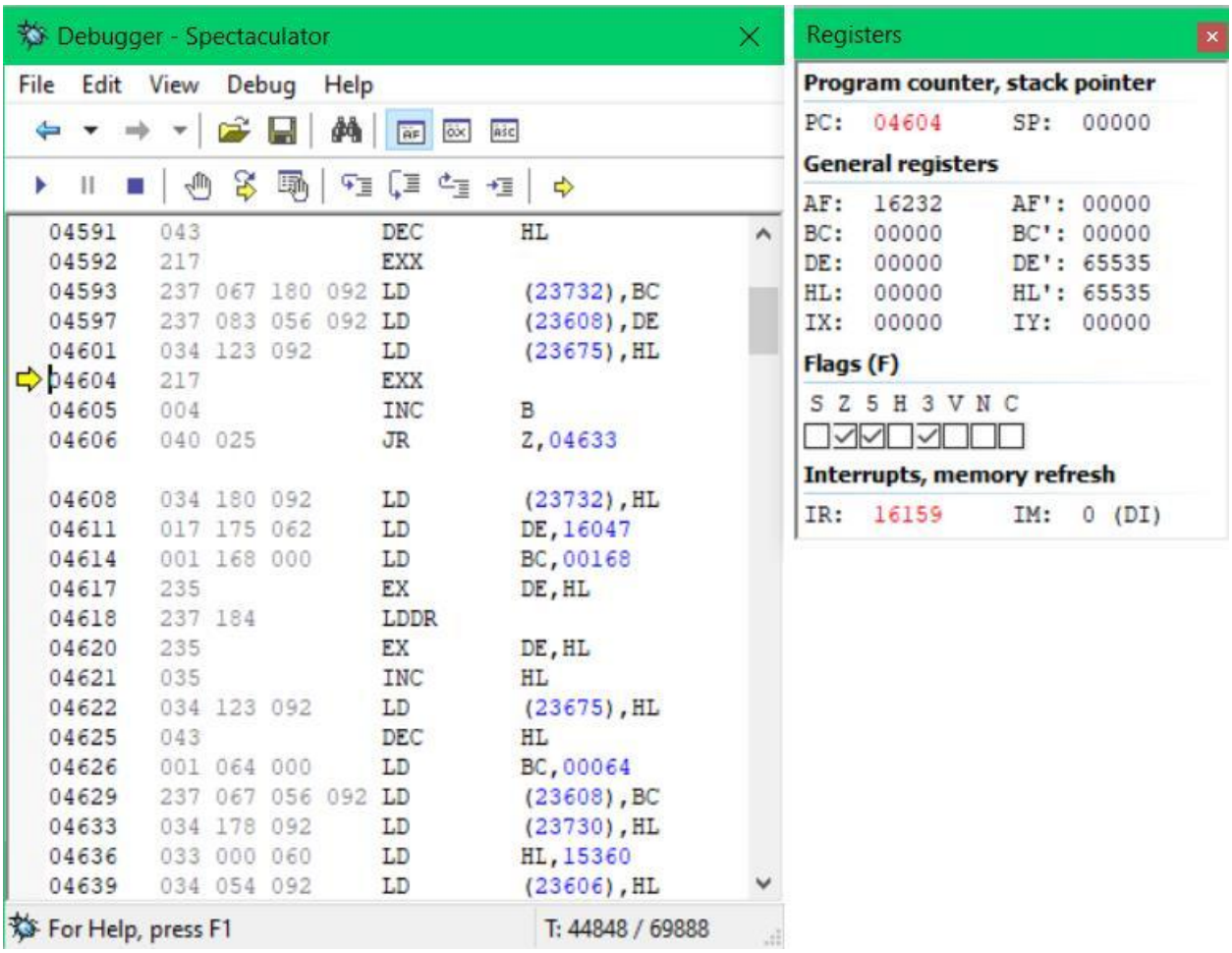


Рис. 9. Бесполезное сохранение пустых значений.

А дальше Стрелочка неожиданно спохватывается, что делает фигню и проверяет «B», в который заблаговременно скопировала напоминалку, какой именно сброс происходит, и по какой схеме планировки расставлять элементы ландшафта. Ведь она так заработалась на очистке памяти в одну и другую сторону, что совсем забыла об остальной инфраструктуре! Вспомнив, что это серьёзный сброс, она продолжает идти по ПЗУ и выполнять нескончаемый список подготовительных заданий.

С 4608 начинаются работы по обустройству первых ячеек памяти в области системных переменных. Самой первой формируется `P_RAMT (23732)`, а записывается туда значение последней исправной ячейки памяти, которая идёт без разрыва с первой точки области экрана (16384). В эпоху «псевдо-спектрума» это число всегда будет 65535:

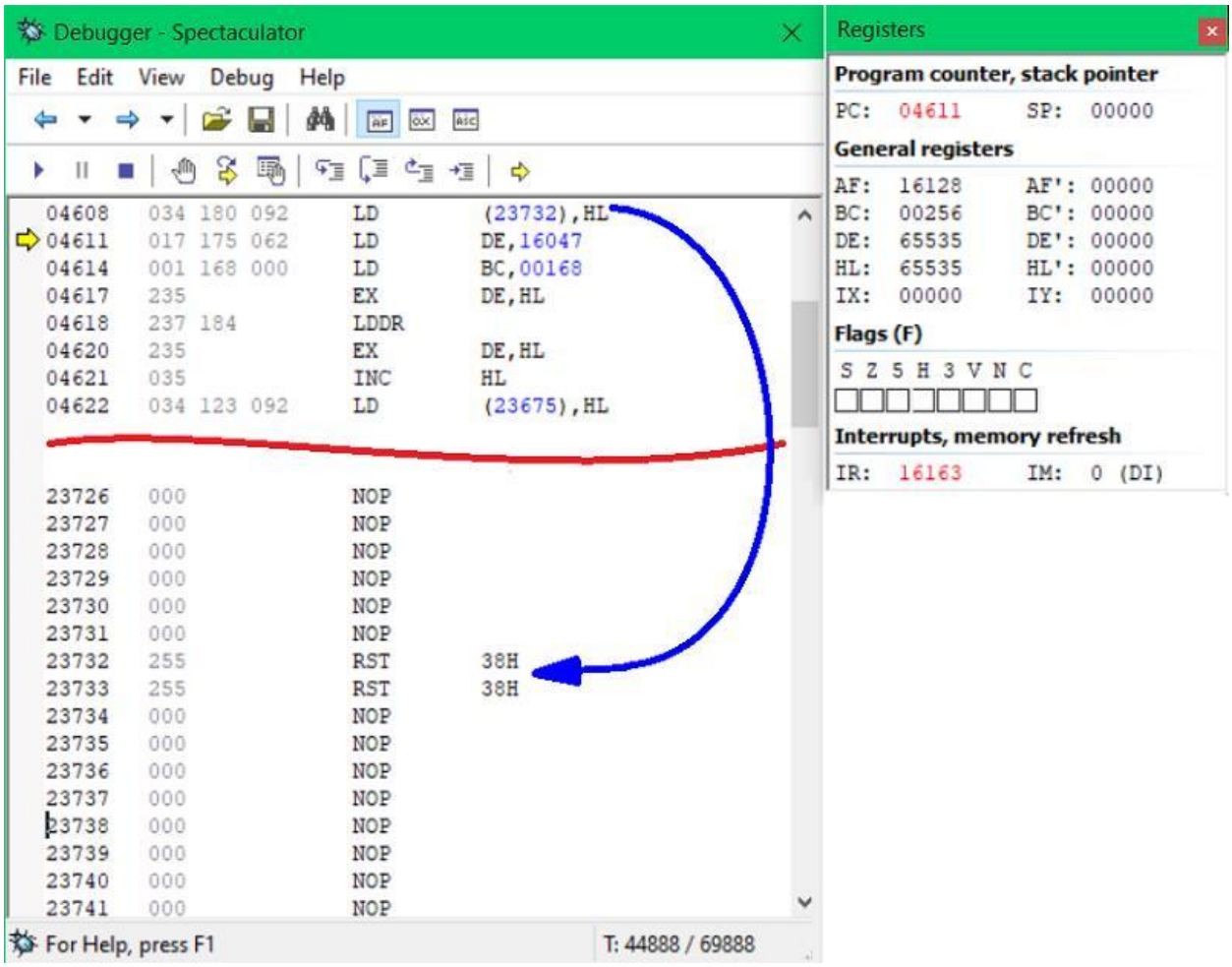


Рис. 10. Первый предмет интерьера «P_RAMT» в области системных переменных.

Следом, из 16047, берётся 168 байт рисунков буковок, и снизу вверх копируются в память с 65535 по 65368-ю ячейку. Таким образом, рождаются символы от «A» до «U» для графического режима.

Обустроив мир курсора , немедленно создаётся переменная `UDG (23675)` с адресом размещения этих буковок. Записав число 64, формируется звуковой параметр `RASP (23608)`:

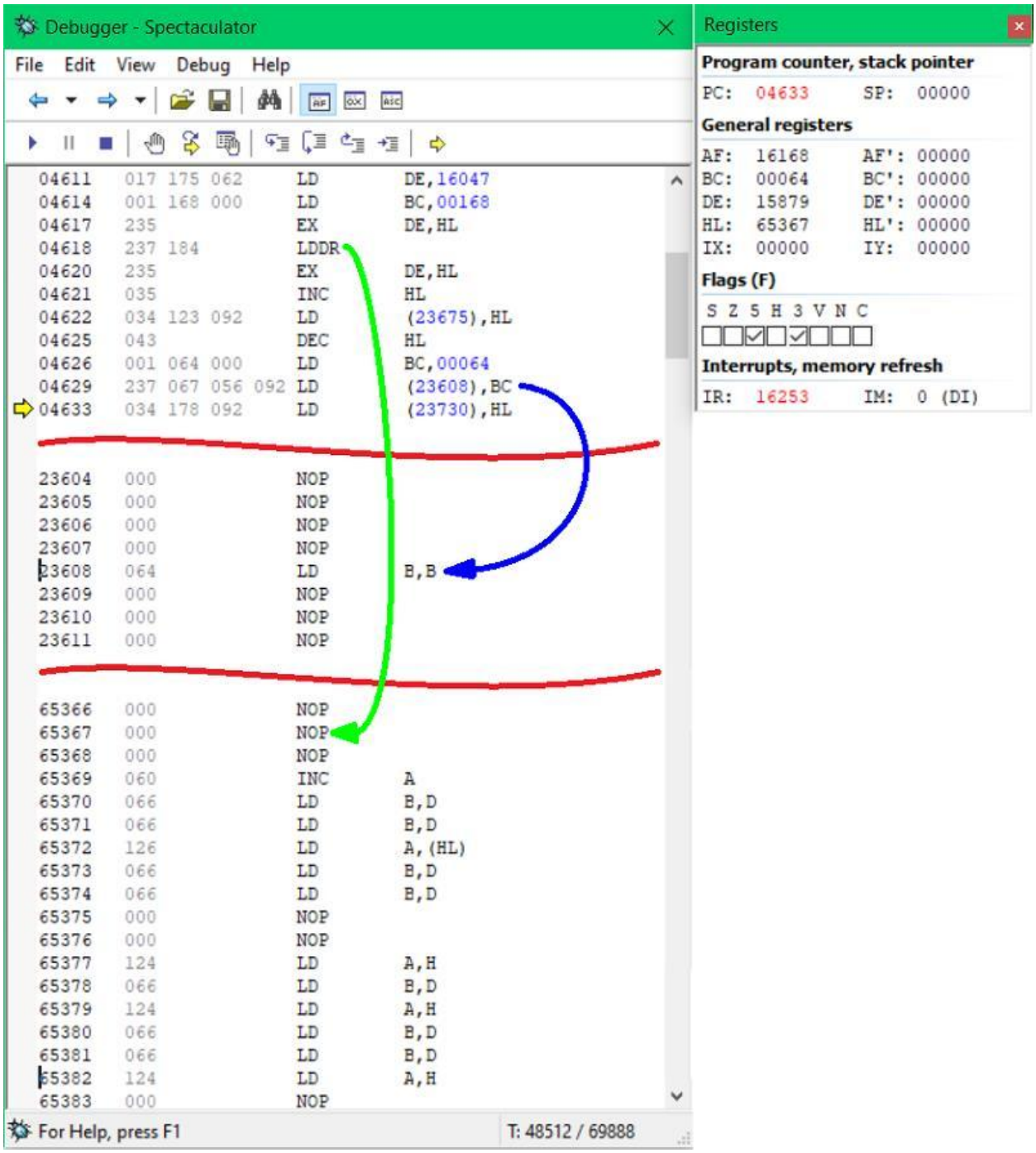


Рис. 11. Формирование области графических символов UDG.

Путём вычитания единицы из адреса размещения буковок в графическом режиме, рождается `RAMTOP (23730)`. Он же – будущая граница, которая отделит основание многофункционального стека от шрифта `UDG`, и обеспечит ему надёжную защиту.

Попутно создав ячейки с адресом размещения обычного шрифта `CHARS (23606)`, Стрелочка приступает к разметке самого важного элемента для будущей бесперебойной работы BASIC системы – башенке машинного стека.

Из недавно созданной переменной `RAMTOP (23730)` она берёт сохранённый адрес, и над крышей UDG чертит разделительную границу в виде маркера «62». А как же иначе? Вдруг из BASIC кто-то захочет выполнить команду «`GO SUB`» (`ГО СУБ`). Как потом узнать, где кончаются результаты промежуточной работы этой команды?

Поднявшись на этаж выше, Стрелочка присваивает регистру «SP» своё первое значение. Он становится 65366, установившись на прочерченную разделительную границу `RAMTOP (23730)`, но при всём этом приготовился принять свои первые значения в две

вышестоящие ячейки (65365/64). Поднявшись вверх на два этажа, Стрелочка создаёт очередную системную переменную `ERR_SP (23613)`, присвоив ей адрес 65364 и поставив на 2-й этаж пустого машинного стека в качестве подкладки. Таким образом, первое записанное значение в SP-башенку, будет просматриваться из комплекта ячеек 23613/14.

На этом формирование стека заканчивается. Башенка спроектирована и готова к росту вверх, но пока останется невысокой:

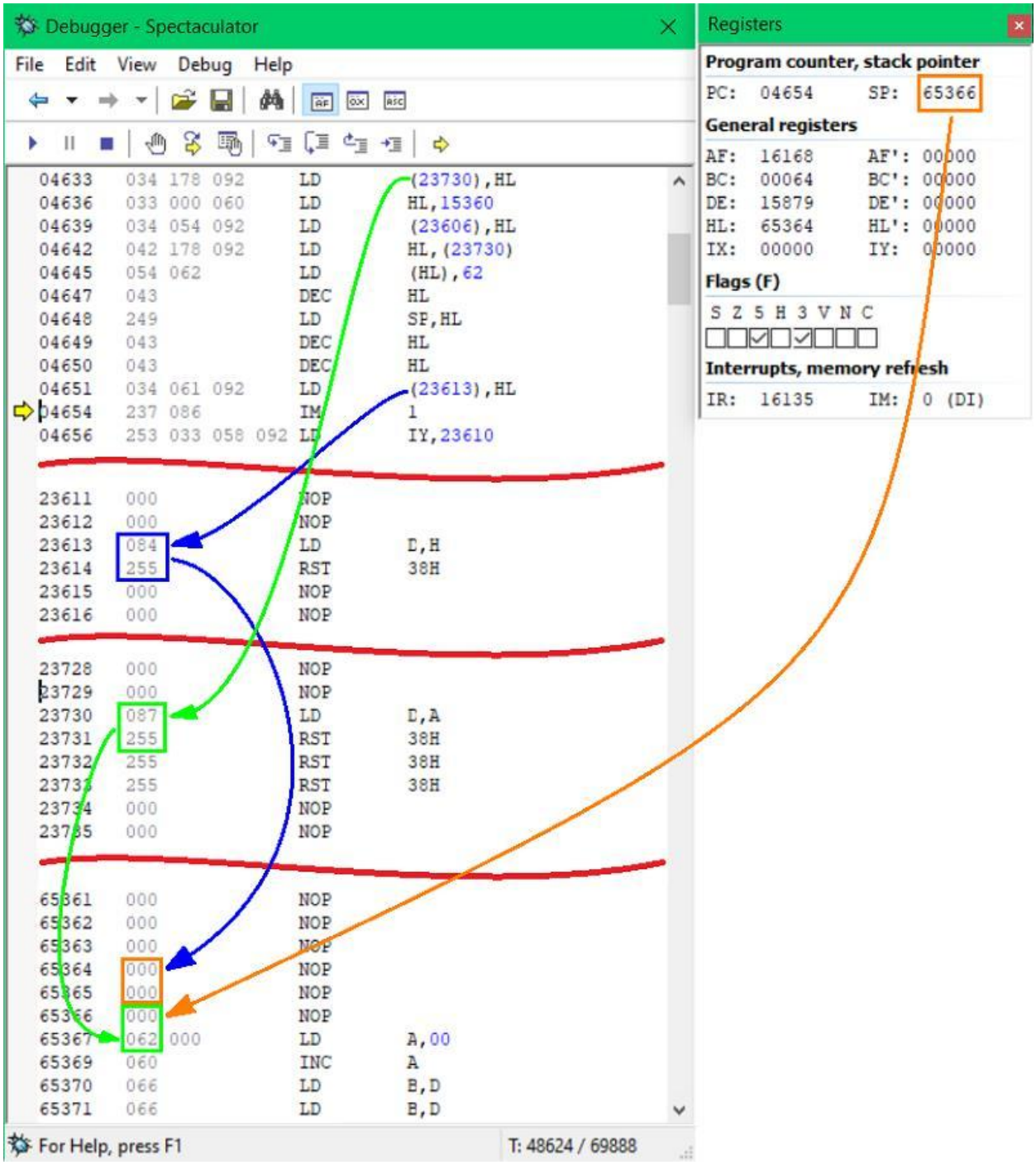


Рис. 12. Процесс формирования чистого машинного стека.

На отсечке 4654 Стрелочка включает режим прерываний `IM 1`, а следом «IY» приравнивается к 23610, чтобы было удобно обращаться к будущему массиву системных переменных как напрямую, так и по IY+xx IY-xx. Разрешаются прерывания для параллельной работы под опрос клавиатуры и счётчика времени компьютера. С этого момента нажатия клавиш чувствуются программами.

Начинается формирование переменной `CHANS (23631)` с переносом 21 байта из ПЗУ (5551) в адреса 23734 по 23754. После перетаскивания всей этой мишуры создаётся переменная `DATAADD (23639)`, которая указывает на последний байт перенесённой таблицы:

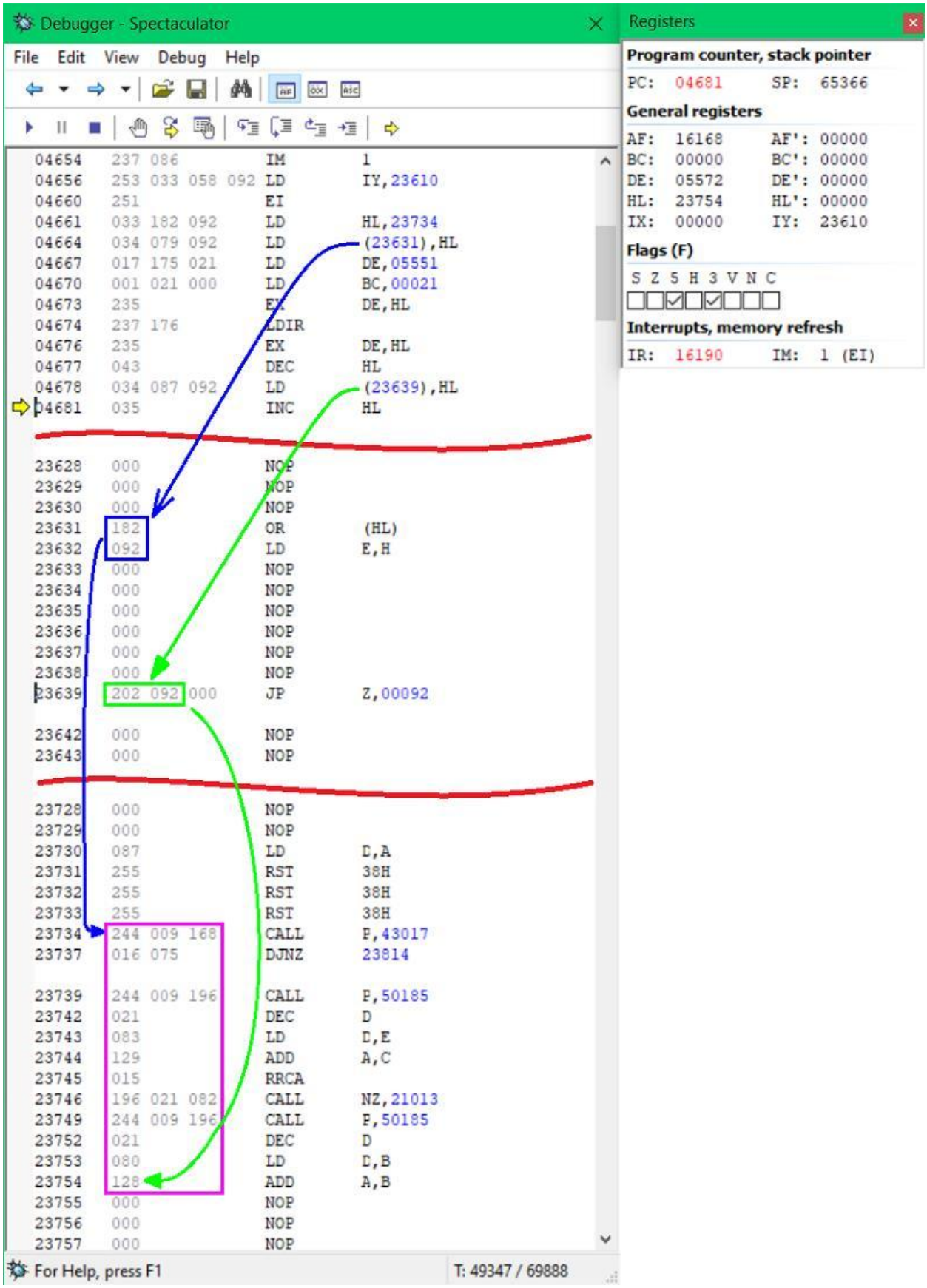


Рис. 13. Рождение и схема массива CHANS-DATAADD.

И наконец, Стрелочка подошла к самой долгожданной цели. В 4681 прибавляя единицу к предыдущей переменной, получается верхняя граница комплекса массивов для будущих BASIC областей, которую предполагается поместить в подвал под канальный склад CHANS-DATAADD (23734-23754).

Не задумываясь, Стрелочка расставляет сразу две переменных PROG (23635) и VARS (23627), которые пока указывают на одну и ту же ячейку 23755. В эту ячейку временно втыкается маркер «128», чтобы как-то отделить художество, от соседних областей.

Шагнув на следующую за маркером ячейку, Стрелочка создаёт переменную E_LINE (23641), куда собирается помещать набираемые на экране, и выполняемые безномерные строки BASIC программы. Далее за этой областью заботливо устанавливается код ENTER и очередной маркер-разделитель «128».

А Стрелочка вошла в азарт и следом сформировала еще три переменных WORKSP (23649), STKBOT (23651) и STKEND (23653). Три последние переменные также указывают пока на один и тот же адрес 23758 с голым нулём внутри. Так заканчивается разметка и формирование будущих областей для обслуживания программных строк BASIC:

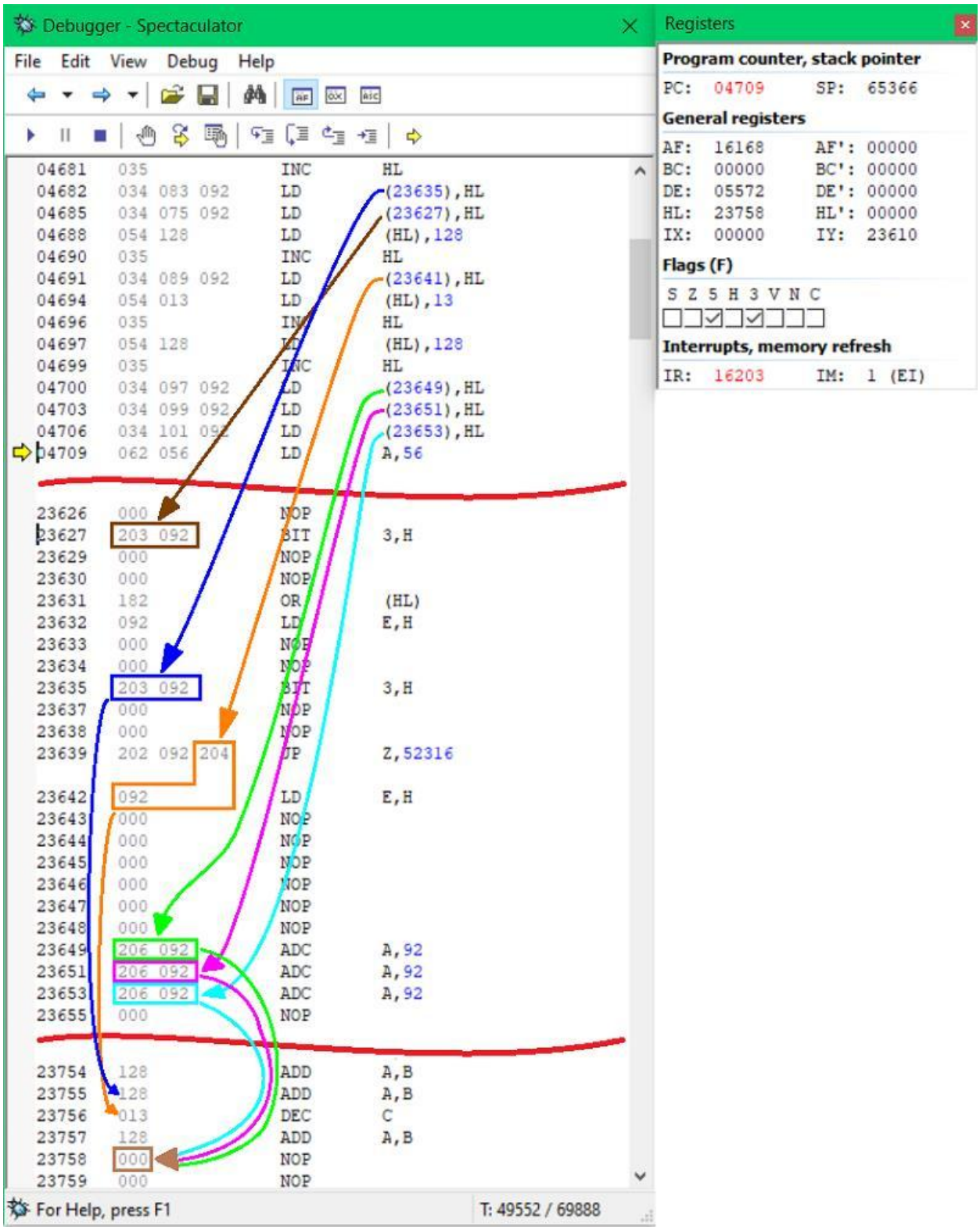


Рис. 14. Создание переменных и процесс временной разметки областей под BASIC строки.

Всё трудится Стрелочка и готовится к обустройству пригородов BASIC, что даже про экран забыла. А он так и остаётся чёрным. Пора заняться подготовкой и уборкой экранной области. В «A» задаётся комплект цветов «белый экран чёрный текст» комплексным значением 56, после чего создаются три цветовых переменных `ATTR-P (23693)`, `ATTR-C (23695)` и `BORDCR (23624)`. Между делом формирует `REPDEL (23561)` для задания задержки повтора печати при зажатой пальцем клавише.

Для чего задавали IY на массив системных переменных? А для того, чтобы понтово ими воспользоваться и козырным методом вычитания из нуля получить пару 255-х

значений, тем самым создав переменную KSTATE (23552), которая, КСТАТИ, достаточно массивная и простирается до адреса 23559.

А дальше снова читерство и плагиат. Опять из ПЗУ с адреса 5574 копируется 14-ти байтная табличка с переменными для каналов и потоков STRMS (23568). Хитрая какая, лишь бы заполнить и так небольшое ОЗУ тем, что уже имеется. На самом деле в ПЗУ реального Спектрума много не поменяешь, вот и приходится плодить копии для текущей работы:

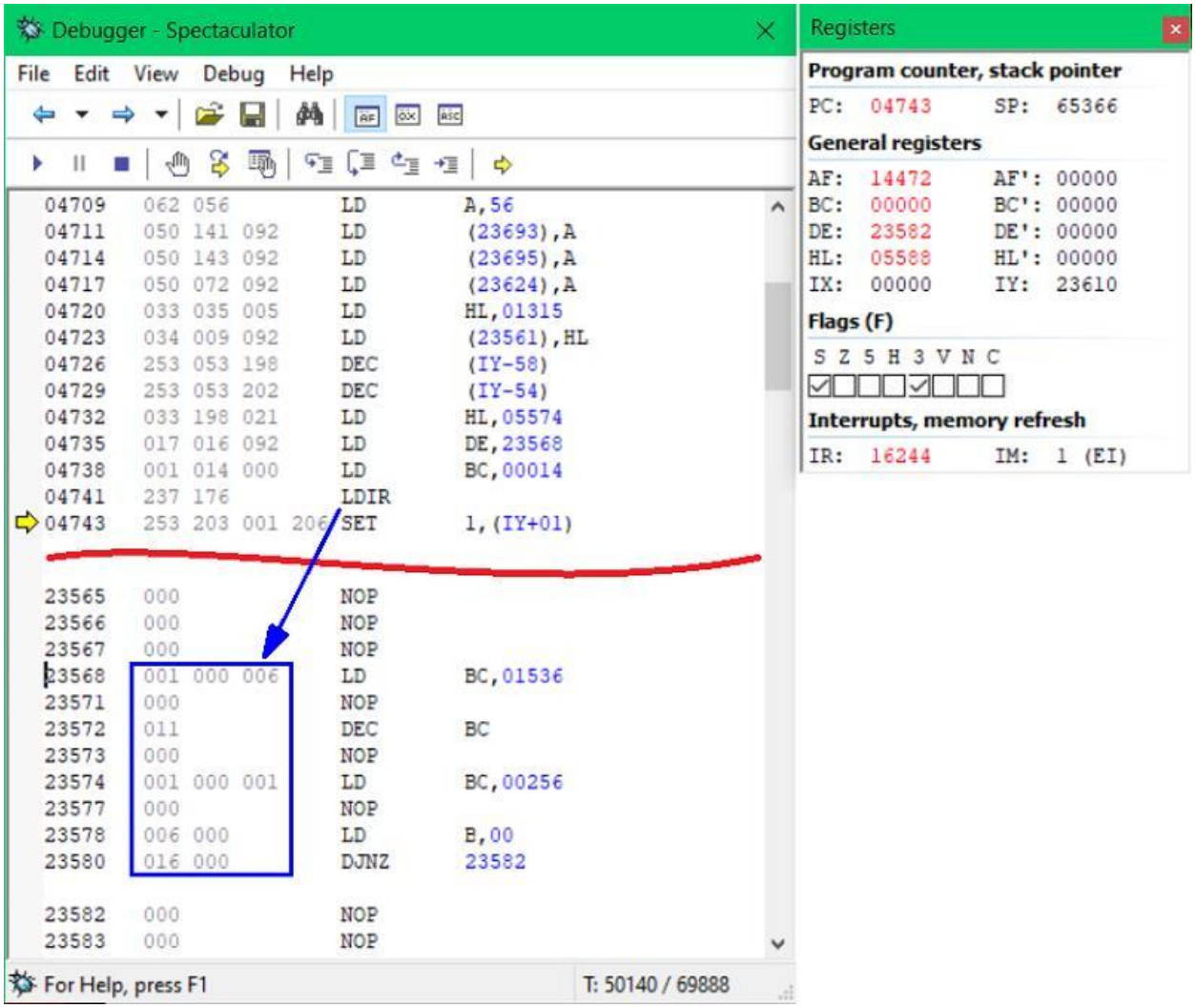


Рис. 15. Создание копии массива STRMS (23568-23581).

В переменной FLAGS (23611) Стрелочка включает бит-тумблер №1 принтера и убегает на подпрограмму очистки буфера принтера CLEAR_PRB по адресу 3708. Также быстро и без сюрпризов возвращается назад. Хотя смысл этого не совсем понятен. Ведь после сброса область с 22296 и так чистая. Ах да, программа, то универсальная! Видимо для пути с команды «NEW» задумывалась уборка этих мест.

А вот дальше Стрелочка откусывает от экранной области две нижние строки и передаёт их во владение буфера редактора путём создания очередной переменной DF_SZ (23659), в которую сразу вписывает значение 2.

Наконец-то Стрелочка вспоминает, что после разрушающего шквального сброса, над миром продолжает нависать тьма, Недолго думая, она заглядывает в комплексную подпрограмму CLS по адресу 3435. Теперь Стрелочка уже ни на что не отвлекается и доводит дело до конца. Она не только очищает экран, но еще и задаёт цвета в соответствии с ранее установленной переменной ATTR_P (23693). После такой нелёгкой

работы, Стрелочка возвращается на главную дорожку заданий, от продвижения по которой благоустраивается волшебный город BASIC.

Экран чистый, вроде всё хорошо, но не хватает десерта в виде приветственного сообщения:

```
© 1982 Sinclair Research Ltd
```

После обнуления «A», Стрелочка отыскивает ящичек, в виде адреса размещения сообщения (5432). Достав оттуда материал для оформления, она записывает его в «DE», и отправляется на поиски ближайшего фотоателье «Message Printing» по адресу 3082, чтобы напечатать этот скромный рекламный плакатик.

Остаётся последний штрих. Для блокировки локального обновления буфера редактора и приостановки выдачи мигающего курсора раньше положенного времени, Стрелочка включает бит-тумблер №5 на пульте управления `TV_FLAG (23612)`.

И вот неподалёку виднеется город или комплекс программ `MAIN EXECUTION`, который является сердцем BASIC системы. Впереди располагаются шикарные въездные ворота (4770), но помня прошлую инструкцию, Стрелочка проходит мимо них. Вскоре обнаруживаются ещё одни ворота в волшебный город BASIC (4777). Они оказываются не такие роскошные. Недолго думая, Стрелочка направляется к ним и входит в город:

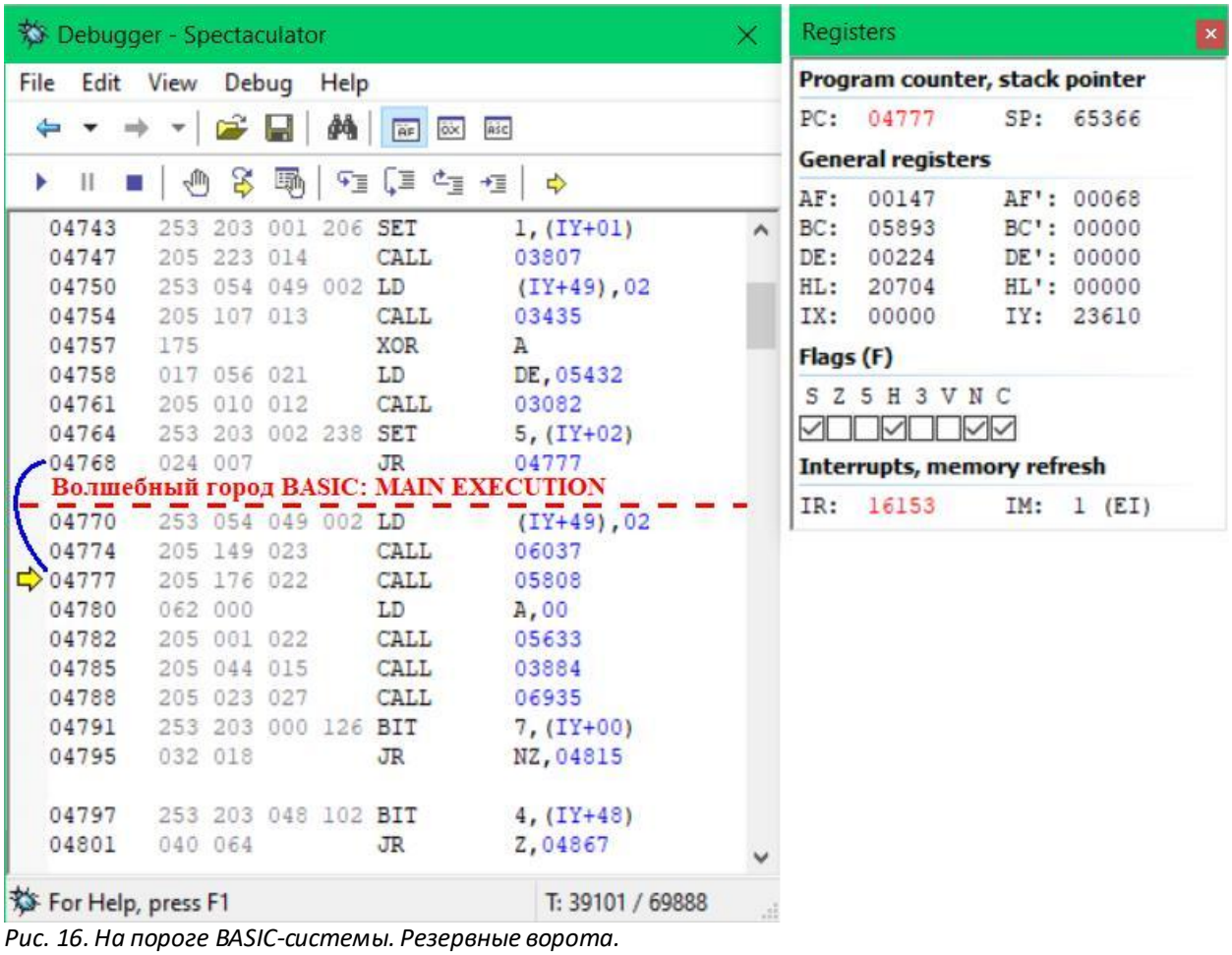


Рис. 16. На пороге BASIC-системы. Резервные ворота.

Глава 3
Подготовка «BASIC-носителя» к запуску

Краткое содержание: Spectaculator, настройка режима 48K, возвращаясь к напечатанному

– Ну что чуваки, вам понравилось?

– Да, да, да-да-да, да-да, да-да, дада!

Тогда предлагаю сварганить что-нибудь простенькое. Только имеются некоторые сложности. Запуск бейсика придётся начать не с повторения некоторых моментов предыдущей вводной части, а с установки имитатора компьютера ZX-Spectrum на платформу IBM PC. Программа, в которой планируется делать эксперименты,, называется Spectaculator. Конкретно в этой книге я буду пользоваться версией 7.51.1761 под Windows 10 (*версия 20H2 сборка 19042.631*).

Процесс установки и «лечения таблэтками», как говорят в торренте, придётся пропустить. Нет, мне не жалко объяснить, просто я установил программу заранее, до создания этой книги. Зная свой старенький компьютер, рисковать с переустановкой, ради нескольких поясняющих картинок не хочется. Изображать что-то по памяти тоже не буду, чтобы не вводить в заблуждение, если информация окажется неточной. Процесс инсталляции там не сильно сложный и есть поясняющие инструкции на каждом шагу.

Тогда начну сразу с настройки агрегата. Вот вы установили программу и запустили spectaculator.exe. После цветных квадратиков и «сброс-перфоманса», экран побелеет и в центр выскочит подобная заставка:

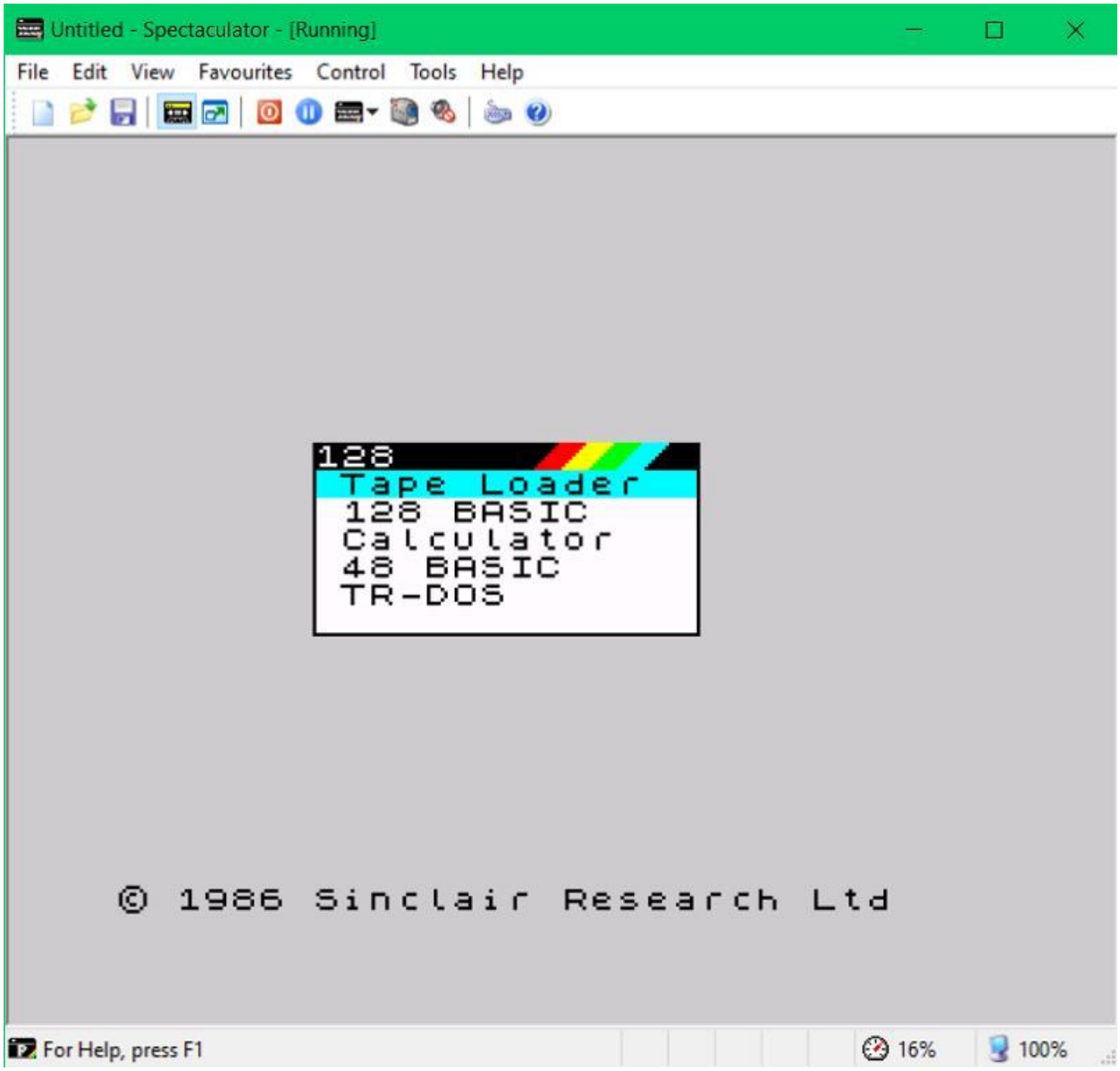


Рис. 17. Spectaculator. Типовая заставка режима 128К.

Дело в том, что все примеры программ, которые будут в книге, рассчитаны на чистый режим «Spectrum 48K» без всякой периферии, поэтому некоторые галочки в пунктах настройки, могут хорошо попортить нервы.

Первым делом предлагаю избавиться от режима 128К. Для этого нажмите кнопку «*Control*» и наведите мышку на пункт меню «*Switch Model (Ctrl+W)*»:

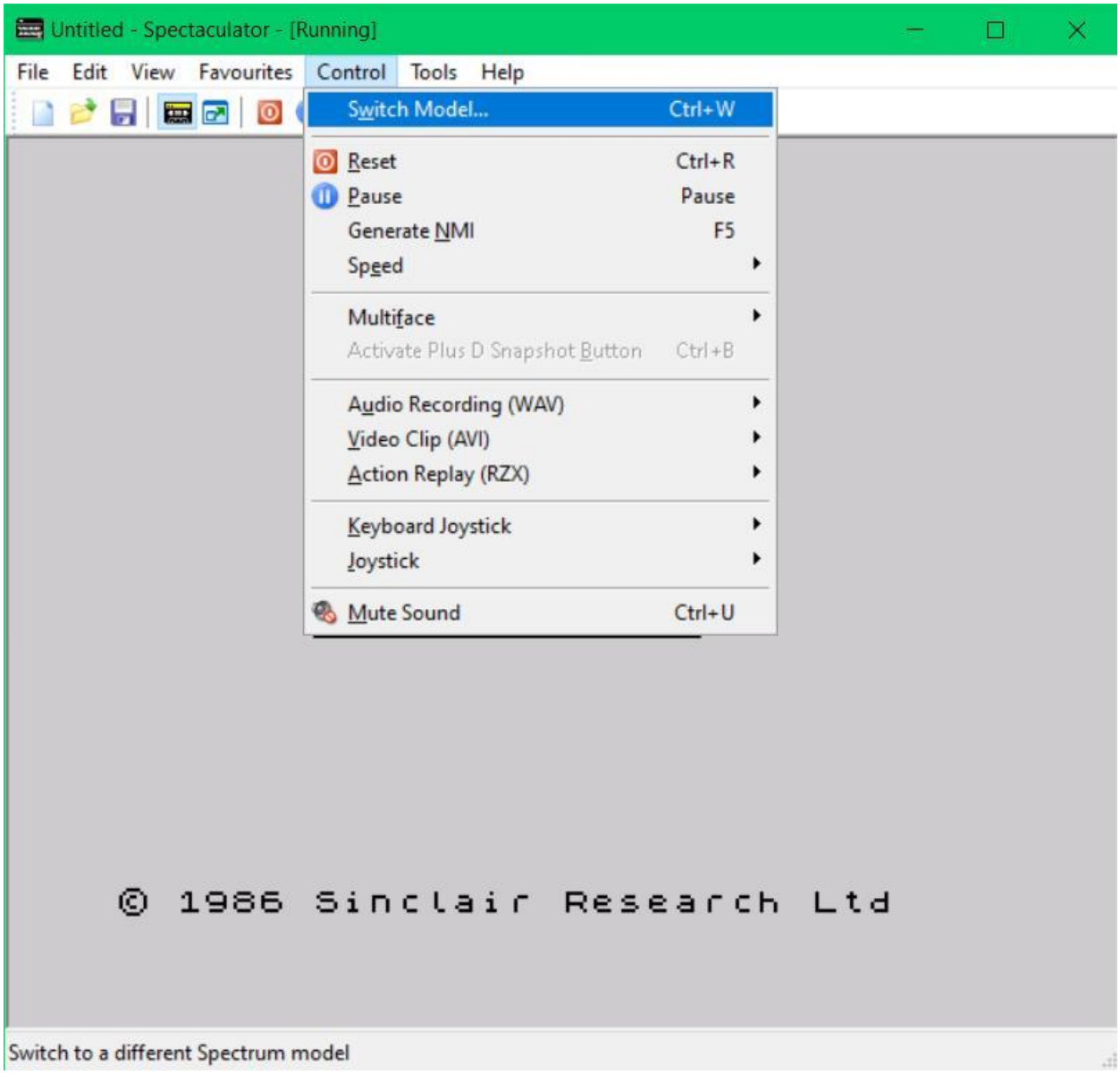


Рис. 18. Spectaculator. Меню выбора модели компьютера.

Нажмите на него правой кнопкой мыши. Откроется дополнительное маленькое окошко, внутри которого будет список базовых моделей компьютеров. Курсорными клавишами или кнопкой мыши выберите пункт меню «*48K ZX-Spectrum*»:

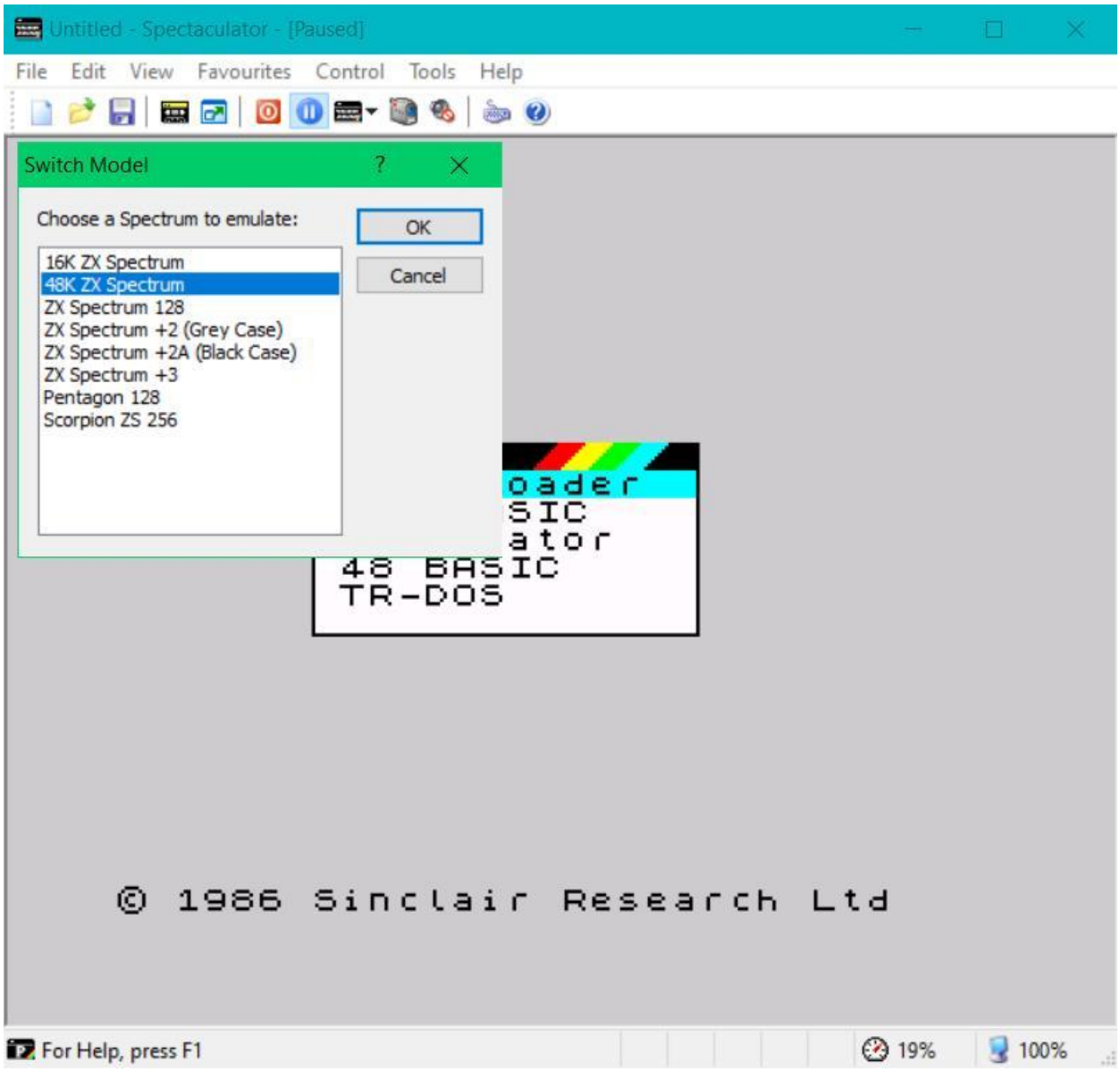


Рис. 19. Spectaculator. Выбор модели компьютера.

Нажмите кнопку «*OK*». Программа на пару секунд задумается и вскоре после очередного сброса на экране появится знакомая заставка:

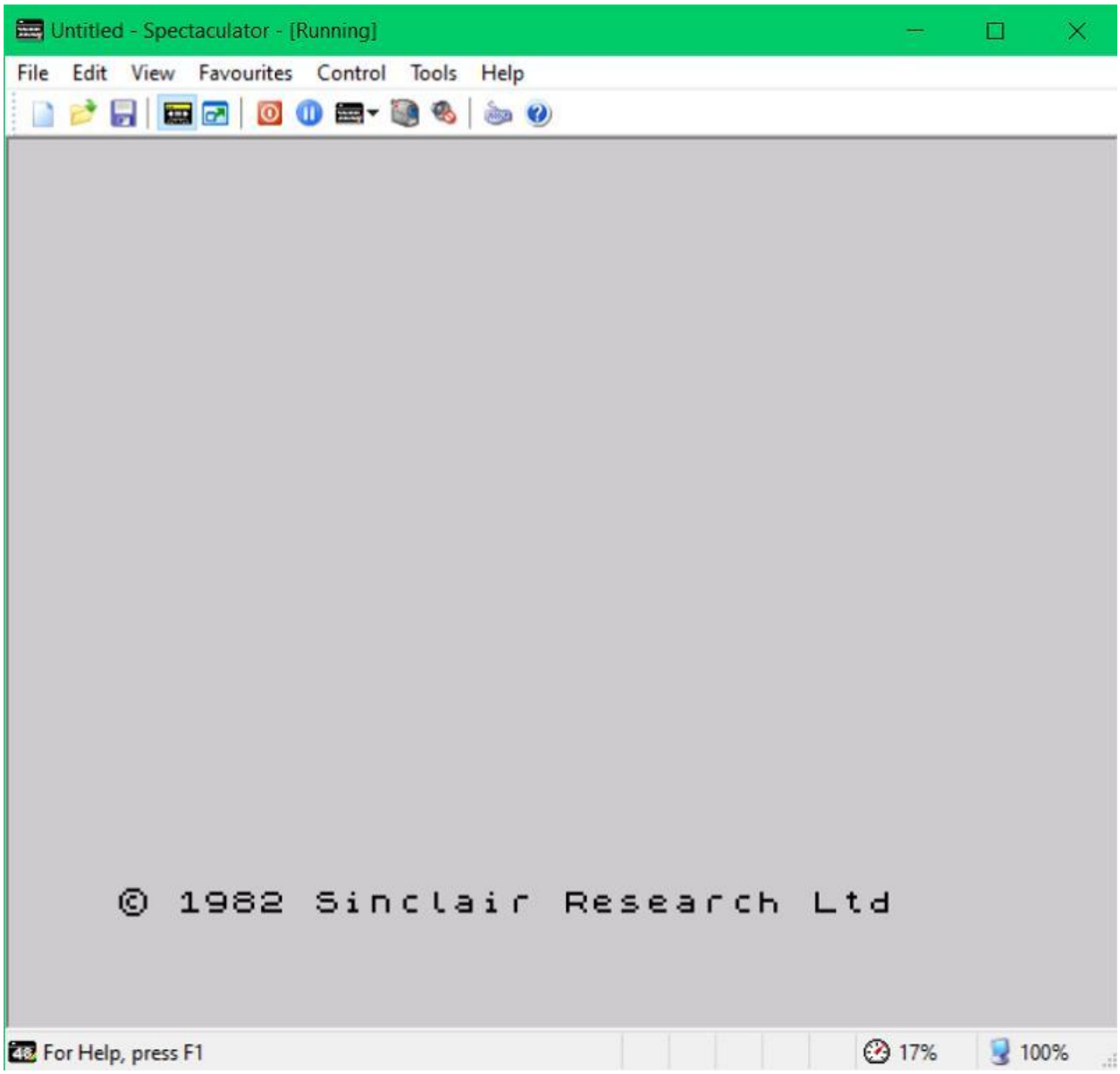


Рис. 20. Виртуальный ZX-Spectrum 48K в режиме ожидания и готов к работе.

На первый взгляд всё, и можно начинать, но вспоминается один недосказанный момент в моих старых исследованиях. В журналах «*ZX-Ревю*» была такая занятная рубрика «*Возвращаясь к напечатанному*». Вот и я дошел до такого торжественного момента. Предлагаю возвратиться в 2013 год, и посмотреть в моё первое издание книги:

Кстате говоря, вместо RANDOMIZE, перед USR можно поставить некоторые другие команды, требующие после себя переменную. Например: PAPER USR 0, BORDER USR 0, BEEP USR 0, и так далее. Главное, чтобы «скормить» USR 0, и заставить его выполняться, а далее уже запускается машинная программа, игнорируя переменные, требующиеся для этих команд в BASIC-режиме.

А вот после выполнения такой машинной программы, если предусмотрен выход в BASIC, то задним числом получаем сообщение об ошибках и некоторые глюки. Второй раз программа с этого адреса уже не запустится. Все области бейсик системы сместятся на несколько десятков байт вниз.

Такой метод можно использовать для одноразового использования, когда после запуска загрузится машинная программа, выход из которой осуществляется только через кнопку *RESET*.

Некоторые команды, в сочетании с USR, ведут себя более корректно. Например: CLOSE #USR 0 и GO TO USR 0 можно запустить повторно. Но в любом случае происходит смещение BASIC программы вниз, за область ввода строки, а используемая ранее область становится заброшенной и теперь бездействует.

Почему USR 0, а не USR 23759, например? В старые времена этот метод применялся в качестве защиты, чтобы сбить с толку начинающего любителя взламывать игры. Ведь если такую строку взять на редактирование, а затем снова ввести, то значения обнулятся, и ноль станет нулем. Потребуется снова туда записывать значения, чтобы при запуске программы не произошел настоящий переход по адресу 0, то есть сброс.

Рис. 21. Ретро фрагмент из книги 2013 года.

Одни команды смещают область, другие нет. И всё на полном серьёзе, как будто, так и надо. Так и хочется крикнуть самому себе в прошлое: а тебя не смутило это «смещение»? А в ответ – тишина. Ну вот, дожил. Уже сам с собой начал разговаривать.

На самом деле, если бы я тогда хоть немного разобрался даже не в работе ПЗУ, а характере и содержимом переменных, то понял бы проблему. Ну не должны эти области сдвигаться от синтаксических ошибок.

Но сейчас 2024 год, поэтому давайте поставим точку в этой проблеме. В самом вверху окна найдите кнопку «*Tools*» и нажмите на нее. В открывшемся меню последним пунктом стоит «*Options» (Ctrl+Y)*:

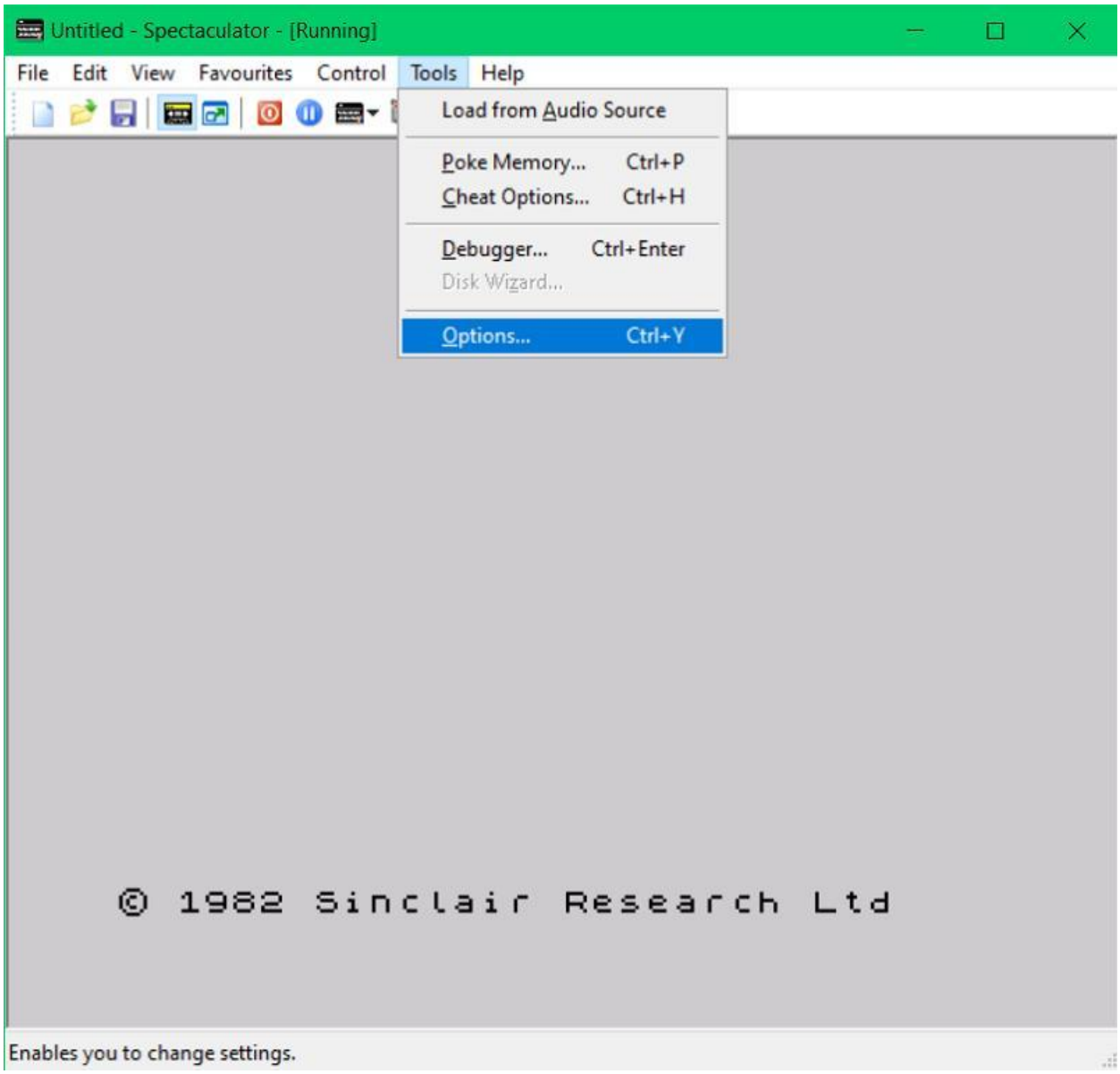


Рис. 22. Spectaculator. Пункт «Tools». Открытие меню настроек.

Нажав на пункт «*Options*» (или комбинацию *Ctrl+Y*) откроется окошко настроек «*Spectaculator Options*». По умолчанию, при первом открытии должна высветиться вкладка «*Hardware*» (*в описываемой версии программы точно*).

А теперь смотрите внимательно внутрь окошка, где в рамочке под надписью «*Additional Hardware*» расположен список виртуальных устройств. Снимите галочки со всех ШЕСТИ пунктов, промотав список до конца. Вот это всё периферийное оборудование и является виновником сдвига BASIC-области. Оно срабатывает при разных командах и добавляет свои специфические системные переменные. И не меняйте частоту процессора. В общем, для работы по следующим главам самоучителя, настройки должны быть следующими:

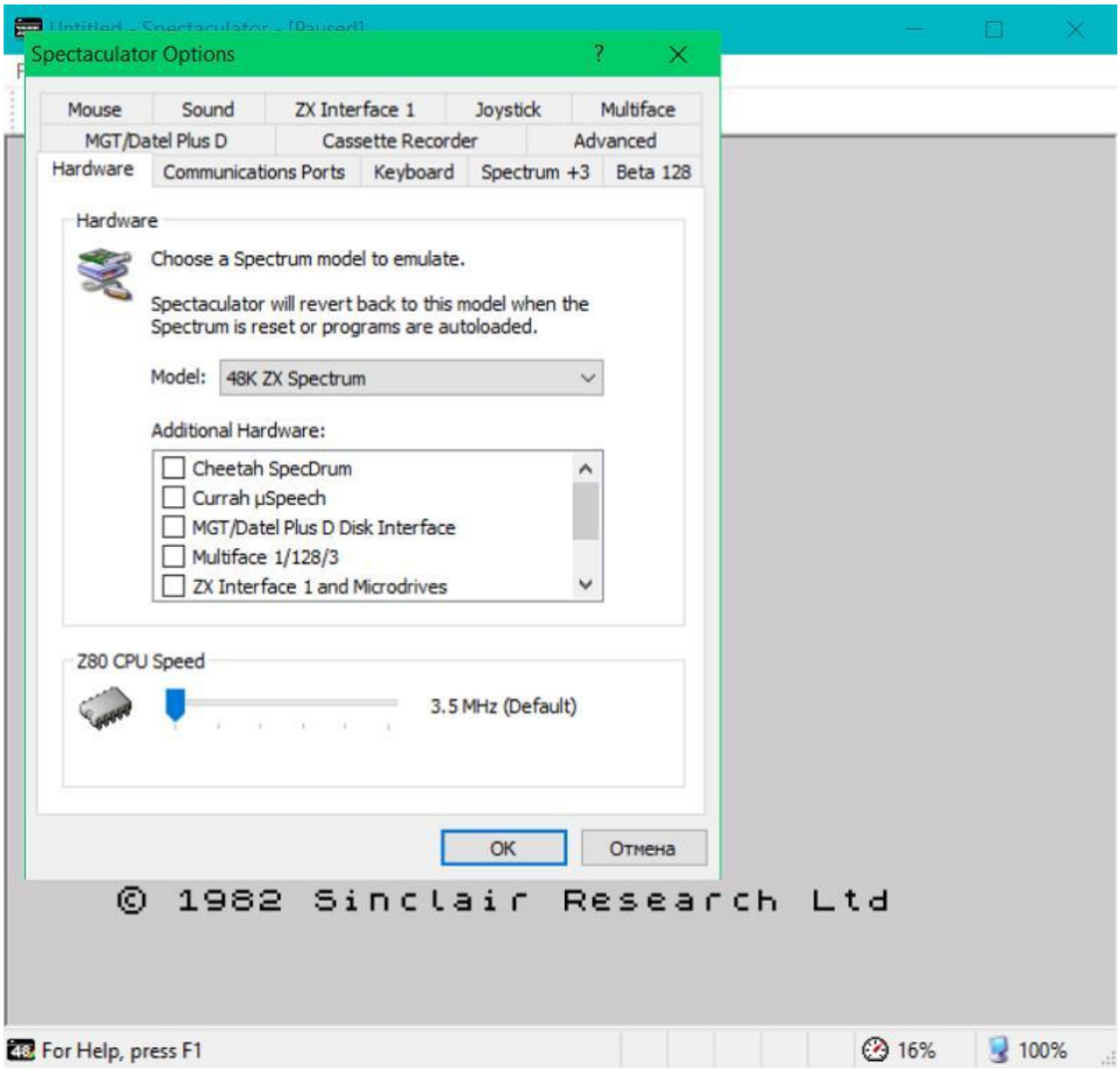


Рис. 23. Spectaculator. Вкладка «Hardware». Настройка периферийных устройств.

Внимательно проверив настройки, нажмите «*OK*». Окно закроется и больше не будет никакого сдвига по команде «`BEEP USR 0`». Теперь точно всё, программа настроена и готова к практике. Чтобы результат ваших программ полностью соответствовал рисункам из книги, можете выставить рамочку размером «*Medium*»:

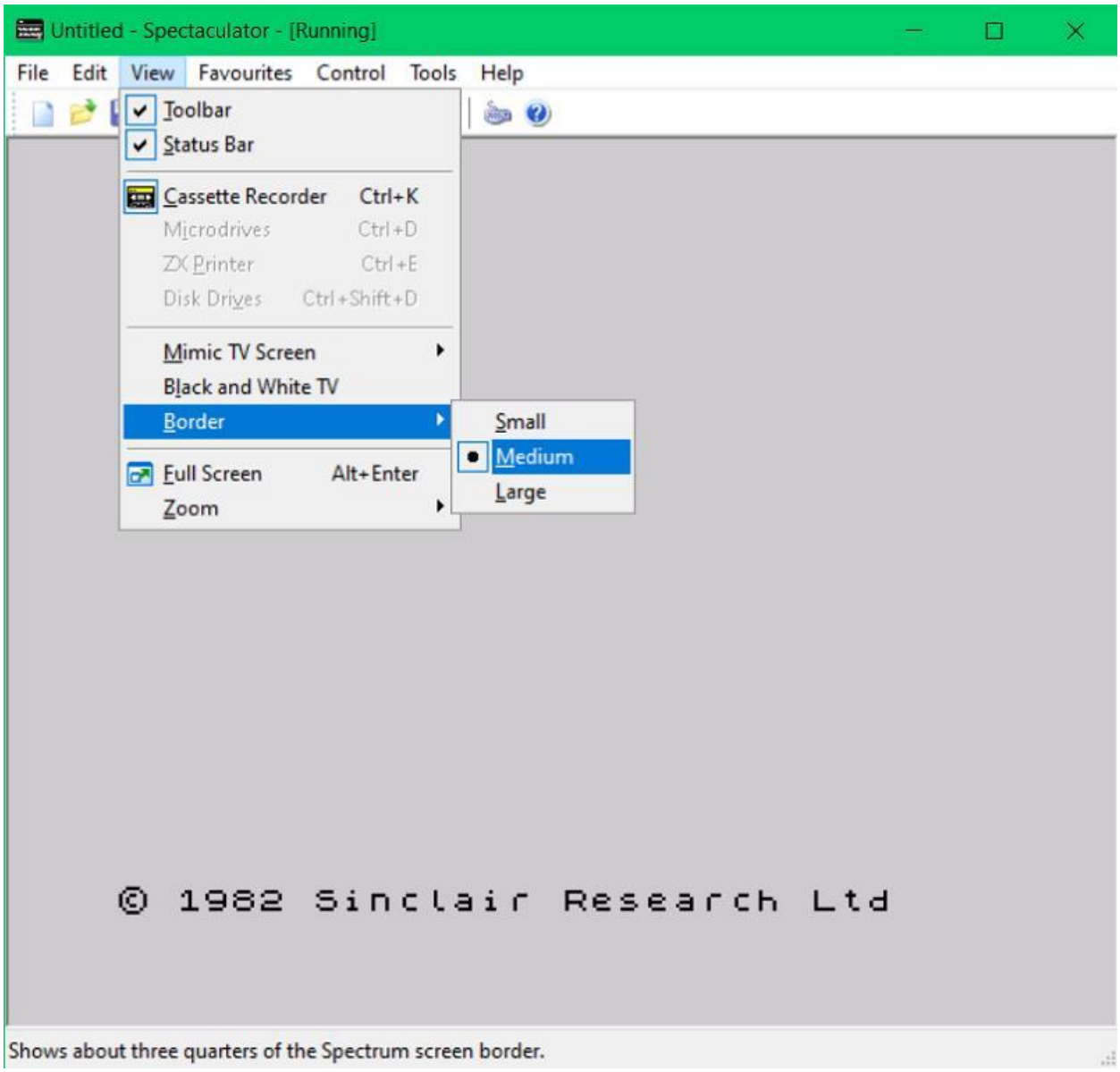


Рис. 24. Spectaculator. Настройка рамки размера «Medium».

И в заключение хочу рассказать о том, почему я выбрал режим 48K. Дело в том, что Spectrum-совместимый компьютер «ДУБНА-48К» у меня появился лишь 7 сентября 1992 года, когда Spectrum «Продолжал триумфальное шествие» по стране. К тому времени уже получила популярность легендарная книга Николая Родионова «ZX-Spectrum для пользователей и программистов» и полтора года, как выходил журнал «ZX-Ревю».

Естественно, я даже не подозревал о существовании этих интересных изданий. Из литературы имелся только легендарный корявый перевод самоучителя по BASIC, в версии завода «ТЕНЗОР» и такая же сомнительная книга от Петербургского издательско-полиграфического комплекса «Вести». В ней, кроме стандартного перевода самоучителя по языку BASIC имелось описание расширений «BETA BASIC», «LASER BASIC» и «MEGA BASIC»:

Рис. 25. Та самая странная книга по BASIC. Фото 4 мая 2017 года.

Даже не имея на руках нормальной литературы, я безумно влюбился в эту модель компьютера. Но счастье длилось недолго. Отец наслушался о «супер компьютерах» с дисководом и уже весной следующего, 1993 года, года отдал «Дубну» на доработку в какую-то сомнительную мастерскую «на несколько дней».

По закону подлости, «Дубна» оказалась несовместимой с дисководом и 128К версией Спектрума, поэтому компьютер завис в мастерской на много месяцев. К осени отец принёс домой «доработанный компьютер». От «Дубны» там осталась только клавиатура. Куча грубо спаянных плат, находилась в большом сером корпусе, из которого виднелся дисковод 5,25’’ со щелью для диска. Телевизионный адаптер от «Дубны» также теперь не подходил, и далее потянулись дни в ожидании телемастера, который должен был припаять к телеку плату.

Настал момент, когда компьютер заработал. Вместо привычной `© 1988 DUBNA`, внизу виднелась `© 1986 Sinclair Research Ltd.` По центру экрана я увидел цветное меню с выбором функций. Оно состояло из пяти пунктов.

Новая модель компьютера мне сразу не понравилась. Во-первых, я не понял, где увидеть эти самые «128К». Во-вторых, через пару недель дисковод перестал записывать, а следом посыпалось и остальное. Все следующие пару лет отец ходил по мастерским, как на работу. Только починят, снова сломается. После третьего похода в мастерскую, дисковод вообще стал работать только на считывание. Пришлось с этим смириться и использовать компьютер только для игр. Единственное, что звук стал трехканальным. Но качество звука, оставляло желать лучшего. К тому же у меня стали появляться кассеты с любимой музыкой. Гораздо приятнее было слушать нормальные песни с магнитофона, чем этот электронно-синтезаторный дребезг из динамика, от которого уставали уши.

К концу 1996 года из магазинов полностью исчезла литература по Spectrum, а к середине февраля 1997 года компьютер окончательно сломался. Эпоха Спектрума ушла, и мастерские уже больше не ремонтировали такие древние компьютеры.

Именно поэтому с компьютером «128К» у меня самые негативные ассоциации. Это нечто постоянно ломающееся, без привычной магнитофонной загрузки, да еще и в BASIC неудобно входить.

Символично, что примерно в тоже время у «Инфоркома», который выпускал журнал «*ZX-Ревю*», весной 1997 года начались финансовые трудности. Кое-как довыпустив остатки номеров на дискетах, он тихо и бесследно исчез, даже не попрощавшись, хотя первый зимний номер начинался с оптимистичных планов.

По прошествии времени, уже в эпоху эмуляторов, я ещё больше разочаровался в режиме «128К». Никакого прямого доступа, например, `RANDOMIZE USR 125000` сделать не получится. По сути это разбивка памяти на куски по 65535, с сопутствующим гемором.

Мое личное мнение: «Spectrum-128» и его клоны, это попытка на спаде общемировой популярности, с помощью костылей создать конкурента IBM-совместимого компьютера. «Дисковод», «RAM диск» и прочие приблуды начисто убили тот непередаваемый шарм, который был у «Spectrum-48» с магнитофоном и прозрачным доступом к памяти. Вообще, если захочется поиграть во все эти свистелки, достаточно просто включить MS-DOS или вернуться в Windows.

Году в 1994-м, когда начался бум на Спектрумы-128 с дисководом, в журнале кто-то критиковал IBM-совместимые за сегментированную память. Риторический вопрос: а «Пентагон'ы» со «Скорпион'ами» то чем лучше? Да точно такое же обращение к памяти по кусочкам без прямой адресации. Так что, я пока предпочитаю вариант «48К» и только его. По крайней мере, в этой книге речь пойдёт о программировании под чистую базовую версию. А дальше, если тема «48К» исчерпает себя, посмотрю по ситуации.

Глава 4
Запуск BASIC. Рождение языка «God Mode»

Краткое содержание: Debugger, основы работы, ввод значений в память, Go To, Set Next Instruction, Trace, язык алгоритмов «God Mode».

– Юрец, ну хоть в этой главе ~~до гаражей доползём~~ «запуск BASIC» начнётся или продолжишь трендеть о прошлой жизни?

– Всё, начинаю, смотрите краткое содержание главы.

Для начала просто откройте *Spectaculator*. В окне программы нажмите на пункт «*Tools*». Откроется меню. Наведите курсор на подпункт «*Debugger*». По подсказке в пункте меню видно, что аналогичных результатов можно добиться, если нажать комбинацию кнопок «*Ctrl+ENTER*»:

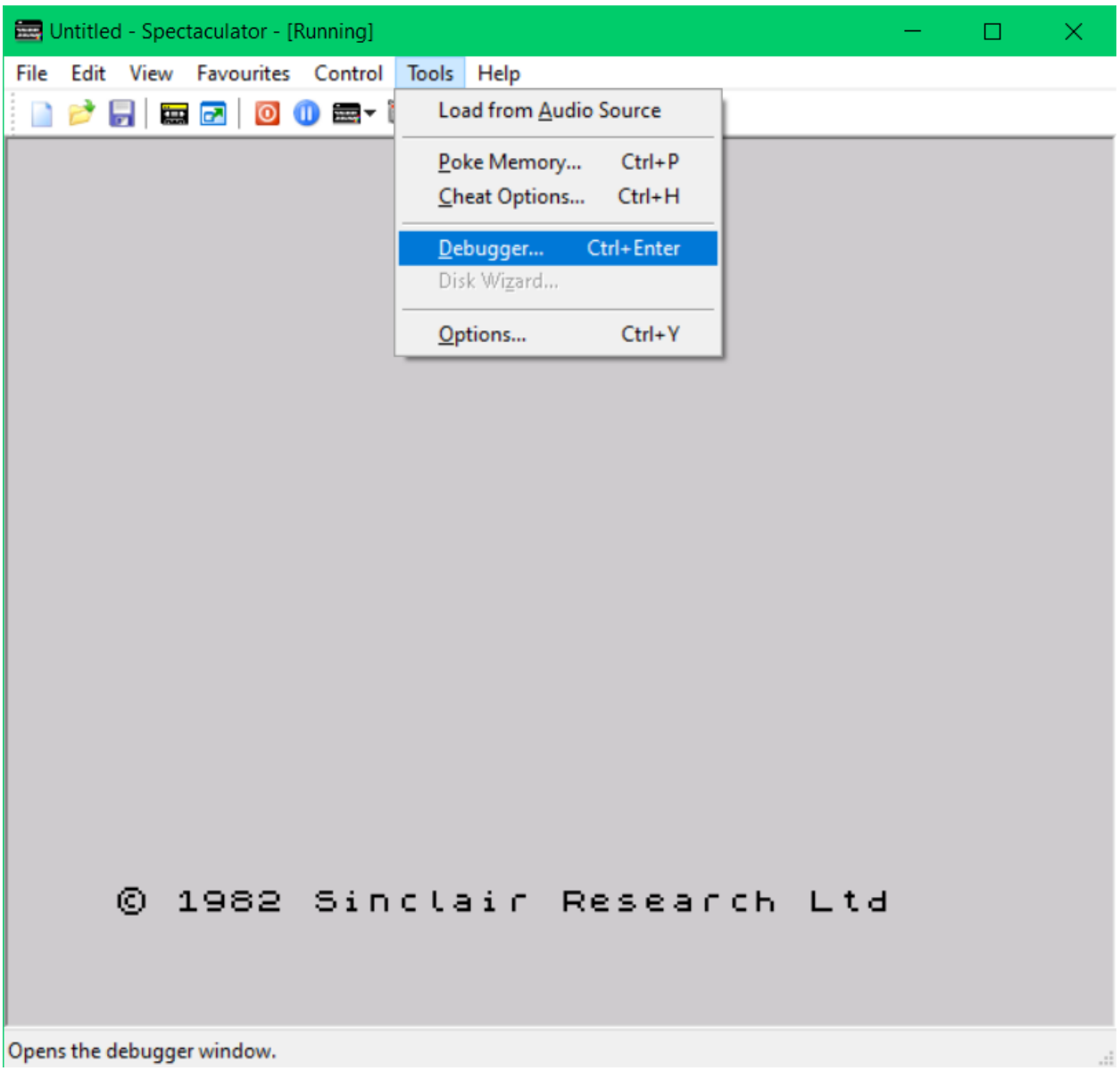


Рис. 26. Spectaculator. Процесс открытия отладчика «Debugger».

В дальнейшем, я буду его открывать именно комбинацией горячих клавиш, а сейчас нажмите кнопку мыши на выделенный пункт. Работа программы приостановится и откроется окошко редактора-отладчика «*Debugger*». Растяните его на экране, чтобы его размер стал комфортным:

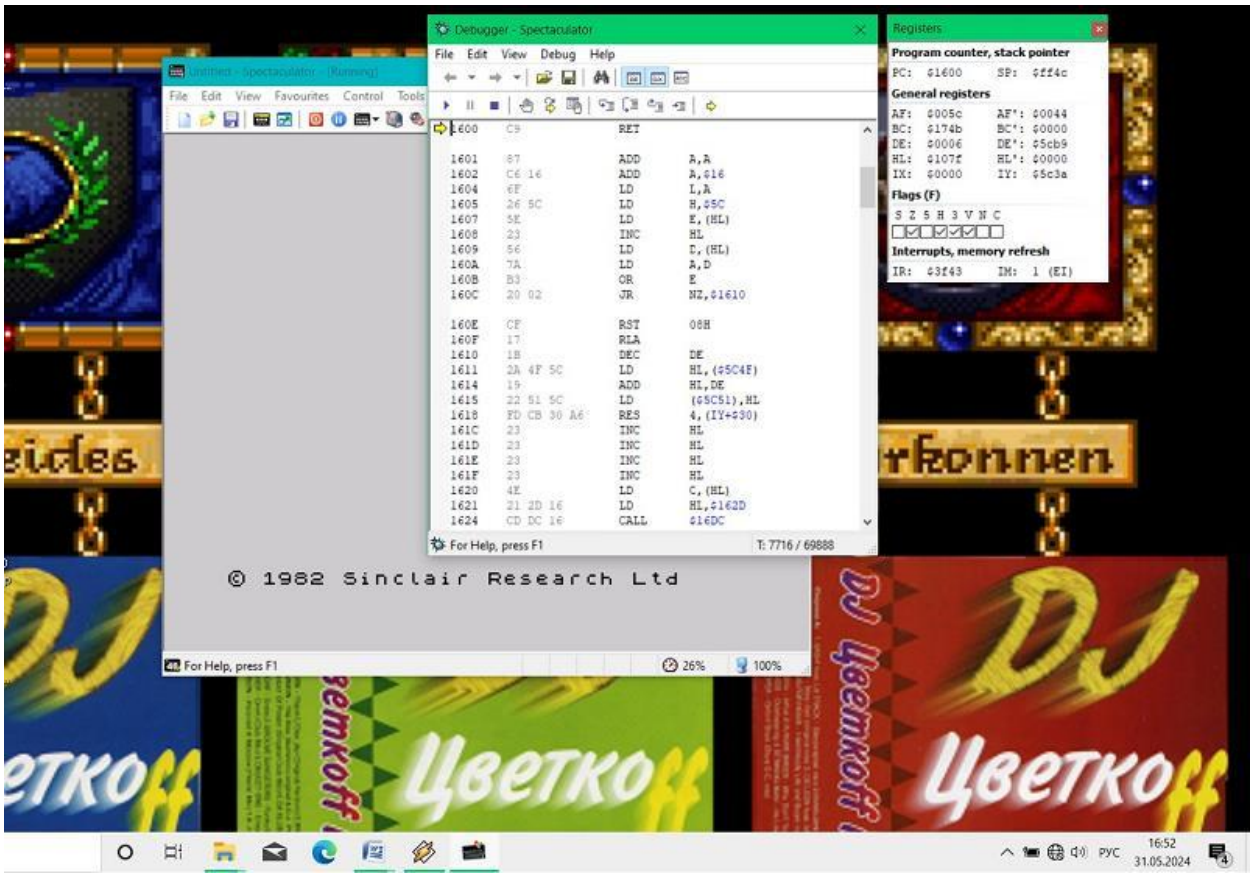


Рис. 27. Отладчик «Debugger» с окном «Registers».

Слева расположен столбец с адресами памяти. Та самая Жёлтая ➪ Стрелочка из вводной главы показывает текущий адрес, в котором выполнялась команда, до момента открытия этого окна. У каждого текущая позиция стрелочки будет произвольной и в разном месте.

Следом стоят группы чисел с содержимым этих ячеек. Левая колонка текста является переводом значений машинного кода в ассемблер. В верхнем углу приютилось окошечко с текущими значениями регистров «*Registers*». Всё вроде просто, кроме одного: все числа в шестнадцатеричных значениях. Если вам, как и мне, комфортнее десятичный вид, просто нажмите «*Ctrl+H*» или отожмите кнопочку «*HEX*» во втором ряду окошка:

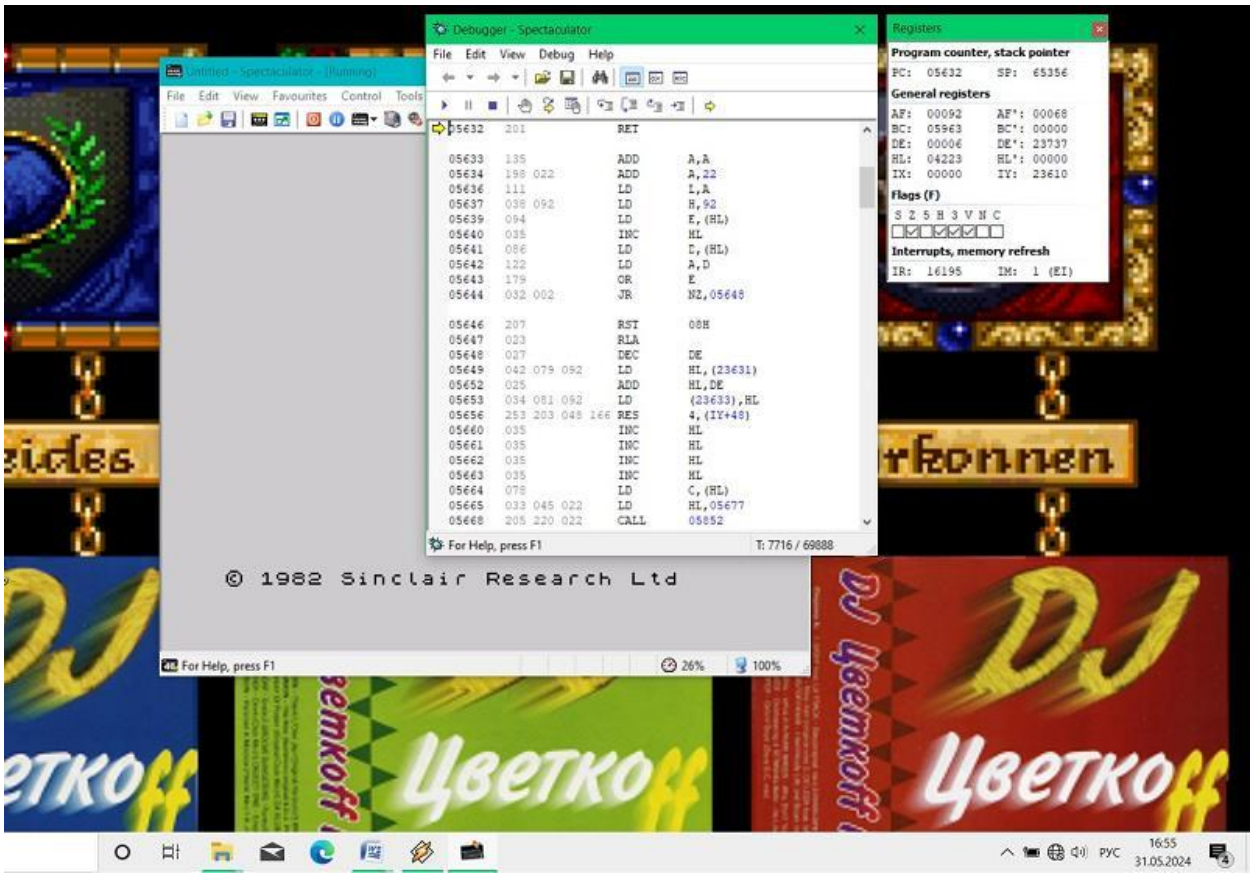


Рис. 28. Переключение отображения чисел в десятичном виде.

Числа моментально трансформировались в привычный вид. Запомнить легко: «HEXер тут числа всякими буквами изображать».

А теперь вспомните сказку о приключениях Жёлтой ➪ Стрелочки. Предлагаю самостоятельно перезапустить BASIC с самого нуля. А сделать это очень легко. Достаточно просто переставить Стрелочку в начало своего пути, на адрес 0.

Отладчик достаточно гибкая программа, поэтому сделать это можно несколькими способами. Самым простым будет изменение значения «PC» в окошке «*Registers*» с текущего на нулевое.

Наведите курсор на окошечко со значением…. в моём текущем случае «05632», в вашем какое-то иное, и нажмите правую кнопку мышки:

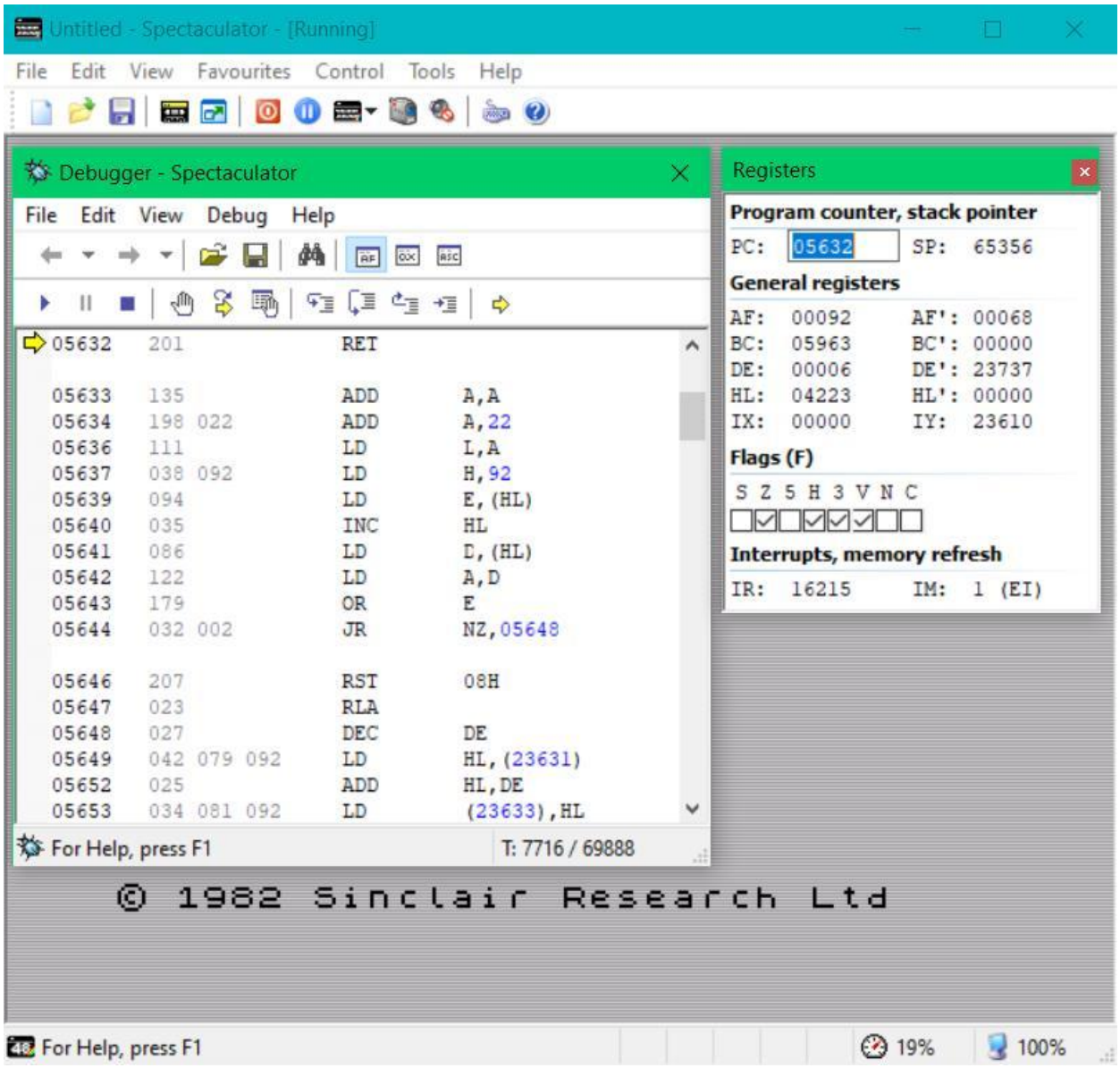


Рис. 29. Debugger. Редактирование значения в окошке «PC».

Значения в строке посинели. А теперь начинайте вводить число «0»:

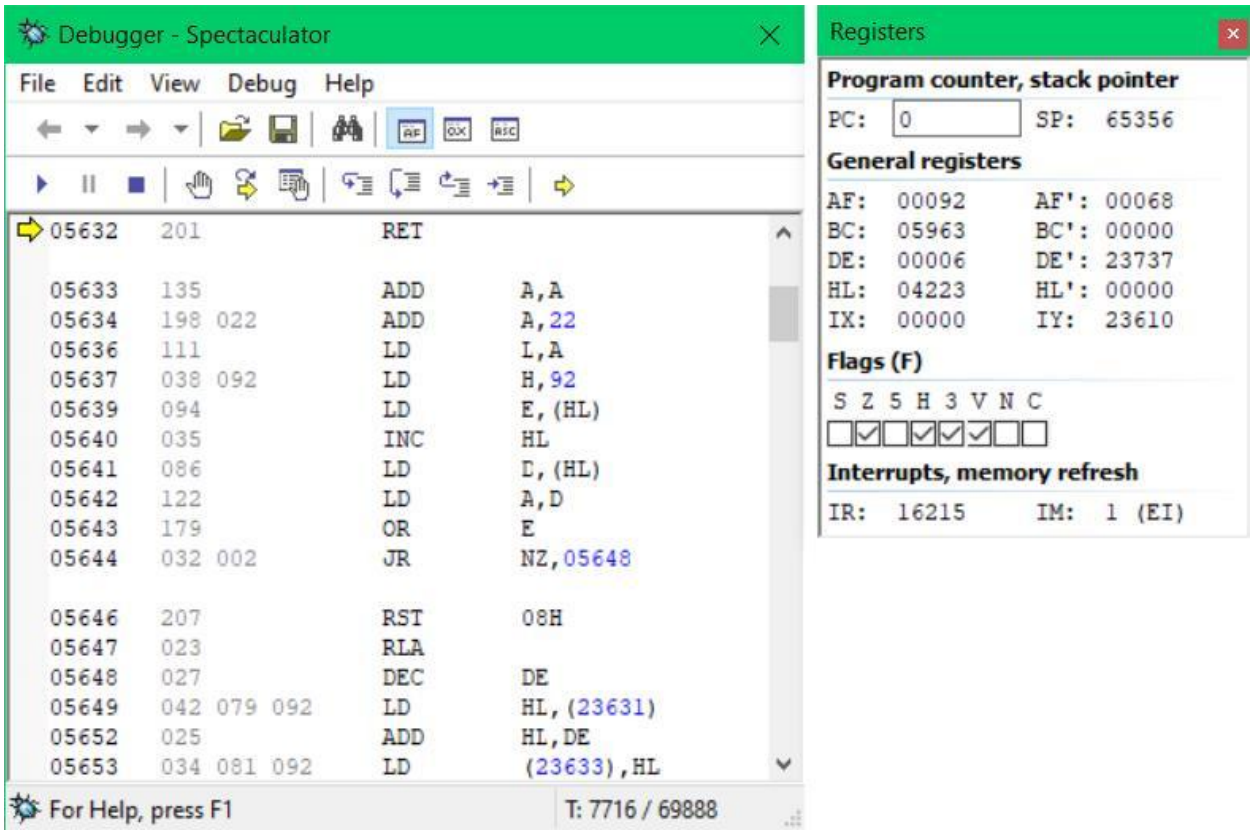


Рис. 30. Ввод нового значения в окошко «PC».

Старое выделенное число автоматом стёрлось и на его месте появилось напечатанное вами. Нажимайте ENTER:

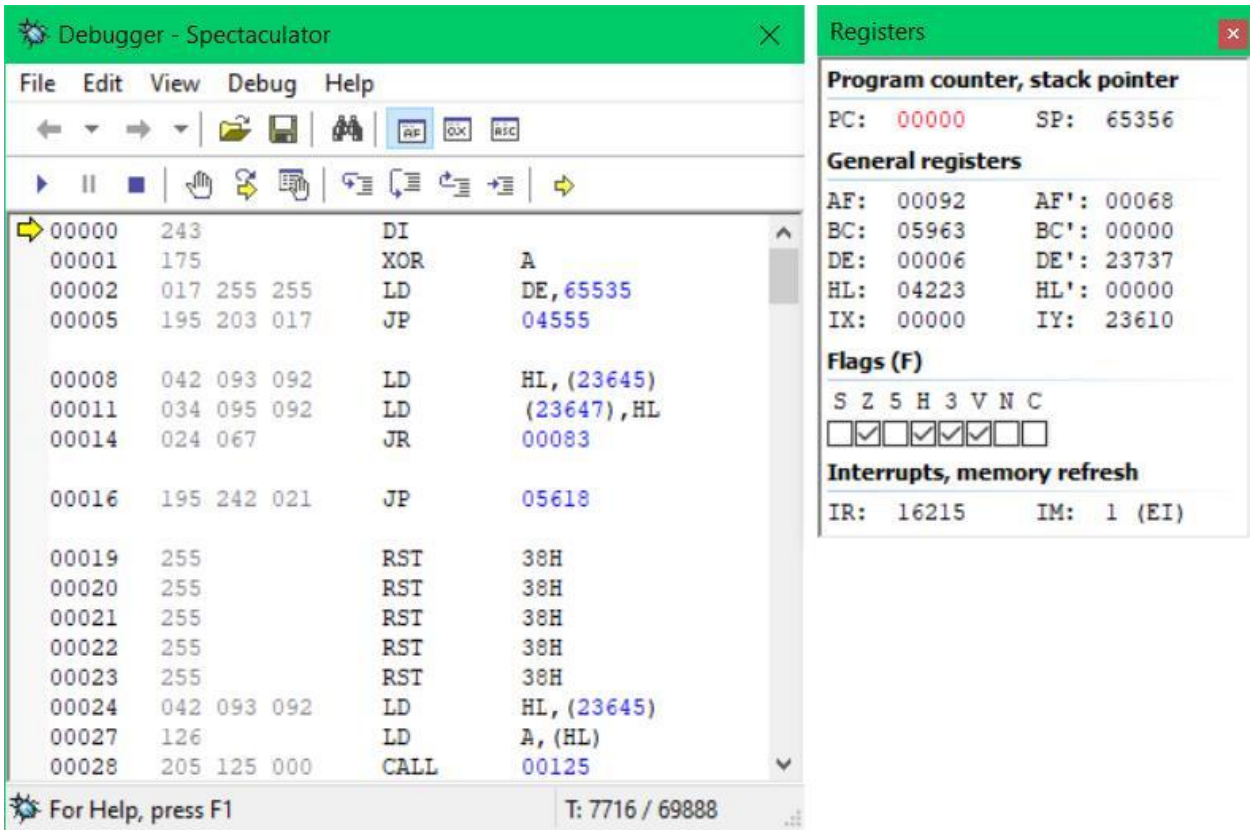


Рис. 31.Debugger. Установка стрелочки по адресу «0».

Новое значение «PC» принялось и покраснело. А в это время в соседнем большом окошке Стрелочка моментально перенеслась в начало своего пути по адресу «0».

Теперь осталось запустить программу и вывести результат работы. В левом верхнем углу найдите кнопочку «*Trace (F5)*» в виде синего ▸ треугольничка:

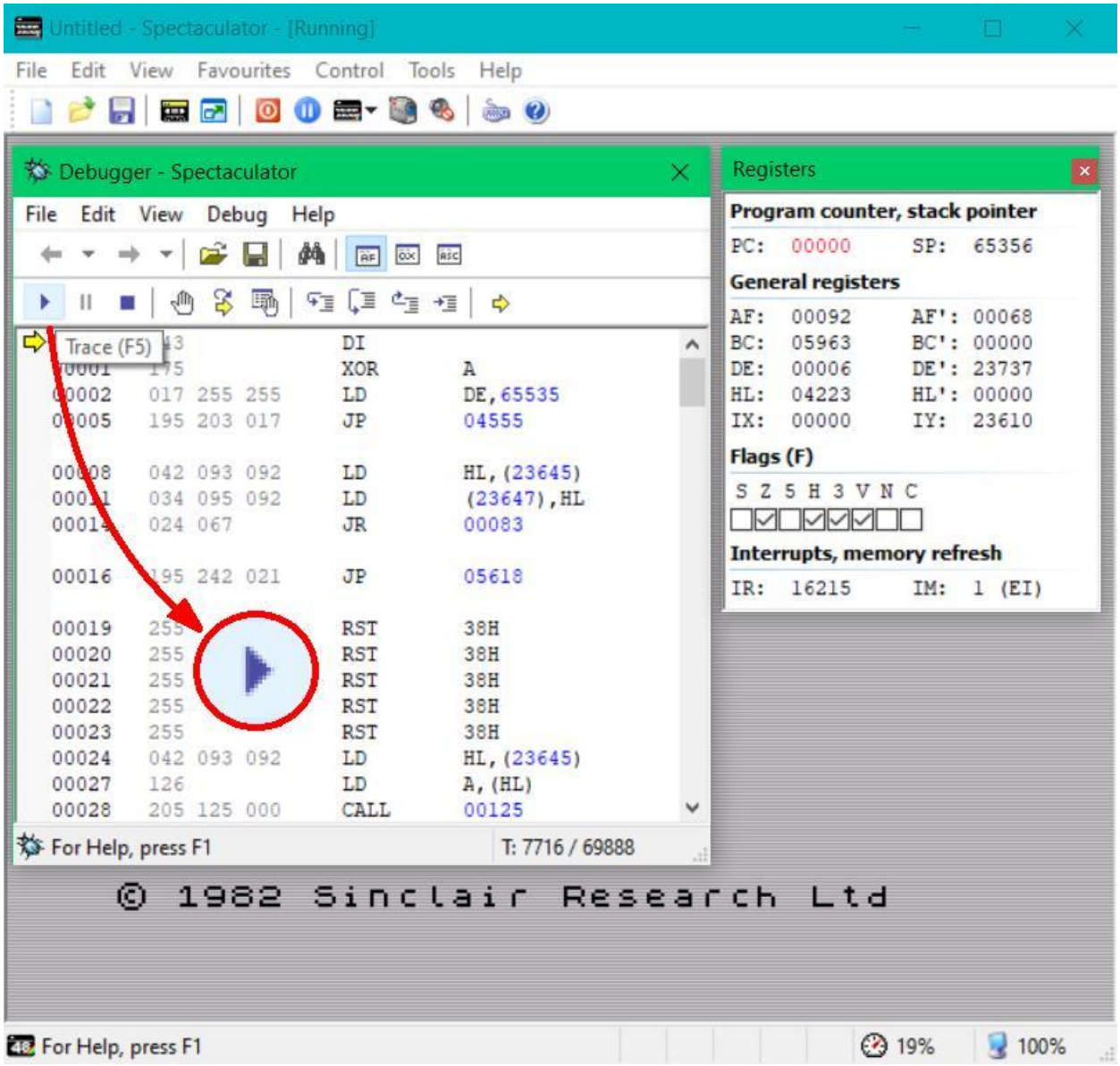


Рис. 32. Debugger. Кнопочка «Trace» для запуска программы.

Нажимайте её, и… Стрелочка повторно отправилась в увлекательное путешествие запускать BASIC-систему:

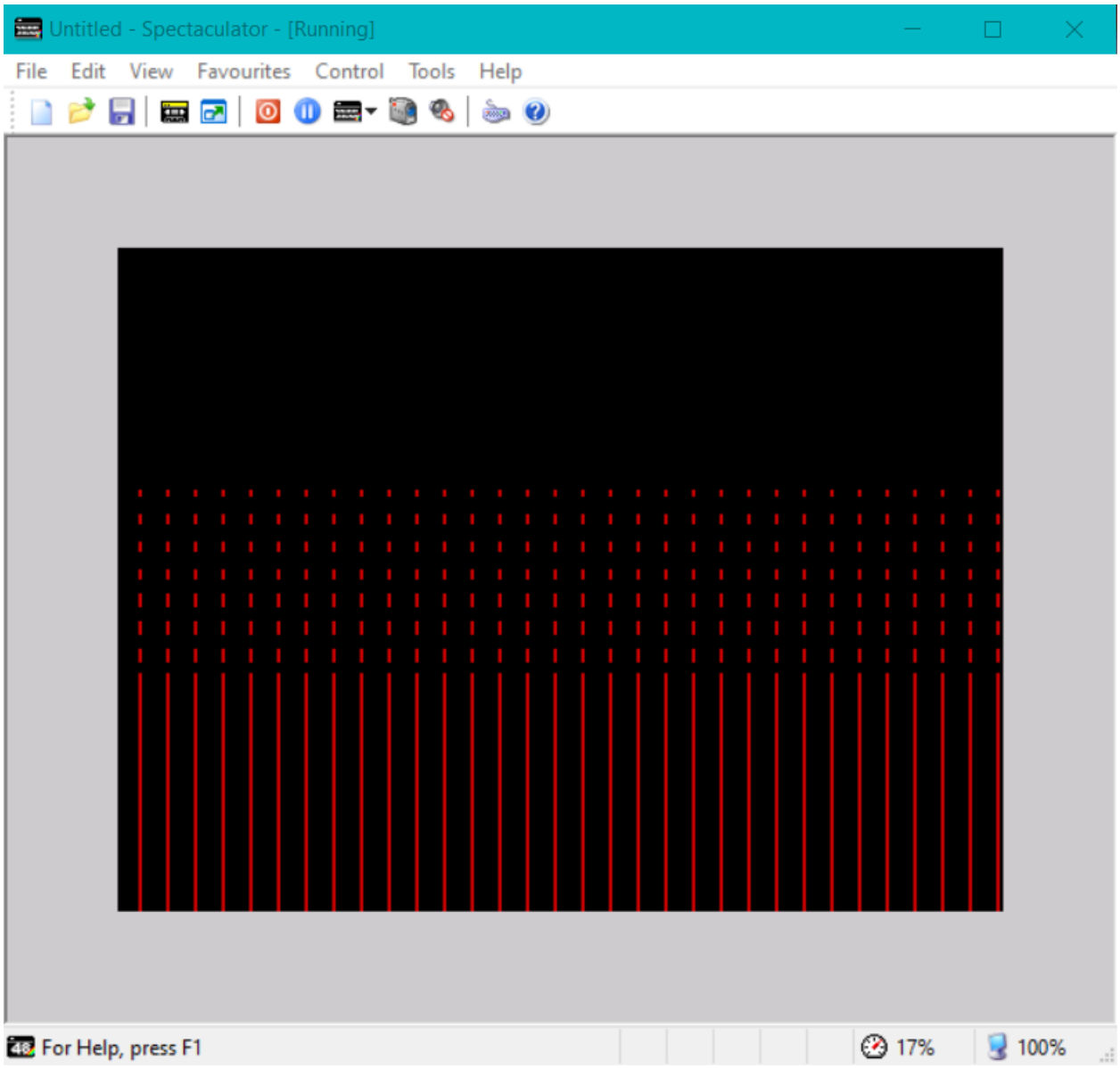


Рис. 33. Spectaculator. Сброс.

На первый взгляд, произошёл обычный сброс. На ассемблере данное действие будет эквивалентно «`RST`, `JP` или `CALL 0`», а на BASIC – «`ЧТО-НИБУДЬ USR 0`».

А теперь посмотрите на это событие под другим углом.

Вы только что произвели некое действие, которое повлекло за собой другое действие. Иначе говоря, вы создали и запустили свою первую короткую программу извне. Любая программа пишется на каком-либо машинном языке, значит, в эту минуту на этой странице происходит историческое событие – рождается новый и неведомый язык алгоритмов, который пока не имеет своего названия. Как назвать появляющийся в прямом эфире язык программирования? Давайте он будет называться «*God Mode*» или алгоритмический язык «Режима Бога».

Невольно вы открыли первые четыре команды нового языка программирования:

«*Debugger*» – открывает окно отладчика (*Ctrl+ENTER*), подготовив среду программирования для ввода данных. Пусть пока будет что-то вроде «ORG» на ассемблере.

«*Dec*» – настройка отображения чисел в десятичный вид (*Ctrl+H*).

«*PC ← X*» – установка курсора-стрелочки на адрес X, где X=0…65535

«*Trace*» – запуск набранной программы путём выхода в программу клавишей F5 или кнопочкой ▸ .

Предлагаю написать эту программу на новом языке программирования:

```
Debugger
Dec
PC ← 0
Trace
```

Как я уже говорил, отладчик довольно гибкий, поэтому одно и то же действие можно выполнить разными методами. Вот еще один способ. Войдите в «*Debugger*», как описывалось в прошлом примере. Теперь нажмите кнопку «*Edit*». В выпавшем меню наведите курсор на пункт «*Go To*». Сбоку выскочит подменю, последним пунктом которого является «*Address (Ctrl+G)*»:

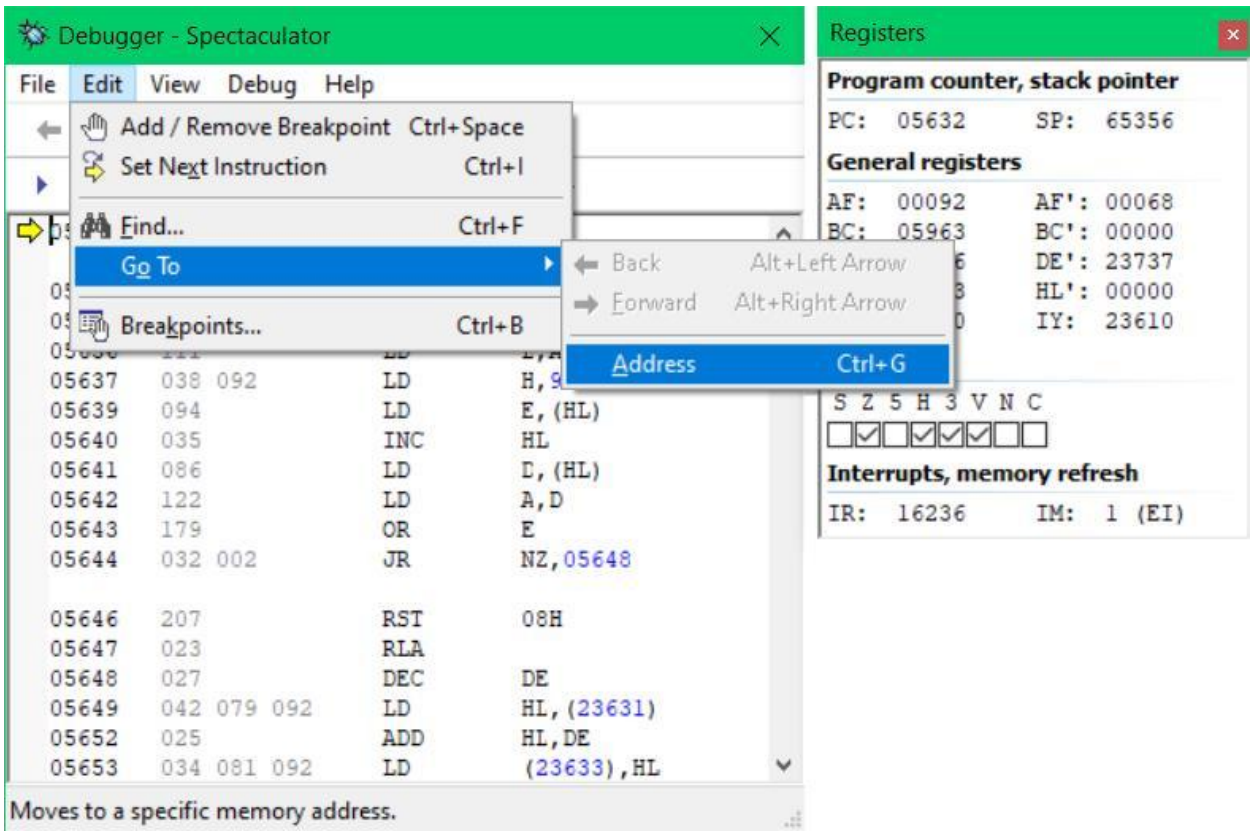


Рис. 34. Debugger. Процесс открытия окошка «Go To» через пункты меню.

Нажмите на него правой кнопкой мышки. Откроется маленькое окошко «*Go To*». В белую строчку под надписью «*Address*» впишите «0»:

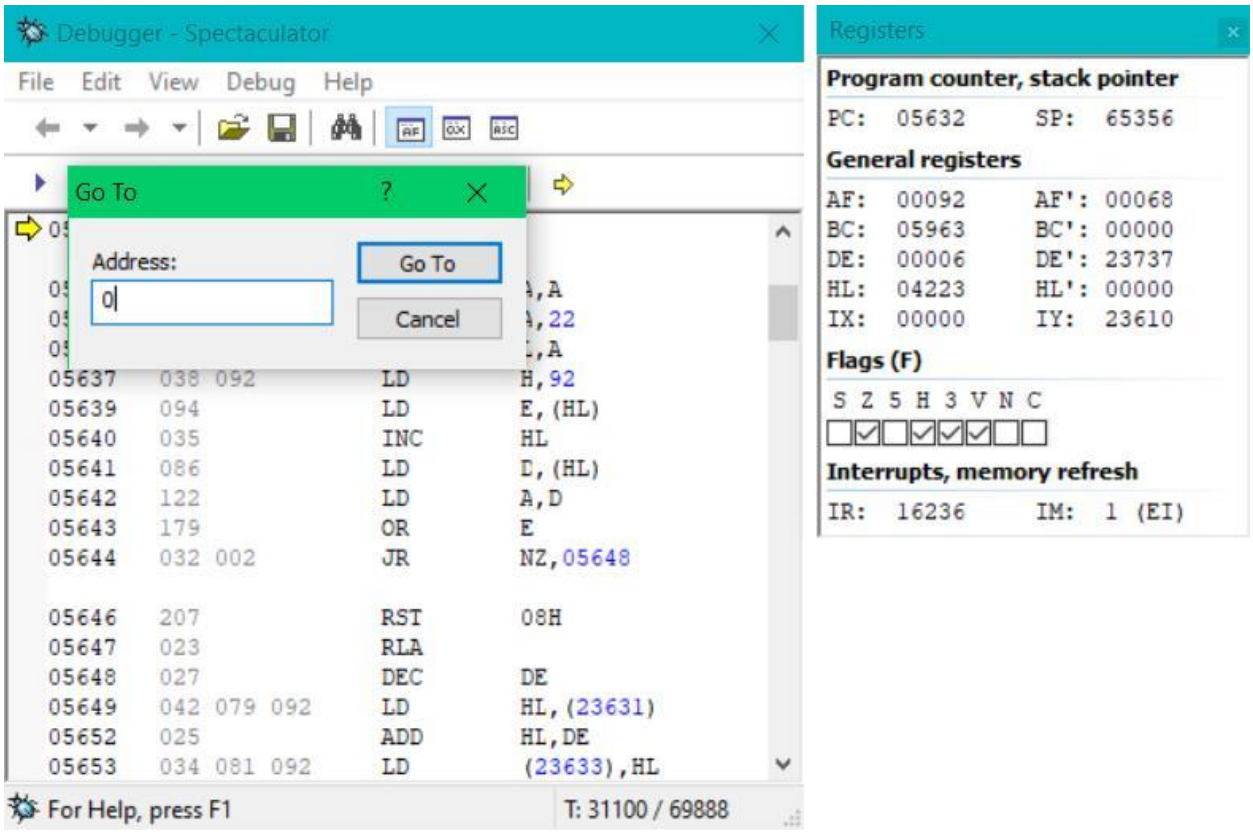


Рис 35. Окошко «Go To». Вписывание адреса для быстрого перемещения.

После нажатия кнопки «*Go To*» или клавиши «*ENTER*» вы мгновенно перенесётесь на требуемый адрес. Курсор «палочка» будет мигать перед нужным значением:

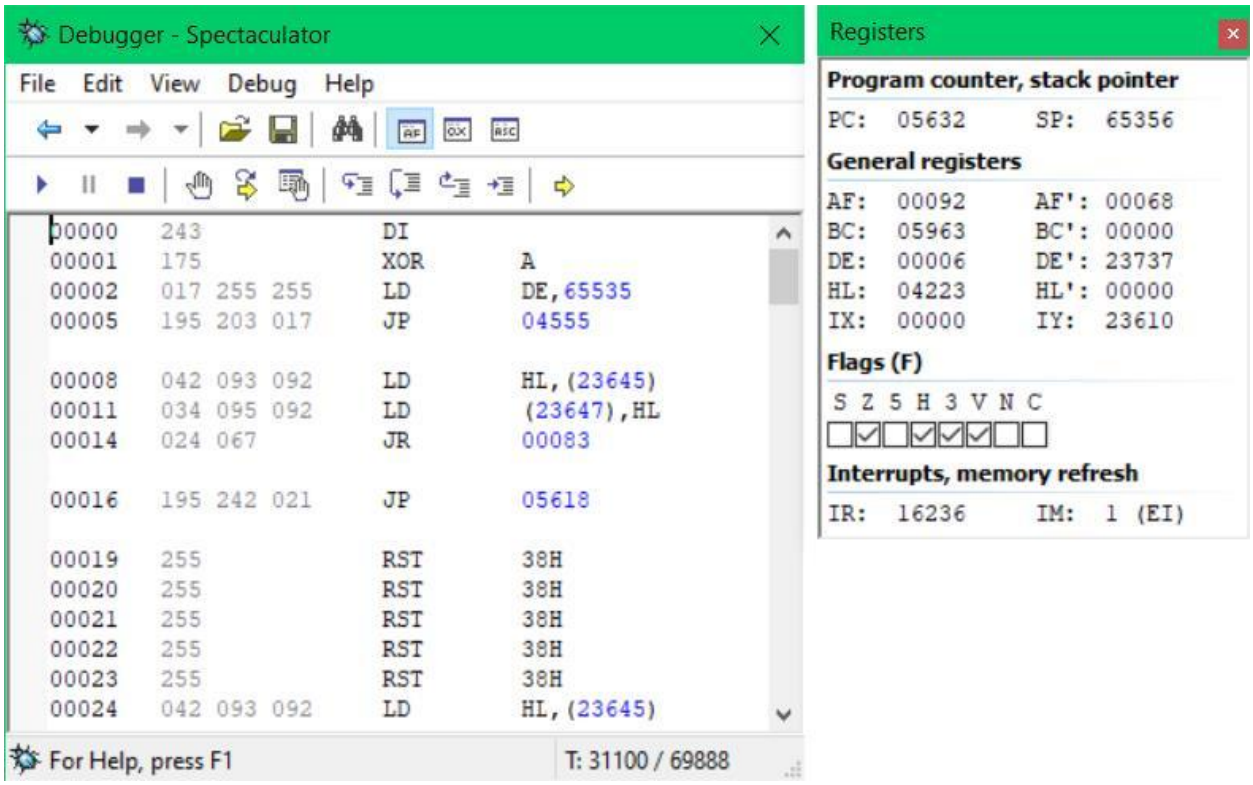


Рис. 36. Результат перемещения по нужному адресу без установки Стрелочки.

Обратите внимание, что Стрелочка при этом осталась в том месте, где стояла, по указанным в «PC» координатам. Такой способ удобен для передвижения по адресам с целью просмотра нужных областей.

Сразу над адресным полем найдите кнопку  «Синяя стрелочка над жёлтой», которая называется «*Set Next Instruction (Ctrl+I)*»:

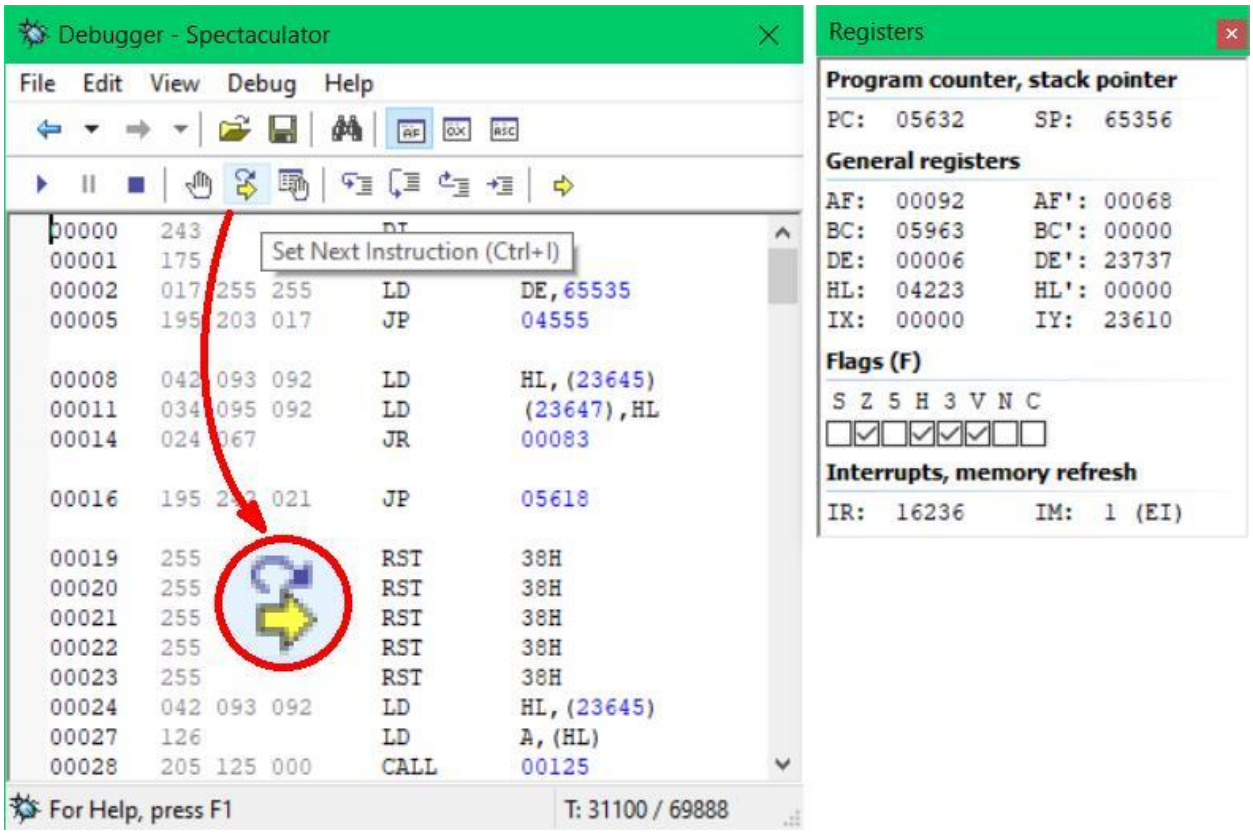


Рис 37. Debugger. Кнопка Set Next Instruction.

Нажав на неё, Желтая Стрелочка появится перед выбранным адресом.

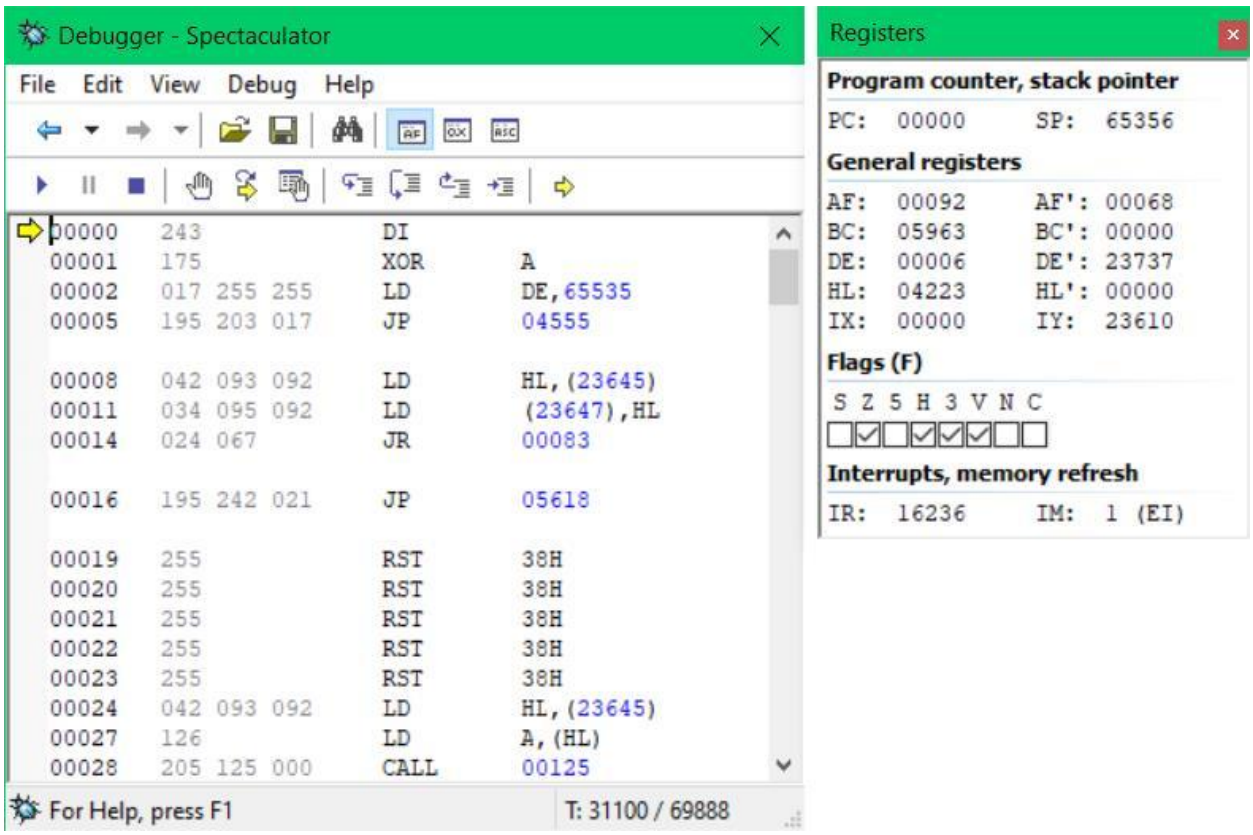


Рис. 38. Результат установки Стрелочки на адрес «0» средствами меню.

Осталось только нажать «*Trace*» или клавишу «*F5*» и Стрелочка пойдёт выполнять задание. Кроме этого имеется еще один способ посадить Стрелочку в нужное место. Просто наведите курсор на нужный адрес и нажмите ЛЕВУЮ кнопку мыши. Откроется такое меню, где последним пунктом «*Set Next Instruction (Ctrl+I)*» предлагается переставить Стрелочку на текущую позицию курсора мыши:

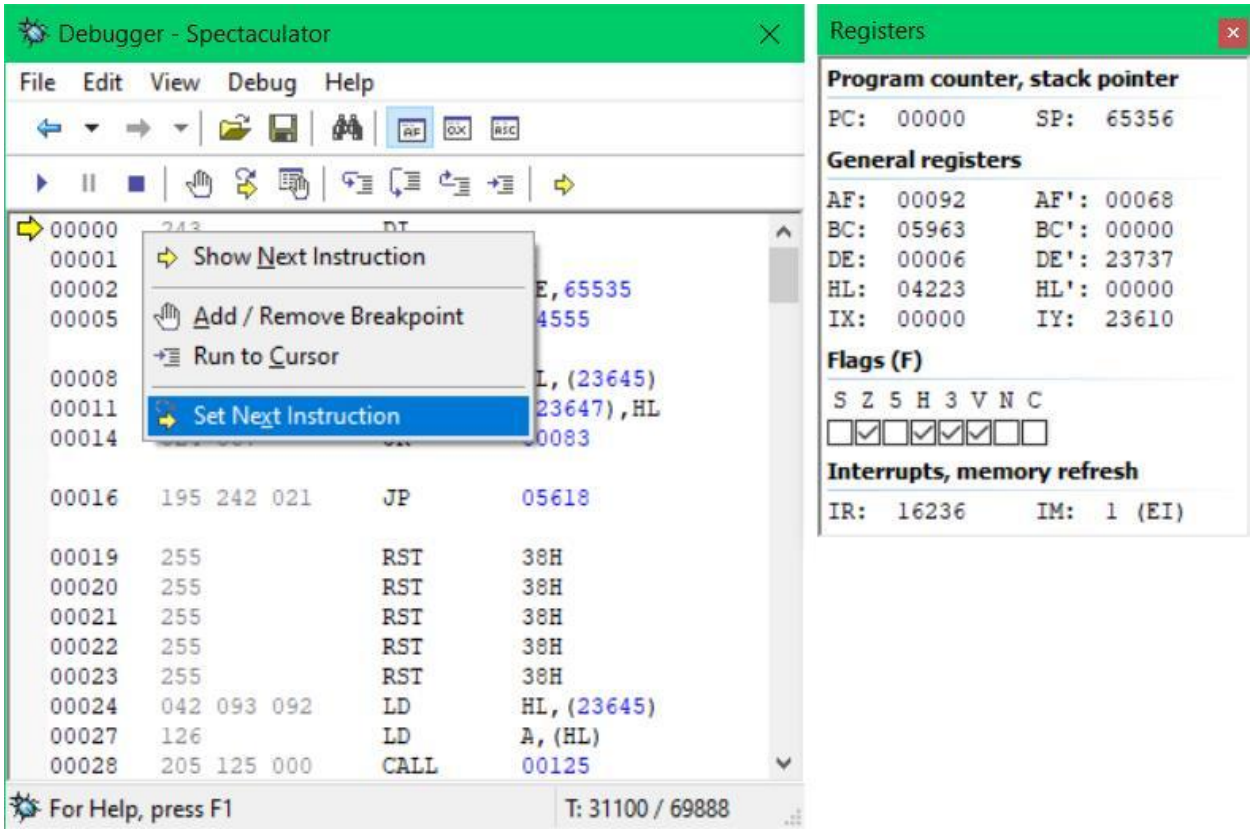


Рис. 39. Еще один вариант установки Стрелочки.

Выделив его, нажмите ЛЮБУЮ кнопку мыши и получите идентичный результат. Вариантов целых четыре штуки, на любой вкус и цвет. В этом плане, отладчик очень хороший.

Ну а теперь нажмите «*Trace (F5)*» ▸ и отпустите Стрелочку в её любимый сказочный город BASIC. Пока она возвращается, предлагаю придумать для вариантов программы «сброс» алгоритмы и выбрать удачные идеи:

Debugger
Dec
Go To 0
Set Next Instruction 0
Trace

Или эквивалент манипуляции с клавиатурой;

Ctrl+ENTER
Ctrl+H
Ctrl+G → 0
Ctrl+I
F5

На мой взгляд, из всего этого целесообразнее команду перемещения Стрелочки оставить «*PC ← X*». Тем более, что аббревиатуру «PC» можно трактовать как Расположение Стрелочки. А вот мнемоника «*Go To X*» достаточно интересная и можно

взять в копилочку команд. Все удачные получающиеся команды будут представлены в приложении к самоучителю.

Подводя итоги главы, что можно еще добавить к вышесказанному? Вариант с изменением значения регистра «PC» не только самый удобный и лаконичный в обозначении, но и самый простой в плане манипуляций руками. Он полезен в том случае, когда вы точно знаете, на какой адрес нужно поставить Стрелочку. Остальные варианты пригодятся, когда требуется просмотреть фрагменты программы, и только потом принять решение, о её перемещении.

Глава 5
Тайны подземелья замка EDITOR

Королева Жёлтая ⇨ Стрелочка вернулась в свой город BASIC и зашла через запасные ворота. Шагая по главной улице города, она наслаждалась пейзажами и радовалась жизни. На всякий случай, сжав и оптимизировав все вспомогательные BASIC области программой `SET MIN (5808)`, она установила вывод печати в нижние строки (`CHAN OPEN`). А тем временем, в городе царила мёртвая тишина. Пятый бит `TV_FLAG (23612)` давал о себе знать.

В центре главной городской площади возвышался королевский замок под названием `EDITOR (3884)`. Внутри он имел разветвлённую сеть таинственных подземных лабиринтов. О замке ходило много слухов и легенд, но только хозяйка знала каждый уголок своего огромного дома. Настал момент, когда Стрелочка подошла к дверям своего таинственного замка…

```
04768   024 007          JR       04777

04770   253 054 049 002  LD       (IY+49),02
04774   205 149 023      CALL     06037
04777   205 176 022      CALL     05808
04780   062 000          LD       A,00
04782   205 001 022      CALL     05633
⇨04785  205 044 015      CALL     03884
04788   205 023 027      CALL     06935
04791   253 203 000 126  BIT      7,(IY+00)
04795   032 018          JR       NZ,04815

04797   253 203 048 102  BIT      4,(IY+48)
04801   040 064          JR       Z,04867

04803   042 089 092      LD       HL,(23641)
04806   205 167 017      CALL     04519
04809   253 054 000 255  LD       (IY+00),255
04813   024 221          JR       04780
```

Рис. 40. Стрелочка у входа в подземелье EDITOR.

Королева открыла дверь. Изнутри повеяло приятной прохладой. Стрелочка шагнула в темноту и растворилась в неизвестности.

Процессорные года складывались в столетия и плавно переходили в тысячелетия, а Стрелочка всё не возвращалась. Хотите узнать тайны волшебного замка `EDITOR`, а быть может предложить Стрелочке выйти на прогулку, чтобы она не скучала в своих лабиринтах?

Придётся зайти на чердак и перетрясти старые чемоданы и сундуки. А вдруг найдётся старая карта замка или план подземелья? После долгих поисков, нашлась

маленькая пожелтевшая бумажка, на которой обнаружилось всего одно слово «Клавиатура».

Печально. Придётся отправляться в замок вслед за Стрелочкой по команде `CALL 3884` и самостоятельно изучать его разветвлённую сеть. Одна подсказка есть и это уже неплохо. Нужно больше узнать о клавиатуре. Ведь таинственная программа называется `EDITOR` и завязана на работе с клавишами…

Итак, работа клавиатуры состоит из двух изолированных кусков.

Первый – это комплекс подпрограмм-производителей по адресам 00604-00783. Вызываются они по режиму прерывания `IM 1` и опрашивают клавиатуру. Считывая комбинацию клавиш, они переводят полученный результат в символы с кодами ASCII от 0 до 255. Записав готовый продукт в ячейку `LAST_K (23560)`, подпрограмма `K-END (00775)` включает сигнальный тумблер, роль которого играет 5-й бит ячейки `FLAGS (23611)`. Этот тумблер сигнализирует программам потребителям, что нажата свеженькая клавиша, и можно её использовать в своих целях. Записав этот готовый продукт, происходит выход из изолированной программы обратно в мир BASIC:

```
00775   119             LD        (HL),A
00776   050 008 092     LD        (23560),A
00779   253 203 001 238 SET       5,(IY+01)
00783   201             RET
```

Рис. 41 Фрагмент программы K-END. Процесс записи готового ASCII кода.

Собственно, эти несколько строк по адресу 00775 и будут важным итогом работы всей этой мощной фабрики по переработке нажатых клавиш.

Вторая группа это программы потребители. Они забирают готовую продукцию и выключают бит-тумблер в положение «0», показав, что «цифровой» товар забрали и используют по назначению.

Нырнув в программу `EDITOR (03884)` из главного цикла BASIC, начинается спуск по ступенькам многослойных вложений. Добравшись до глубины программы `WAIT KEY (5588)` проверяется тумблер №5 ячейки `TV_FLAG (23612)`. Если он отключен, то происходит локальное обновление нижней строки. Если он включен, значит не нужно обновлять содержимое нижней строки и Стрелочка проваливается дальше:

```
05588   253 203 002 110 BIT       5,(IY+02)
05592   032 004         JR        NZ,05598

05594   253 203 002 222 SET       3,(IY+02)
05598   205 230 021     CALL      05606
05601   216             RET       C

05602   040 250         JR        Z,05598
```

Рис. 42. Фрагмент программы WAIT KEY. Первый барьер.

Проскользнув ещё несколько ступенек ниже, она опускается на самое дно, в программу `KEYBOARD INPUT (04264)`. По адресу 02476 стоит непробиваемая стена в виде команды `RET Z`:

```
04264   253 203 002 094 BIT       3,(IY+02)
04268   196 029 017     CALL      NZ,04381
04271   167             AND       A
04272   253 203 001 110 BIT       5,(IY+01)
04276   200             RET       Z

04277   058 008 092     LD        A,(23560)
04280   253 203 001 174 RES       5,(IY+01)
```

Рис. 43. Фрагмент программы KEYBOARD INPUT. Непреодолимый барьер.

Без включения бита-тумблера №5 в FLAGS (23611) или нажатой клавиши, которая его и устанавливает, Стрелочке дальше не пройти. Уткнувшись в условие, она взмывает наверх и происходит закольцовка с возвратом в WAIT KEY (5588). До нажатия клавиши, выше яруса WAIT KEY и далее барьера в KEYBOARD INPUT (04276), Стрелочка не уходит. В это время на экране царит безмолвие, которое вы хорошо знаете.

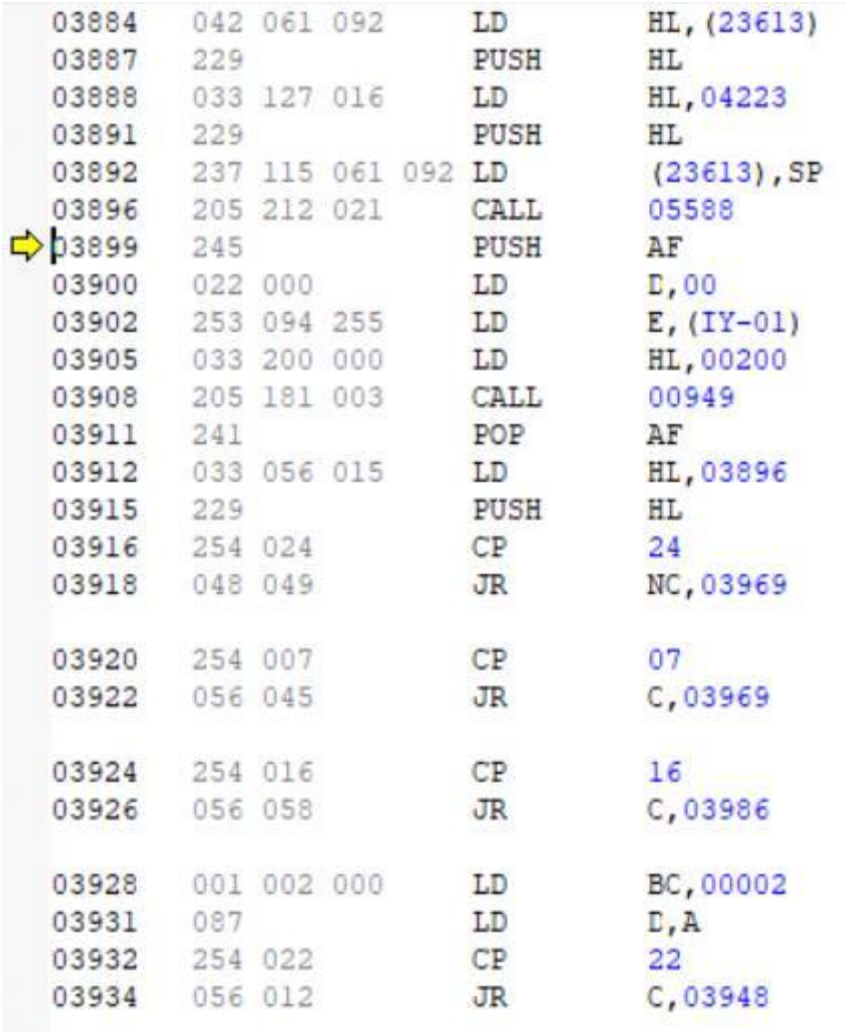
По нажатию любой клавиши происходит пробой защиты, и Стрелочка попадает на следующий адрес после преграды:

```
04273   203 001         RLC       C
04275   110             LD        L,(HL)
04276   200             RET       Z

04277   058 008 092     LD        A,(23560)
04280   253 203 001 174 RES       5,(IY+01)
04284   245             PUSH      AF
04285   253 203 002 110 BIT       5,(IY+02)
04289   196 110 013     CALL      NZ,03438
04292   241             POP       AF
04293   254 032         CP        32
04295   048 082         JR        NC,04379

04297   254 016         CP        16
04299   048 045         JR        NC,04346

04301   254 006         CP        06
04303   048 010         JR        NC,04315

04305   071             LD        B,A
04306   230 001         AND       01
04308   079             LD        C,A
04309   120             LD        A,B
04310   031             RRA
04311   198 018         ADD       A,18
04313   024 042         JR        04357

04315   032 009         JR        NZ,04326

04317   033 106 092     LD        HL,23658
04320   062 008         LD        A,08
04322   174             XOR       (HL)
04323   119             LD        (HL),A
04324   024 014         JR        04340

04326   254 014         CP        14
04328   216             RET       C
```

Рис.44. Первые шаги Стрелочки после нажатия комбинации клавиш.

Первым делом идёт разделение на управляющие символы и буквенно-командные. Если был нажат управляющий символ с курсорами, то формирование действий происходит в этой же подпрограмме.

Для обработки символов с кодом свыше пробела (32) происходит подъём обратно на верхние этажи в программу EDITOR, к отметке 3899. При этом в регистре «A» продолжает лежать ASCII-код нажатой клавиши:



```
03884   042 061 092     LD      HL,(23613)
03887   229             PUSH    HL
03888   033 127 016     LD      HL,04223
03891   229             PUSH    HL
03892   237 115 061 092 LD      (23613),SP
03896   205 212 021     CALL    05588
03899   245             PUSH    AF
03900   022 000         LD      D,00
03902   253 094 255     LD      E,(IY-01)
03905   033 200 000     LD      HL,00200
03908   205 181 003     CALL    00949
03911   241             POP     AF
03912   033 056 015     LD      HL,03896
03915   229             PUSH    HL
03916   254 024         CP      24
03918   048 049         JR      NC,03969

03920   254 007         CP      07
03922   056 045         JR      C,03969

03924   254 016         CP      16
03926   056 058         JR      C,03986

03928   001 002 000     LD      BC,00002
03931   087             LD      D,A
03932   254 022         CP      22
03934   056 012         JR      C,03948
```

Рис. 45. Программа EDITOR. Возврат из подземелья KEYBOARD INPUT с ASCII-кодом нажатой клавиши.

Озвучив нажатие буквенно-числовой клавиши щелчком из `PIP (23609)`, код символа проходит многоступенчатую сортировку с фильтрами, чтобы в конечном итоге вывести на экран соответствующий символ, команду или выполнить действие.

Ознакомившись с тайными лабиринтами программы `EDITOR`, предлагаю приступить к практике и выманить Стрелочку на поверхность.

Глава 6
Клавиатура

Краткое содержание: ввод символов, запись чисел в память, LAST_K, FLAGS

Дёргать по пустякам Стрелочку вариант не очень хороший. Неприлично трогать хозяйку в её собственном домике. Предлагаю выманить королеву путём дистанционного нажатия клавиш и набора команд. Например, «виртуально нажать» клавишу «a», чтобы вывелась заглавная буква «A».

Выполните команду «*Debugger*», открыв отладчик. Следом командой «*Go To 23560*», перейдите на данный адрес памяти, как описывалось в главе 4 на примере одного из вариантов программы «Сброс». На всякий случай повторю. В поле маленького окошечка впишите число 23560:

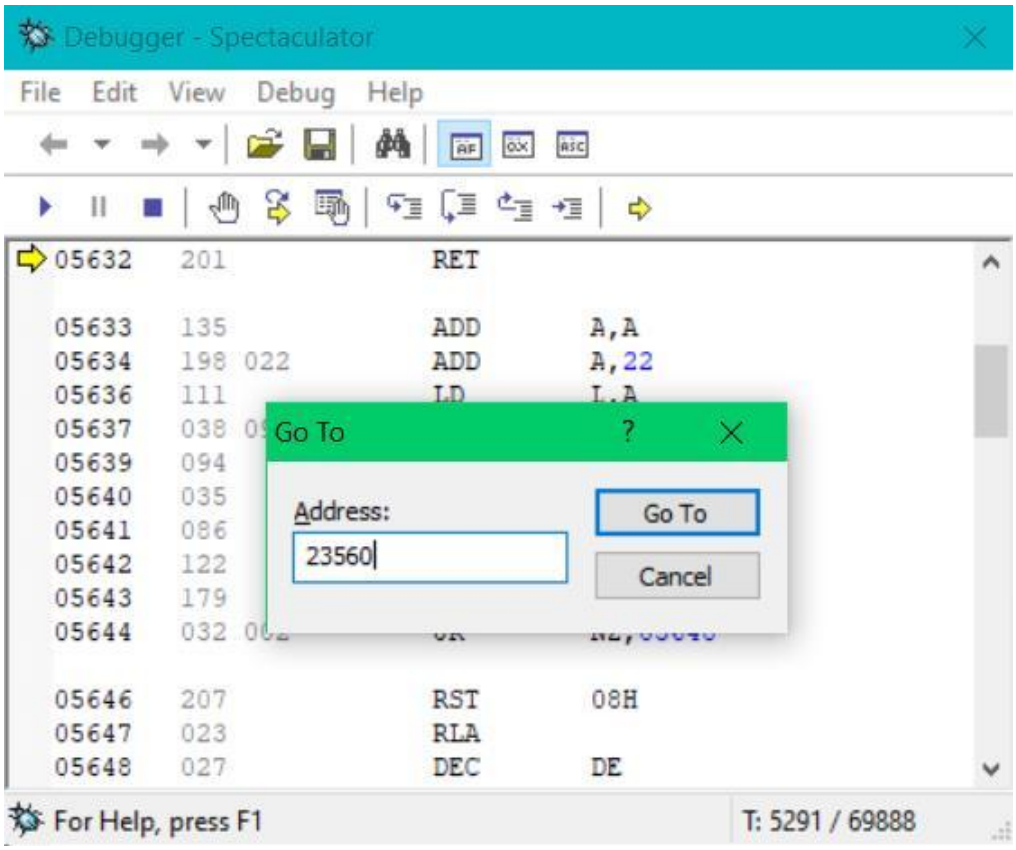


Рис. 46. Окошко «Go To». Быстрое перемещение по адресам.

Нажав «*ENTER*» или кнопку «*OK*», вы перенесётесь к нужному адресу:

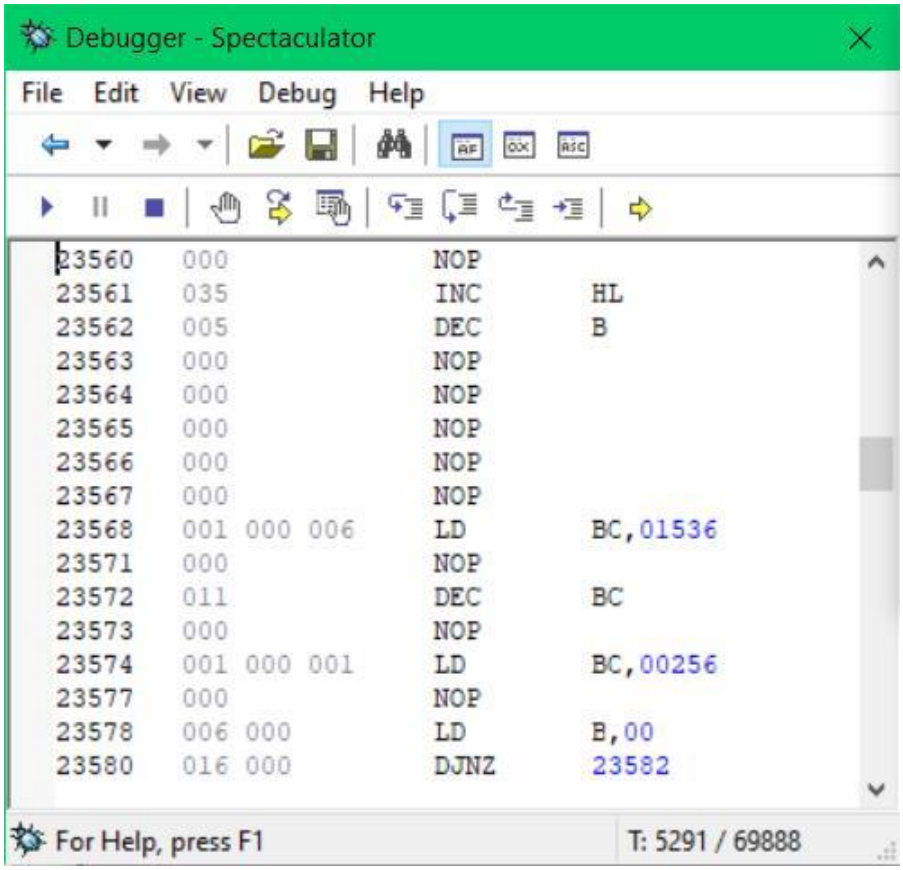


Рис. 47. Результат быстрого перемещения к адресу 23560.

Вы очутились в области по нужному адресу. Наведите курсор на значение ячейки «23560» и нажмите правую кнопку мышки:

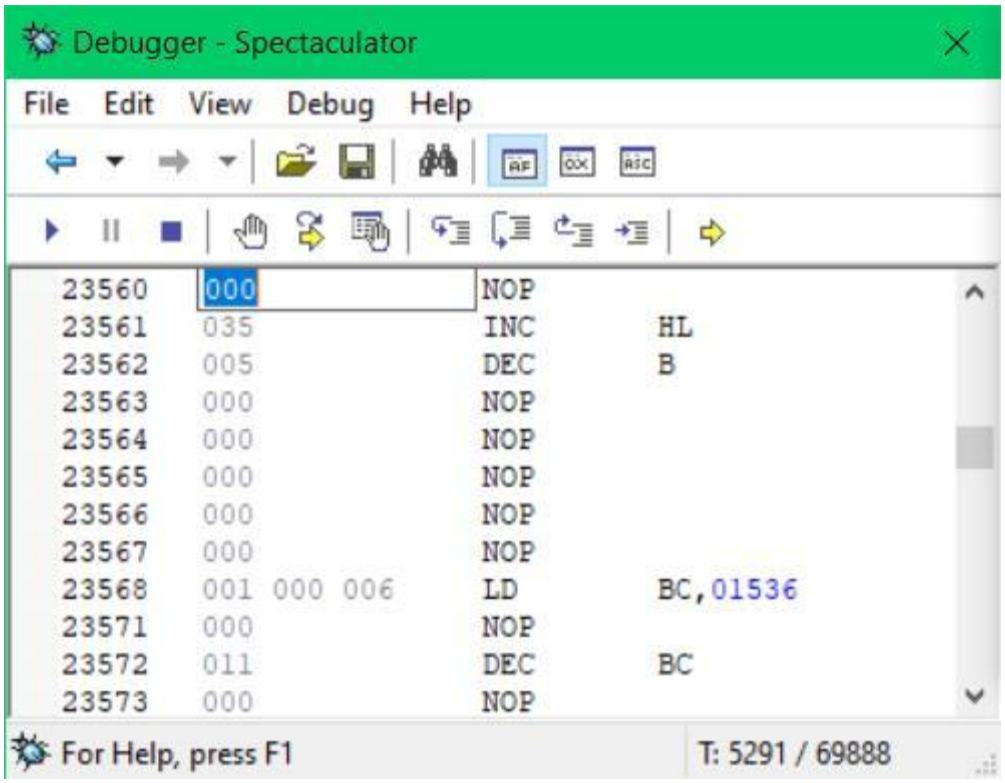


Рис. 48. Редактирование значения в ячейке памяти.

Значения в строке посинели. Ну а теперь начинайте вписывать в эту ячейку число «65», что соответствует коду буквы «А»:

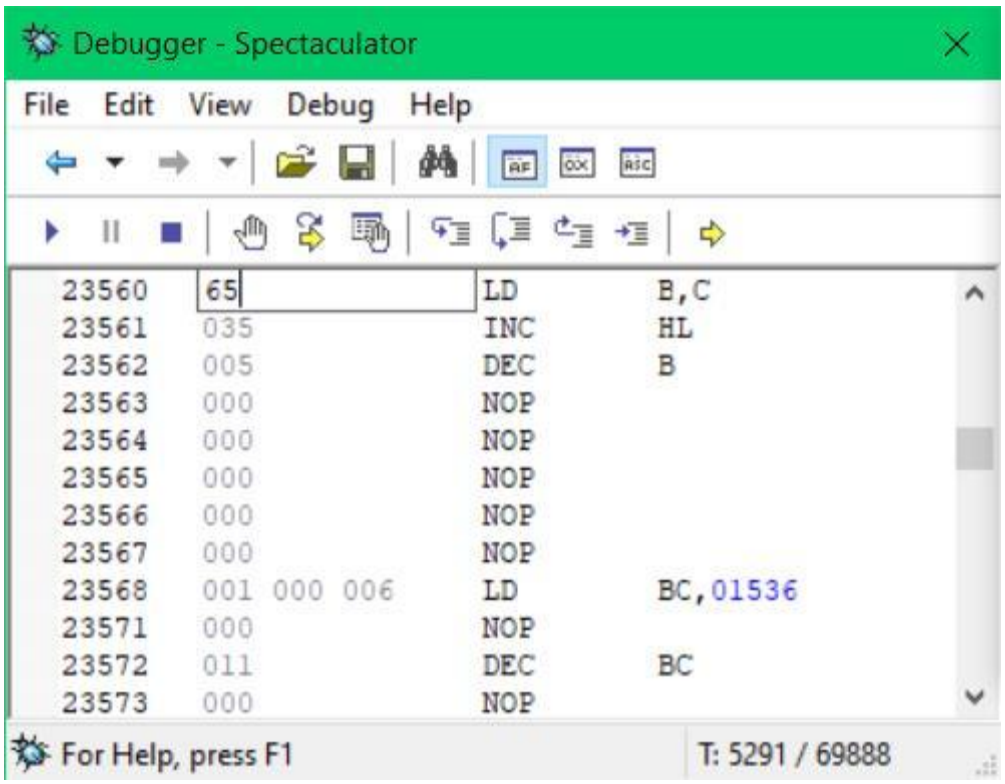


Рис. 49. Вписывание нового значения в память.

Старое выделенное число автоматом стёрлось и на его месте стоит напечатанное вами. Вводите:

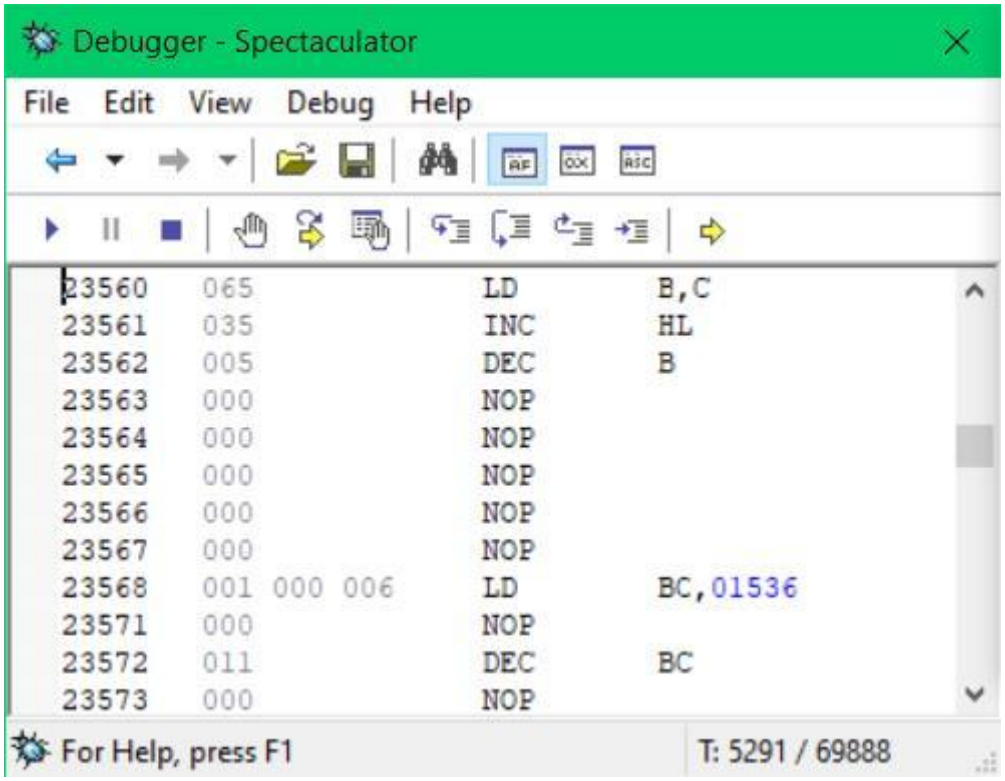


Рис. 50. Ввод нового значения в ячейку памяти LAST_K (23560).

После нажатия «*ENTER*», новое значение принялось и посерело. Можно приступать к записи второго значения. Поскольку второй адрес находится неподалёку, колёсиком мыши или курсорными клавишами, прокрутите вниз список памяти до адреса 23611. Точно также нажмите на значение ячейки памяти и введите туда число 32, что будет означать активацию бита №5, поскольку число 32 равно 00100000 в побитном формате:

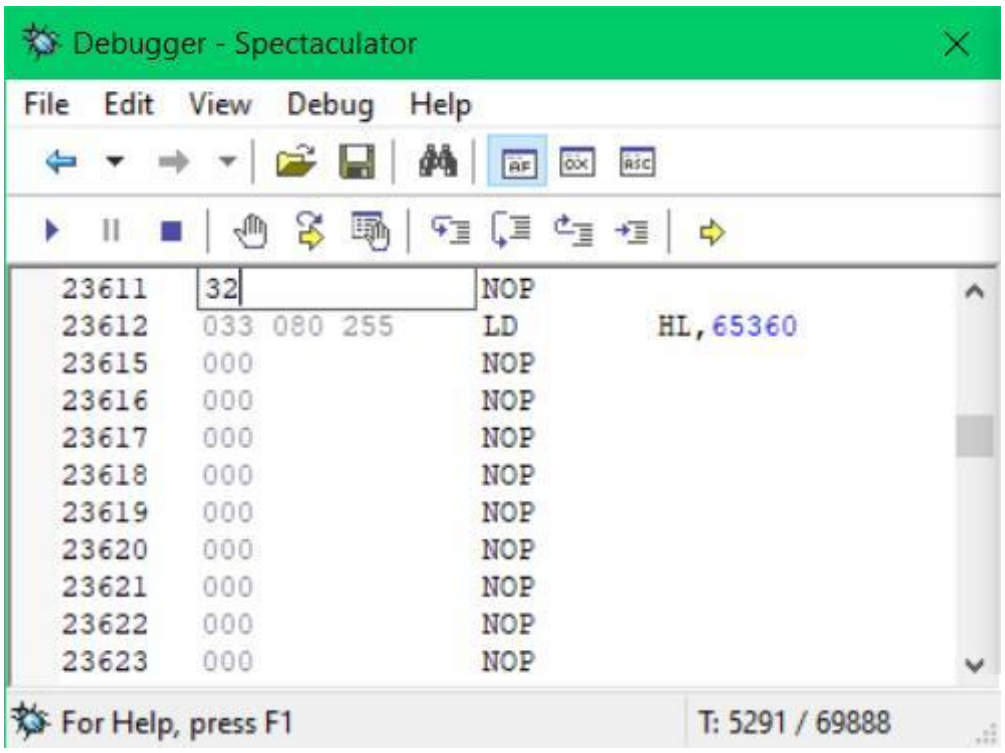


Рис. 51. Ввод нового значения в FLAGS по адресу 23611.

Вводите и это значение. Запись новых чисел в ячейки памяти ничем не отличается от замены значений в регистрах, Теперь точно также запустите программу кнопочкой «*Trace (F5)*» в виде синего ▸ треугольничка, или кнопкой «*F5*»:

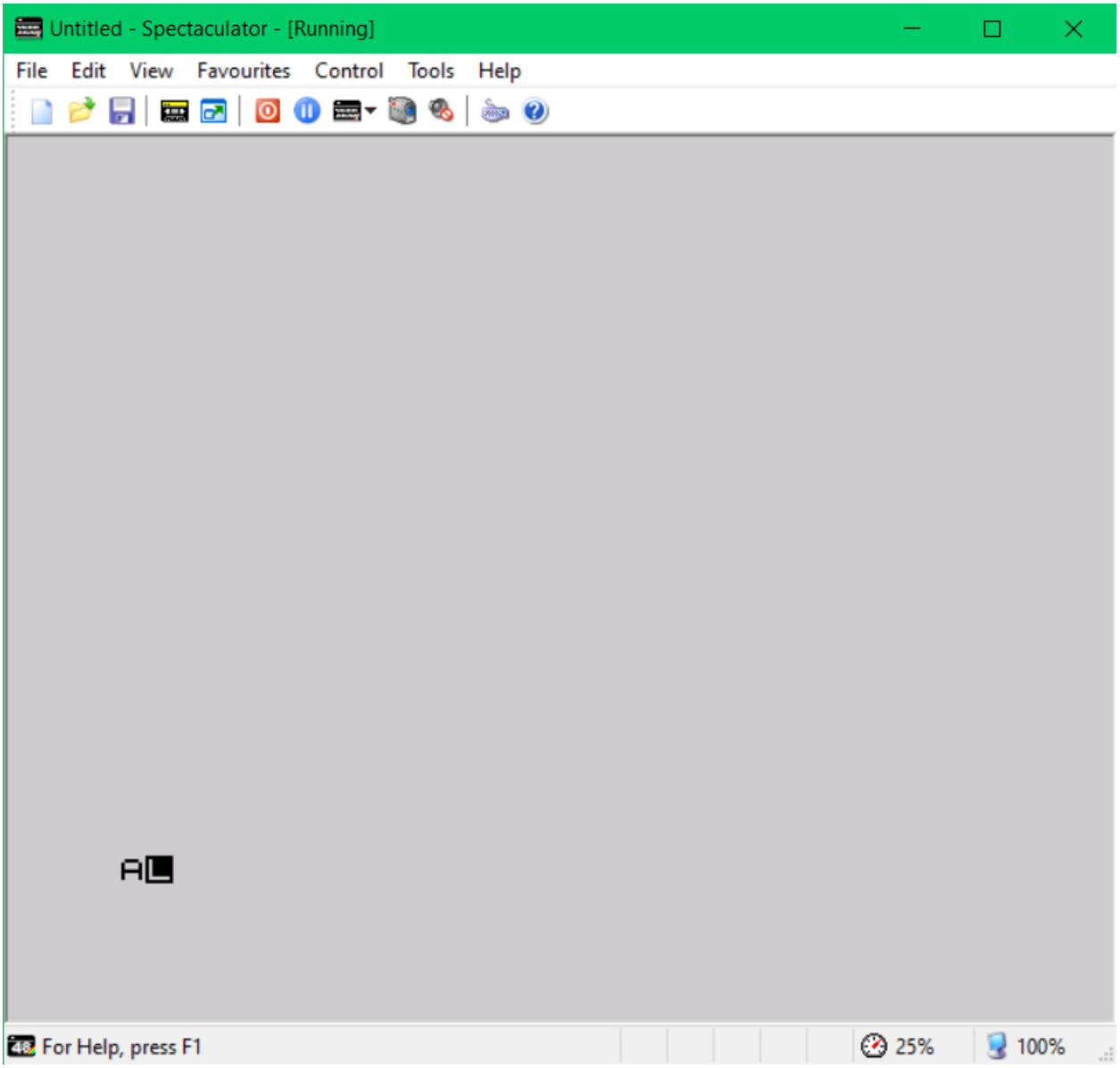


Рис. 52. Вывод буквы «A» в результате прямой записи значений в память.

Гениально, правда? Забив на мигающий курсор K, на экран выдалась заглавная буква A. При таком способе вывода, курсоры не имеют никакого значения. Вы напрямую задаёте действие, символ или целую команду без учёта текущих режимов ввода в обход всех правил и логики.

Снова зайдите в «*Debugger*». В ячейку `FLAGS (23611)` выставите 32, а в `LAST_K (23560)` попробуйте ввести число 231:

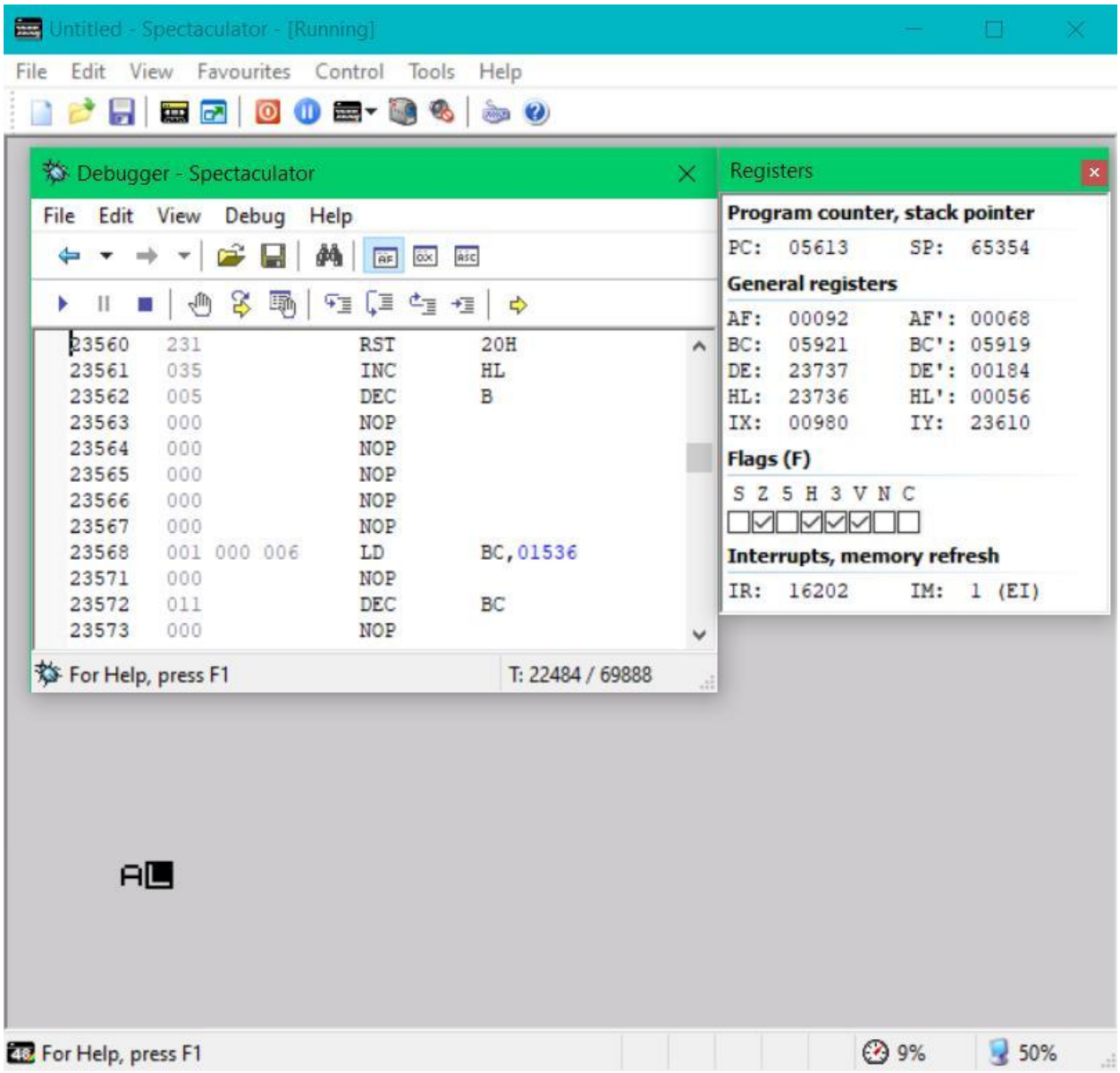


Рис. 53. Процесс ввода числа 231 в ячейку 23560.

Иначе говоря, «23611 ← 32», «23650 ← 231» и нажимайте «*Trace (F5)*». Аналогичного результата можно добиться, если даже просто закрыть окно отладчика крестиком:

Рис. 54. Вывод команды BORDER искусственным способом. Фрагмент нижней части экрана.

А теперь, в режиме курсора L выдалась команда BORDER, которая традиционным методом не набралась бы с клавиатуры после буквенного символа.

Таким же образом вы можете набрать любой символ любого режима. Для этого достаточно открыть приложение «*Полный набор символов*» в любой старой книге и поэкспериментировать с разными значениями.

Как вы могли заметить, с этой главы, в копилку рождающегося языка добавилась новая команда:

«*X* ← *Y*» – Ввод в ячейку памяти X значения Y, где X=0…65535, Y=0…255.

На алгоритмическом языке «*God Mode*» эти две программы будут выглядеть так:

Печать буквы «A»:

```
Debugger
Dec
Go To 23560
23560 ← 65
23611 ← 32
Trace
```

И вторая программа вывода команды BORDER:

```
Debugger
Dec
Go To 23560
23560 ← 231
23611 ← 32
Trace
```

По-моему, неплохо вписалась очередная новая команда.

Но при такой куче преимуществ, у этого метода есть и недостаток. За раз дистанционно можно напечатать только одну команду и после каждого такого захода в «*Debugger*» нужно в ячейке FLAGS (23611) включать бит №5, выставляя эквивалентное число 32. А в целом, управление клавиатурой из «*God Mode*» ломает все стереотипы прошлого и меняет подходы к программированию.

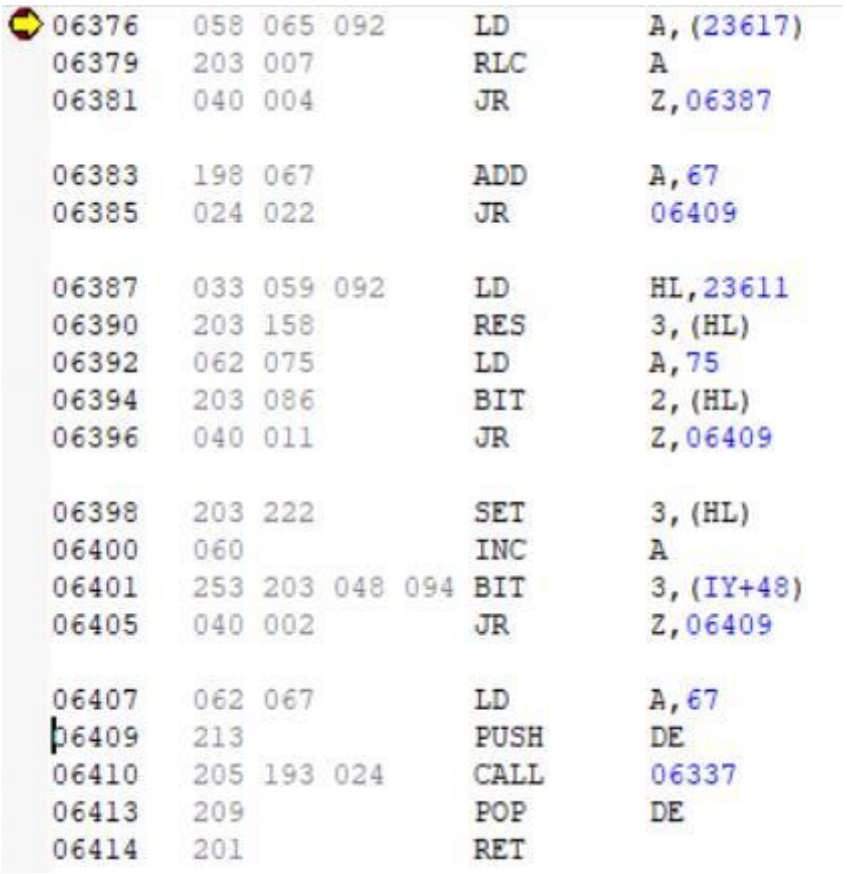
Глава 7
Удивительный режим MODE

Краткое содержание: курсоры, режимы, MODE (23617), TV_FLAG (23612)

По классике литературного жанра дальше нужно знакомиться с курсорами. А почему бы и да! Ведь, этим же способом выставляются управляющие символы. Печатая разные управляющие коды, можно передвигать курсор по тексту, редактировать и вводить строку, выставлять режим E, G или C, удалять отдельные элементы и выставлять цветовые атрибуты.

Как известно из любого BASIC-самоучителя, разновидностей курсоров бывает пять штук: C, E, G, K и L. В паре с курсором или отдельно иногда может мигать знак вопроса ?. Предлагаю разобраться, каким образом они формируются.

Каждый раз при нажатии очередной клавиши происходит добавление символов в строку, либо другая манипуляция, связанная с её модификацией. Если обновление текста в строке иногда бывает частичным, то курсор нужно обновлять после каждого чиха, убирая из старой позиции. За его печать отвечает подпрограмма PRINT THE CURSOR по адресу 6369. Отбор кода символа для курсора на печать происходит с адреса 6376:



```
06376   058 065 092     LD      A,(23617)
06379   203 007         RLC     A
06381   040 004         JR      Z,06387

06383   198 067         ADD     A,67
06385   024 022         JR      06409

06387   033 059 092     LD      HL,23611
06390   203 158         RES     3,(HL)
06392   062 075         LD      A,75
06394   203 086         BIT     2,(HL)
06396   040 011         JR      Z,06409

06398   203 222         SET     3,(HL)
06400   060             INC     A
06401   253 203 048 094 BIT     3,(IY+48)
06405   040 002         JR      Z,06409

06407   062 067         LD      A,67
06409   213             PUSH    DE
06410   205 193 024     CALL    06337
06413   209             POP     DE
06414   201             RET
```

Рис. 55. Подпрограмма PRINT THE CURSOR. Формирование буковок-курсоров.

В «A» берётся проба содержимого переменной `MODE (23617)` с режимами ввода и делается её анализ путём удвоения считанного значения. Если произошла реакция, и число умножилось на два, значит, ввод с клавиатуры происходит в графическом или расширенном режиме и требуется напечатать курсор **E** или **G**. Умноженное значение для расширенного режима станет 2, а для графического – 4. Добавив 67 к этому модифицированному содержимому, получается готовый код ASCII символа курсора **E** (69) или **G** (71). Всё готово и можно переходить по адресу 6409 для подготовки к выводу мигающей буковки.

Если реакции на попытку умножения не произошло, то в `MODE (23617)` лежит нолик. Сколько 0 не умножай, ничего не выйдет, а значит, требуется напечатать курсор **K**, **L** или **C**, но какой именно, нужно разбираться и идти на дополнительную проверку ситуации по адресу 6387.

Для дальнейшего распознавания, в «HL» берётся адрес знакомого «пульта управления», она же ячейка `FLAGS (23611)`. Не выяснив истинного буквенного режима, первым делом выключается бит-тумблер №3 (8), отвечающий за ожидание ввода в режиме курсора **L**. Тем самым принудительно устанавливается курсор **K**. В «A» прямой загрузкой выставляется ASCII код 75, соответствующий буковке «K».

Таким образом, дальнейшее распознавание выполняется через призму курсора **K**, который поставили по умолчанию, а следом пойдёт уточнение и корректировка от противного.

Теперь начинается проверка правильности, сгоряча поставленного курсора **K**. Рассматривается состояние бит-тумблера №2 (4), который отвечает за режим ввода символа в режиме **L** или **K**. Если отключен и он, значит перед вами настоящий курсор **K** и можно переходить к его выводу.

Дальше осталось понять это курсор **L** или **C**. Поэтому включается назад несправедливо выключенный бит №3. Прибавив единичку к сохранившемуся коду символа «K», получается символ для курсора **L**. Для окончательного подтверждения

смотрится бит-тумблер №3 переменной FLAGS2 (23658). Если он выключен, значит можно переходить к печати курсора L.

Ну а если он включен, то вариантов не остаётся, потому что перед вами курсор C, поэтому в «A» записывается значение 67 – кода буковки «C». Следом происходит переход на печать курсора. Примерно так это всё происходит. В теории принцип печати курсора не сильно сложный, хоть и запутанный.

Предлагаю вывести курсор E. Этот курсор напоминает  «Ешку» – выход из уровня игры «*Rockfall*». Вариантов вывода этого курсора, как минимум, три: два на голый компьютер с заставкой и один в качестве добавки к имеющейся строке.

Первый вариант по мотивам прошлой главы выглядит следующим образом:

```
Debugger          ; Открыть отладчик
Dec               ; Переключить отображение чисел в десятичный режим
Go To 23560       ; Переместиться на нужный адрес
23560 ← 14        ; Задать код курсора E
26611 ← 32        ; Включить тумблер №5 активации нажатия клавиши
Trace             ; Выйти и запустить программу
```

Второй вариант установки курсора E более интересный. Из вышесказанного понятно, что по адресу MODE (23617) нужно выставить значение «1», а дальше опять обновить содержимое экрана.

Выполните сброс  любым удобным способом, а затем введите и выполните такую программу:

```
Debugger          ; Открыть отладчик
Dec               ; Переключить отображение чисел в десятичный режим
Go To 23560       ; Переместиться на нужный адрес
23560 ← 9         ; курсором влево и сбить текст заставки
23611 ← 32        ; Включить тумблер №5 активации нажатия клавиши
23617 ← 1         ; Установить режим курсора E
Trace             ; Выйти и запустить программу
```

После запуска, заставка сотрётся и на экране замигает курсор E:

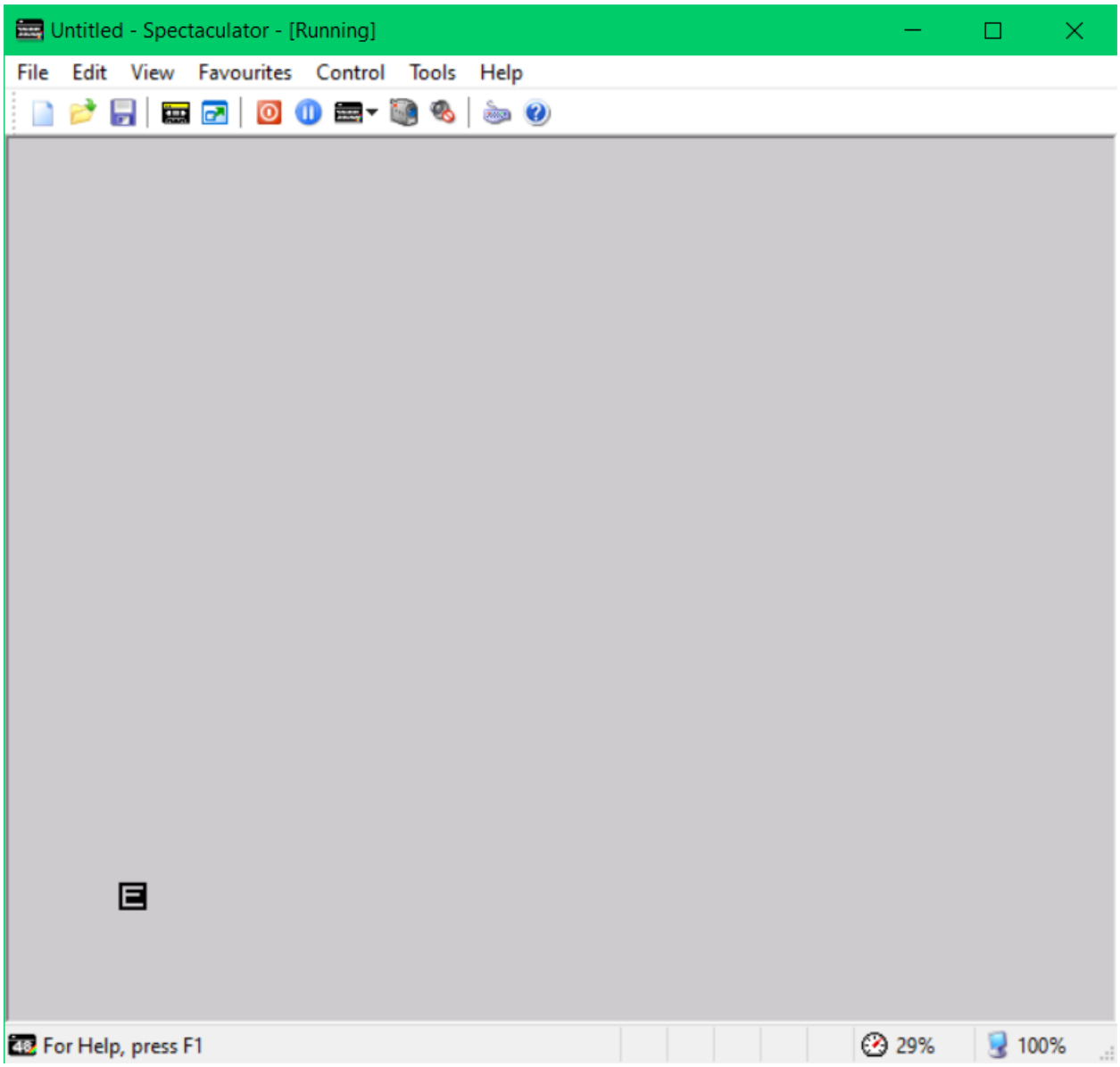


Рис. 56. Вывод курсора E искусственным способом.

Для верности нажмите какую-нибудь клавишу, например «S»:

Рис. 57. Вывод символа с помощью синтезированного курсора E. Фрагмент нижней части экрана.

Но есть и еще один способ. Он более специфичный и подойдёт только для вывода курсора к уже имеющемуся тексту в буфере редактора.

Третьим вариантом предлагаю опробовать метод локального обновления строки с помощью переменной `TV_FLAG` по адресу 23612:

TV_FLAG **23612 IY+2 (#5C3C)**

Отдельные биты этой переменной используются для управления выводом на экран.

Бит 0 установлен при выводе на служебный экран, сброшен при выводе на основной экран.
Бит 3 установлен, если режим вывода на экран мог измениться и требуется проверка режима.
Бит 4 установлен при выводе листинга программы.
Бит 5 установлен при необходимости очистить служебный экран (например, перед выводом сообщения).

Рис. 58. Назначение битов TV_FLAG. Скан из старой книги.

Напомню, его состояние проверяется при спуске в `WAIT KEY (5588)`. Для обновления курсора требуется включить бит-тумблер №3, кроме того для вывода изображения в нижние строки должен быть оставлен еще и самый крайний бит. Итоговое числовое значение, которое нужно будет ввести для двух включенных тумблеров 8+1=9. Короче, в 23612 нужно записать значение 9.

Программа будет выглядеть так:

```
Debugger          ; Открыть отладчик
Dec               ; Переключить отображение чисел в десятичный режим
Go To 23612       ; Переместится на нужный адрес
23612 ← 9         ; Включить тумблер №3 и №1
23617 ← 1         ; Установить режим курсора E
Trace             ; Выйти и запустить программу
```

Заходите в «*Debugger*», с помощью «*Ctrl+G*» перепрыгиваете на ячейку 23612, нажимаете, чтобы сменить значение, и опа:

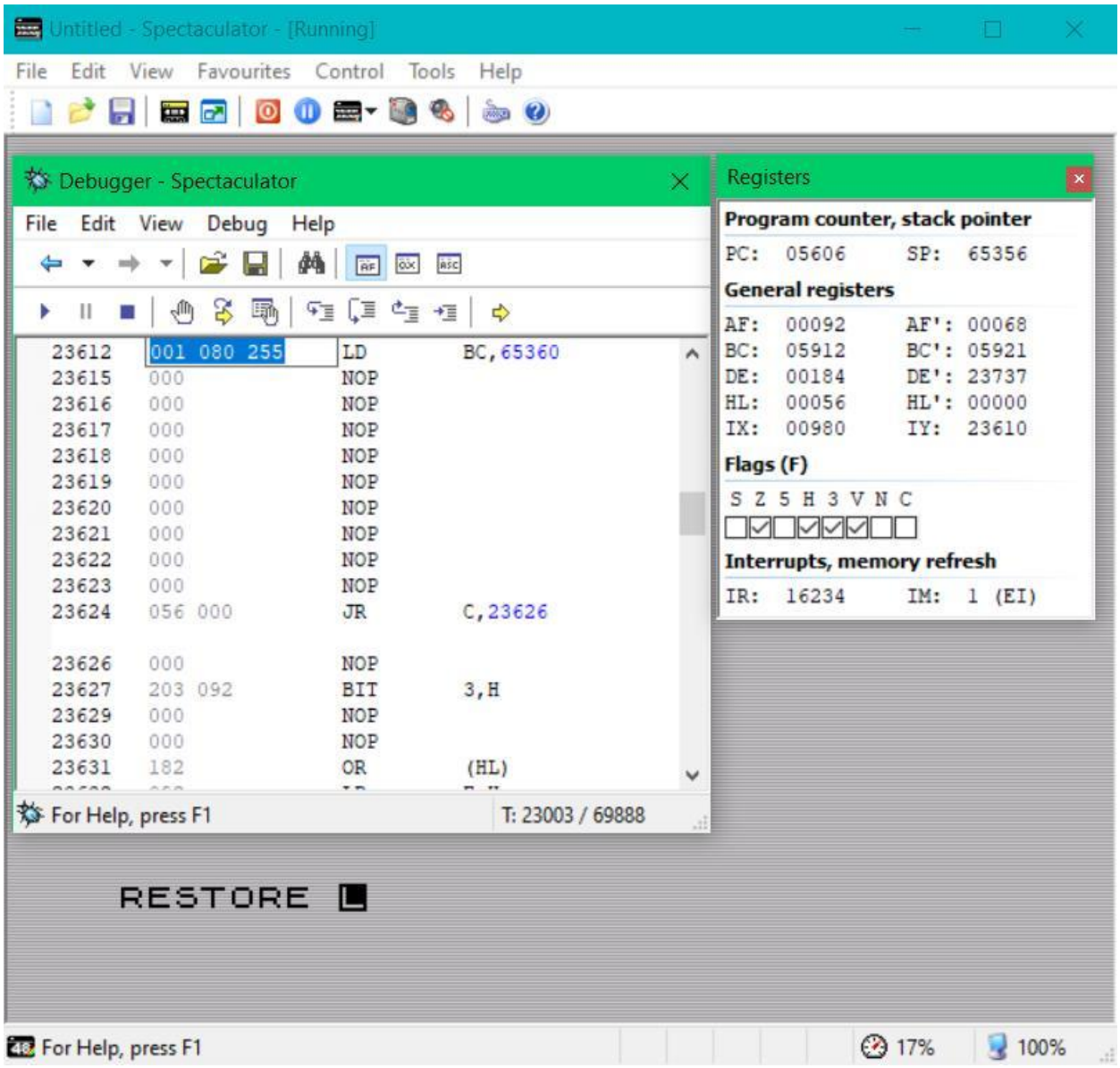


Рис. 59. Выделение значений группы ячеек в отладчике.

Неожиданно выделились все три сгруппированных значения. На самом деле, ничего страшного. Поскольку нужное значение стоит первым, а не вторым или третьим, просто начните набирать число 9:

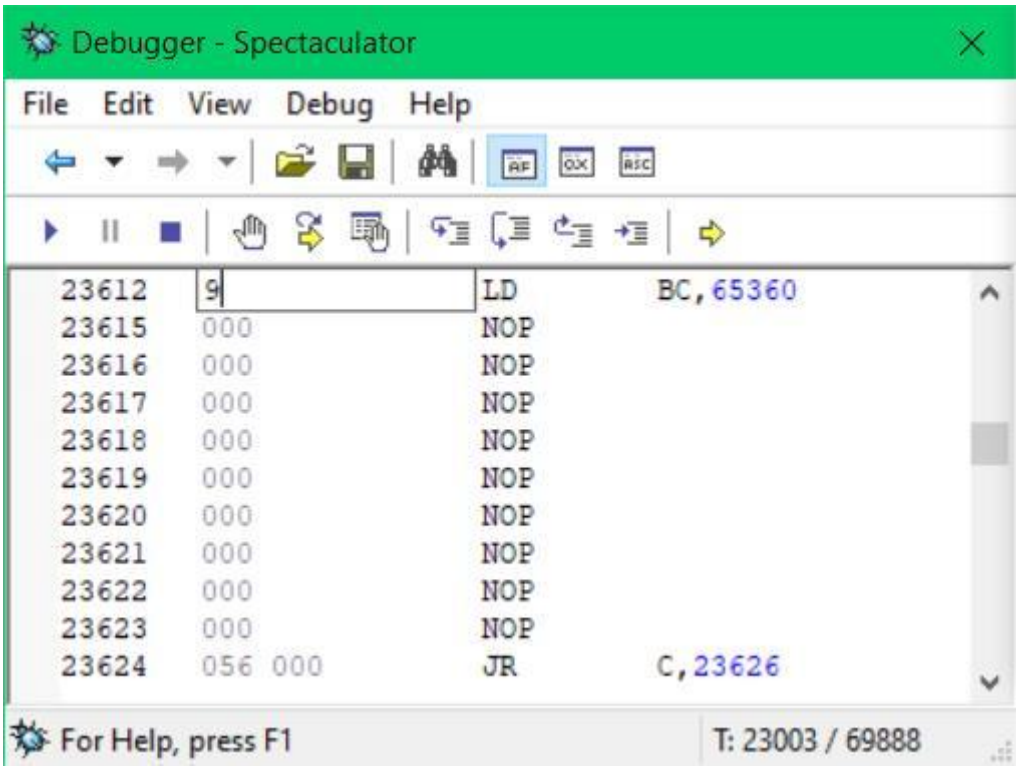


Рис. 60. Ввод числового значения в ячейку.

Проблема исчезла сама собой. А теперь введите. И снова неожиданность, только хорошая. Новое число затёрло только первую ячейку и не повредило остальные:

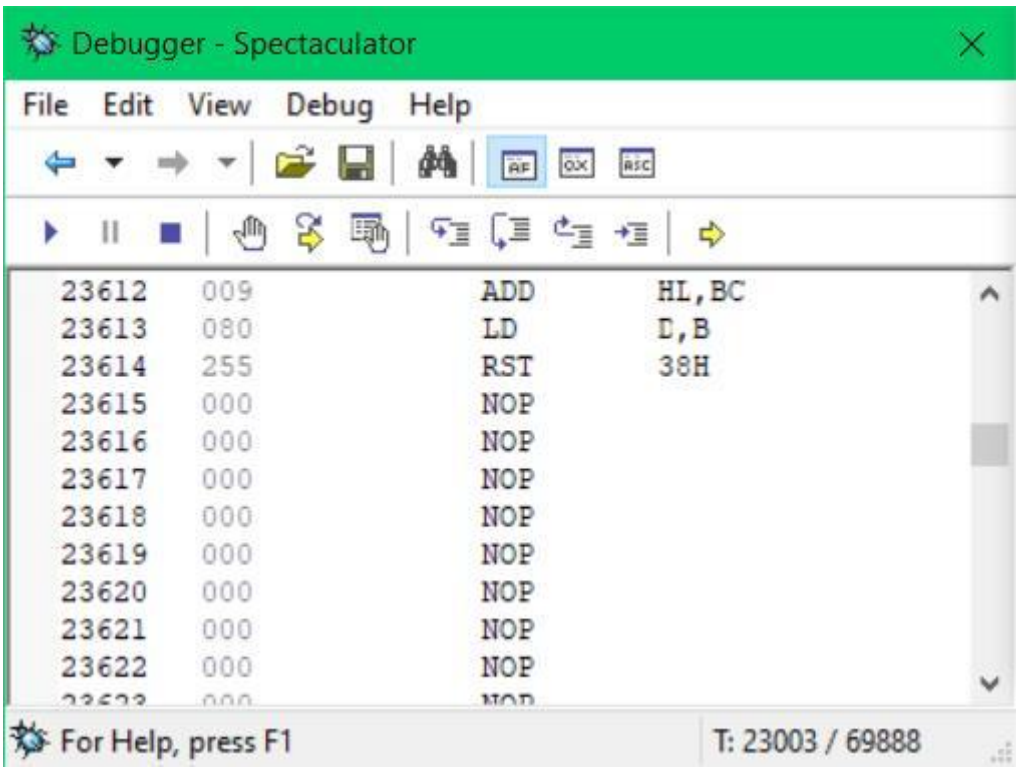


Рис. 61. Замена первого значения в выделенной группе.

Осталось только поменять в `MODE (23617)` значение на 1:

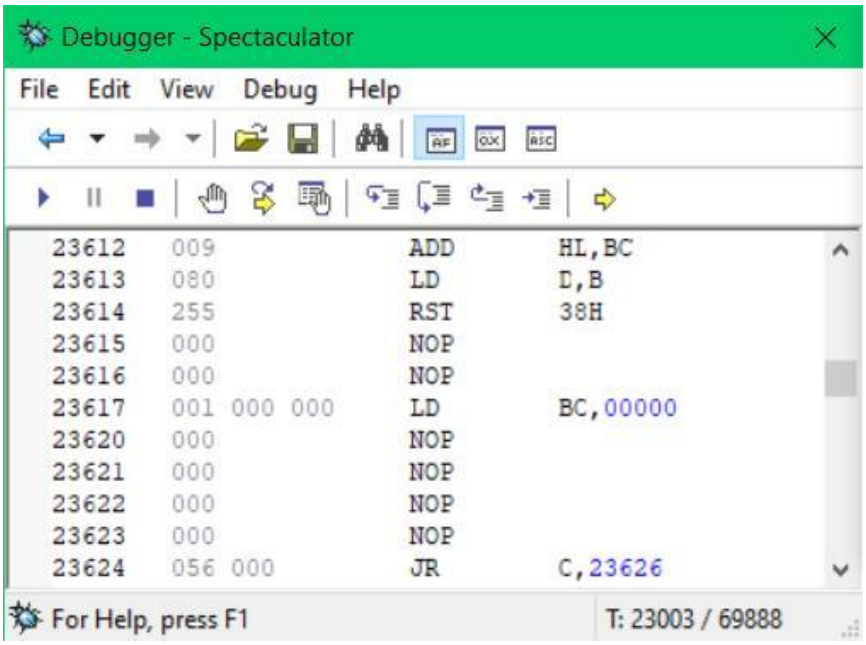


Рис. 62. Ввод числового значения в ячейку.

Ну, вот и всё. Агрегат заряжен нужными значениями. Нажмите «*F5*» и…

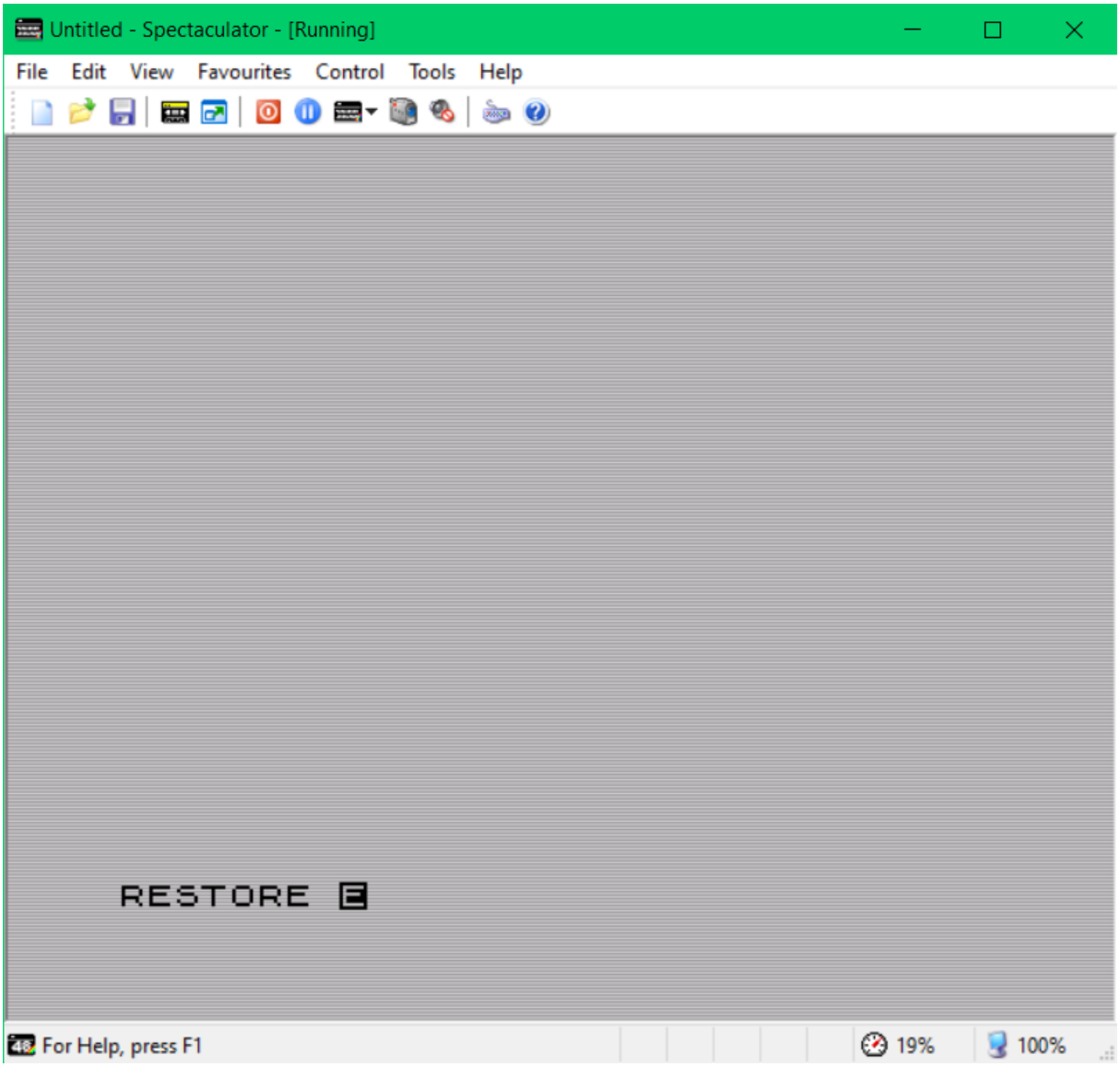


Рис. 63. Появление курсора E.

Как и планировалось, появился курсор E. Всё чики-пуки, а иначе и не должно было быть. Снова что-нибудь нажмите, например «L»:

Рис. 64. Повторная выдача символа синтезированным курсором E. Фрагмент нижней части экрана.

Как видите, никаких новых команд пока не появилось, поэтому особых затруднений вывод курсора E вызвать не должен. Подробно я разобрал лишь потому, что в данном примере пришлось заменять одно из сгруппированных чисел. А теперь допишите к набранному 0 и нажмите ENTER.

После перезагрузки предлагаю попробовать синтезировать курсор G. Его вывод будет аналогичен курсору E:

```
Debugger          ; Открыть отладчик
Dec               ; Переключить отображение чисел в десятичный режим
Go To 23560            ; Переместится на нужный адрес
23560 ← 15        ; Задать код курсора G
26611 ← 32        ; Включить тумблер №5 активации нажатия клавиши
Trace             ; Выйти и запустить программу
```

В этом случае, задав управляющим символом №15 курсор G, активируется нажатая клавиша и происходит вывод курсора на экран.

А вот второй вариант для курсора G будет несколько проще, чем для E:

```
Debugger          ; Открыть отладчик
Dec               ; Переключить отображение чисел в десятичный режим
Go To 23560       ; Переместится на нужный адрес
23611 ← 32        ; Включить тумблер №5 активации нажатия клавиши
23617 ← 2         ; Установить режим курсора G
Trace             ; Выйти и запустить программу
```

Наберите и запустите программу:

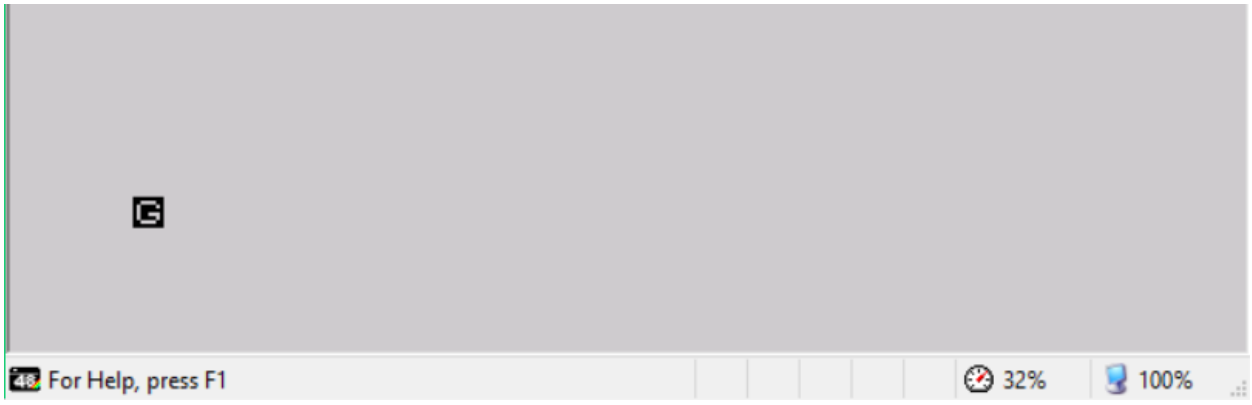


Рис. 65. Вывод курсора G искусственным методом. Фрагмент нижней части экрана.

И вот на экране весело замигал курсор G. Нажмите что-нибудь в «Спектруме», например 123456:

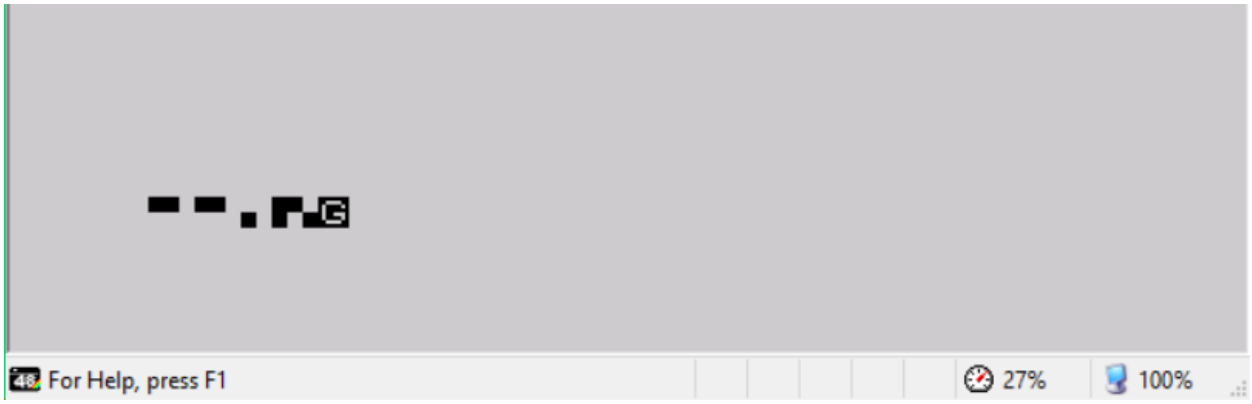


Рис. 66. Проверка реальности синтезированного курсора G. Фрагмент нижней части экрана.

И третьим вариантом точно также будет метод локального обновления к имеющемуся тексту, установкой сразу двух битов (№1 и №3) в ячейку `TV_FLAG (23612)`:

```
Debugger     ; Открыть отладчик
Dec          ; Переключить отображение чисел в десятичный режим
Go To 23612  ; Переместится на нужный адрес
23612 ← 9    ; Локальное обновление нижней строки + вывод туда символа
23617 ← 2    ; Установить режим курсора G
Trace        ; Выйти и запустить программу
```

Какие еще варианты могут быть с этой ячейкой? Можно посмотреть в книжку:

MODE **23617 IY+7 (#5C41)**

Число, определяющее режим ввода с клавиатуры:
0 — очередной символ вводится в режимах курсора **K**, **L** или **C**;
1 — очередной символ вводится в режиме курсора **E**;
2 и более — очередной и последующие символы вводятся в режиме курсора **G**. При значении, превышающем 2, меняется внешний вид курсора.

Рис. 67. Описание переменной MODE из старой книги.

Ответ очевиден: от 0 до 255. Для чистоты эксперимента попытаемся записать в ячейку с номером 23617 число 90, следовательно, наберите такую программу:

```
Debugger
Dec
Go To 23612
23612 ← 9
23617 ← 90
Trace
```

Сначала посмотрите. По программе нетрудно предположить, что после запуска по «*Trace*», с этими значениями, Стрелочка попадёт на подпрограмму `PRINT THE CURSOR (6369)`. Там значение 90 удвоится и добавится 67, после чего должен получиться код 247, который соответствует… команде `RUN`.

Запускайте программу:

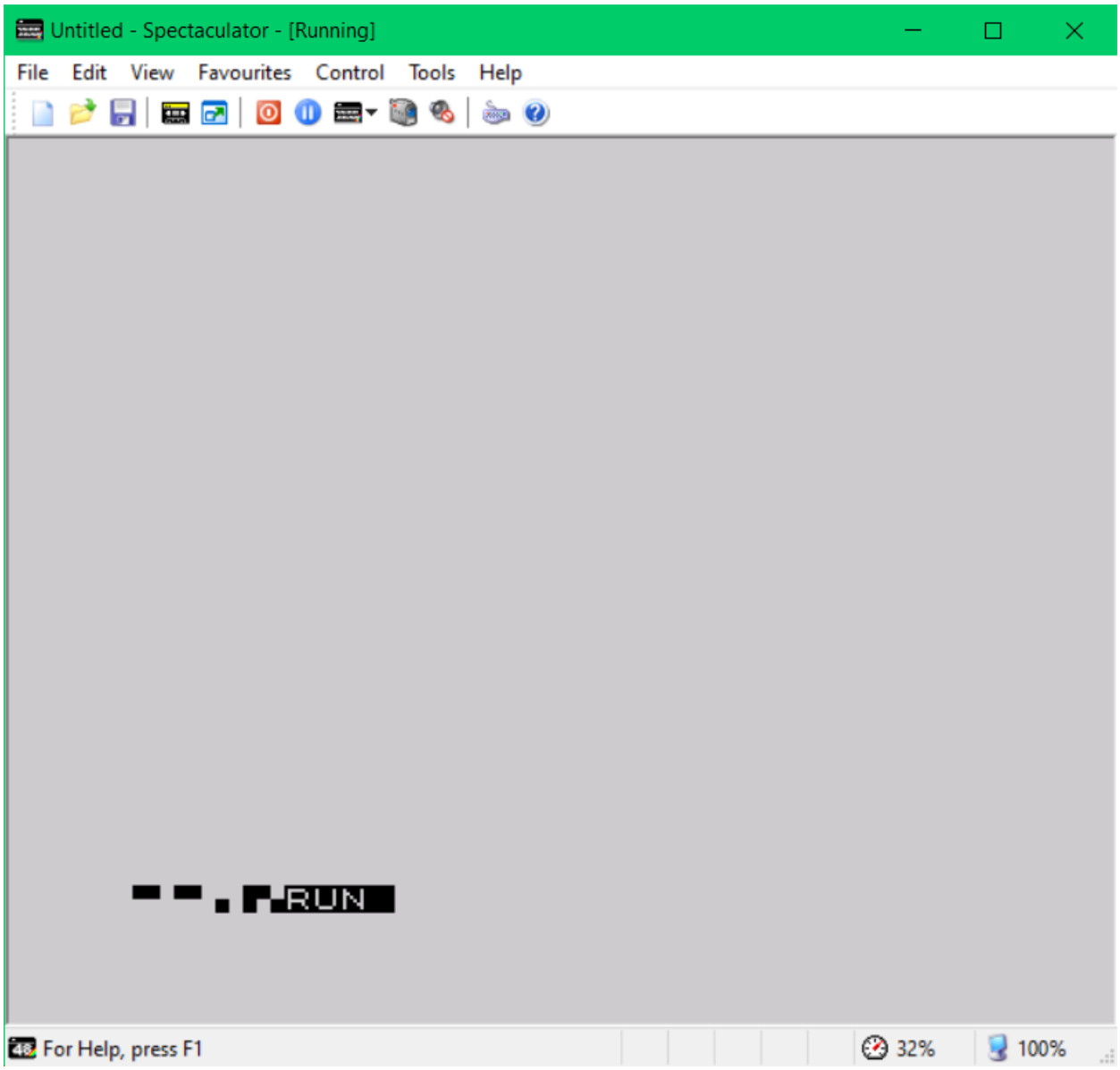


Рис. 68. Spectaculator. Недокументированный курсор «RUN» в графическом режиме.

Так и есть. Однако, несмотря на нестандартный курсор, печать символов всё равно происходит в графическом режиме. Повторите на клавиатуре 123456:

Рис. 69. Проверка печати символов недокументированным курсором «RUN». Нижний фрагмент экрана.

Если вы попробуете традиционным способом выйти из специфического курсора, то при первом нажатии клавиши «9» курсор переключится на G, а при повторном вернётся в K или L в зависимости от ситуации. Выход с клавиатуры будет двухэтапным и вот почему:

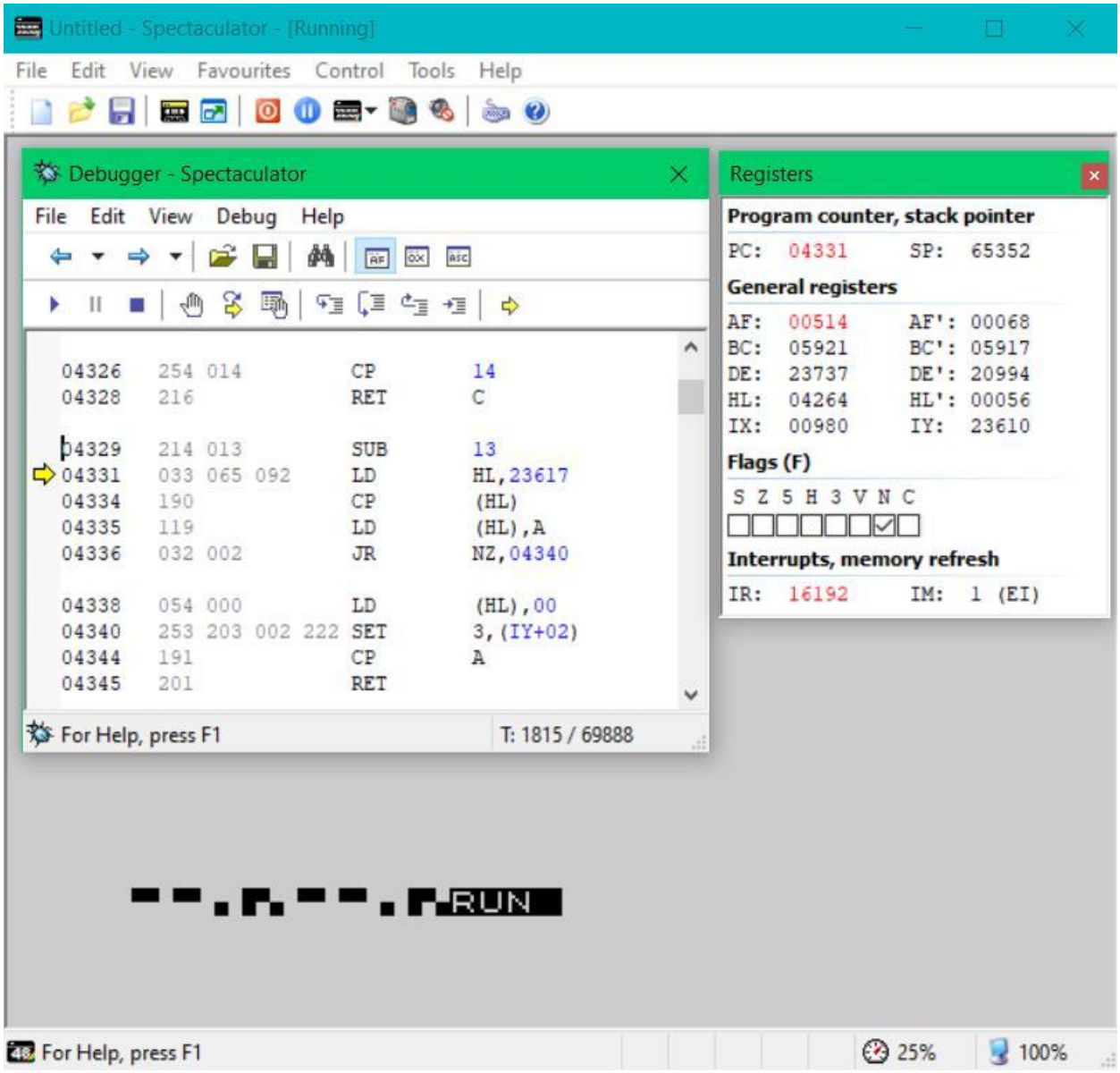


Рис. 70. Причина двухэтапного сброса значения недокументированных курсоров.

На первом этапе переключения, сортируя значение, фрагмент программы `KEYBOARD INPUT` сбросит курсор с едрического значения в привычную двойку, И только со второй попытки произойдёт обнуление ячейки `MODE (23617)`.

Если вы поэкспериментируете с различными значениями, то поймёте, что всё ещё сложнее и необычнее. Попробуйте ввести такую программу:

```
Debugger
Dec
Go To 23612
23612 ← 9
23617 ← 91
Trace
```

Запускайте:

Рис. 71. Появление курсора RANDOMIZE.

И вроде всё логично, что курсор стал RANDOMIZE. А теперь нажмите любую клавишу, например «A»:

Рис. 72.Сброс курсора RANDOMIZE в RUN от нажатия клавиши. Фрагмент нижней части экрана.

Курсор снова превратился в RUN.

Если поэкспериментировать с разными значениями, то становится понятно, что при выводе символов, в нечётных числах, записанных в `MODE (23617)`, теряется единица. Тут проблема уже в подпрограмме `ADD CHAR` программы `EDITOR` по адресу 3969. После распознавания символа вместо записи нуля стоит обнуление бита и непредусмотренному режиму обрубают единичку:

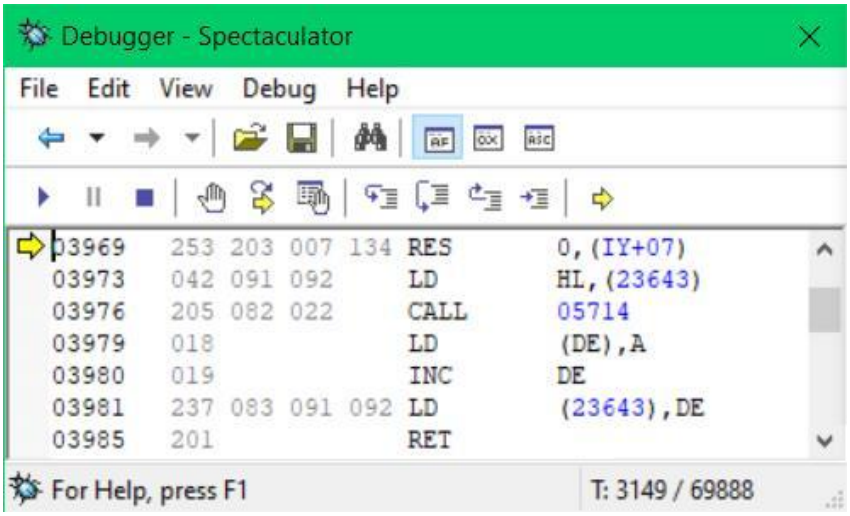


Рис. 73. Причина сброса курсора RANDOMIZE в RUN.

Ну а после значения 129 и по 255, редактор и вовсе входит в вечный командный режим. Это очень интересный эффект, о котором не упоминается в известной мне старой литературе. Возможно, даже эксклюзив:

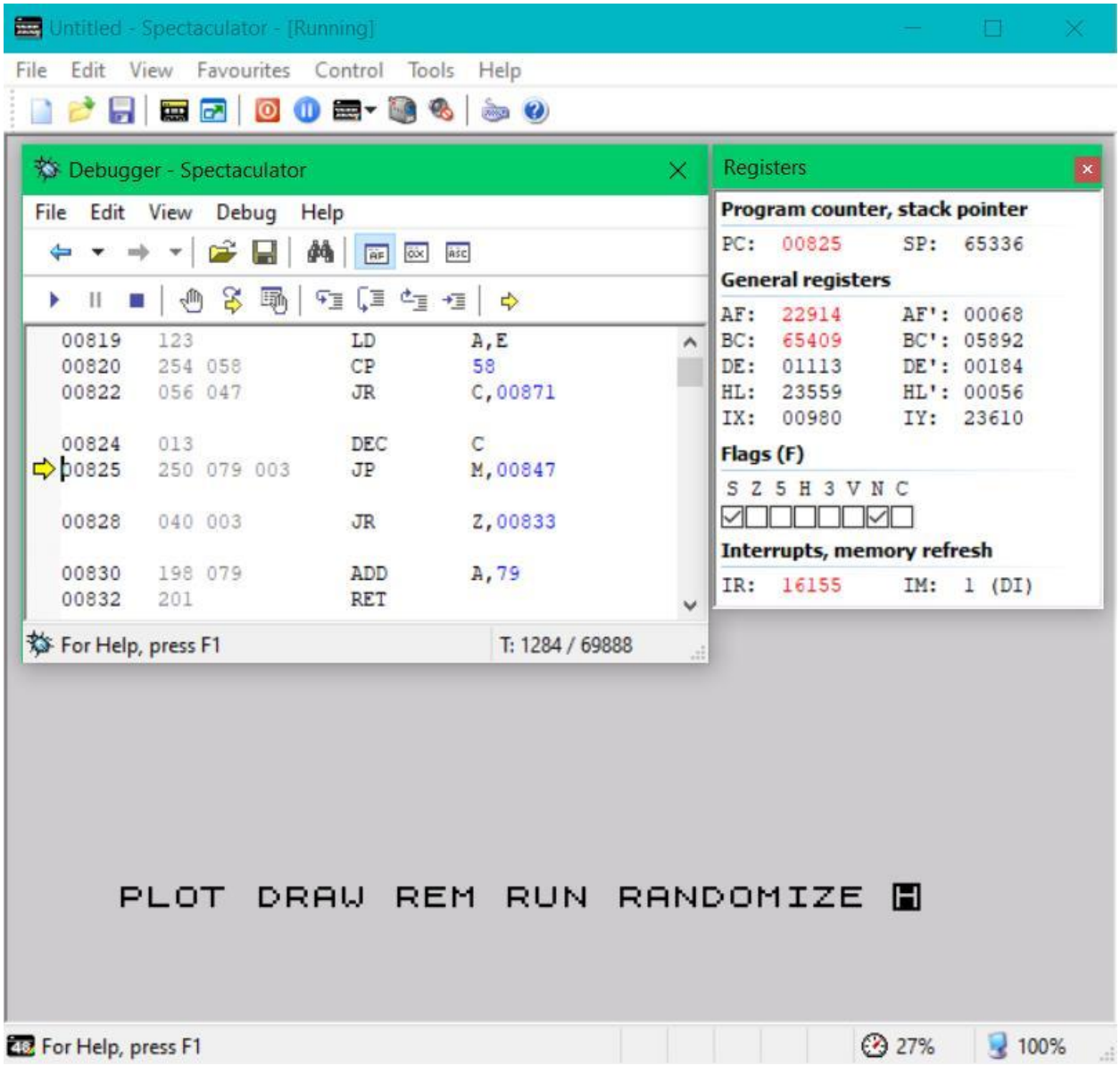


Рис. 74. Вечный командный режим при значениях MODE свыше 129.

Причина такого поведения подсказывает Стрелочка. В программе сканирования клавиатуры `K-DECODE`, числа свыше 128 неправильно трактуются командой «`JP M`» по адресу 825. Вместо прохода в режим графики, как от предыдущих значений, Стрелочку несёт на подпрограмму декодирования клавиши с учётом включенного курсора K.

Вот теперь можно подводить итоги недокументированных режимов `MODE` в упрощённой таблице:

Значение ячейки 23617	Вид курсора	Режим	Примечание
0	K, L, C	Комплексный	Выключение курсоров E или G
1	E	Расширенный	Единоразовый вывод символа в режиме E с последующим сбросом в предшествующий L или C.
2	G	Графический	Графический режим с выходом по клавише «9» в предшествующий L или C.
3	I	Графический	Единоразовый вывод символа в графическом режиме с последующей сменой курсора на G

4	K	Графический	Графический режим с выходом по клавише «9» в G, а при повторном нажатии в предшествующий L или C.
5	M	Графический	Единоразовый вывод символа в графическом режиме с последующей сменой курсора на K.
6	O	Графический	Графический режим с выходом по клавише «9» в G, а при повторном нажатии в предшествующий L или C.
7	Q	Графический	Единоразовый вывод символа в графическом режиме с последующей сменой курсора на O.
8	S	Графический	Графический режим с выходом по клавише «9» в G, а при повторном нажатии в предшествующий L или C.
9	U	Графический	Единоразовый вывод символа в графическом режиме с последующей сменой курсора на S.
…	…	…	…
90	RUN	Графический	Графический режим с выходом по клавише «9» в G, а при повторном нажатии в предшествующий L или C.
91	RANDOMIZE	Графический	Единоразовый вывод символа в графическом режиме с последующей сменой курсора на RUN.
92	CLS	Графический	Графический режим с выходом по клавише «9» в G, а при повторном нажатии в предшествующий L или C.
93	CLEAR	Графический	Единоразовый вывод символа в графическом режиме с последующей сменой курсора на CLS.
…	…	…	…
128	D	Графический	Графический режим с выходом по клавише «9» в G, а при повторном нажатии в предшествующий L или C.
129	F	Командный	Единоразовый вывод команды в командном режиме с последующей сменой курсора на D и переход в графический режим.
130	H	Командный	Постоянный командный режим, не отключающийся после ввода единичной команды. При установке и сбросе курсора E или G, а также EDIT возвращается в предшествующий L или C.
131	J	Командный	Единоразовый вывод команды в командном режиме с последующей сменой курсора на H.
132	L	Командный	Постоянный командный режим, не отключающийся после ввода единичной команды. При установке и сбросе курсора E или G, а также EDIT возвращается в предшествующий L или C.
133	N	Командный	Единоразовый вывод команды в командном режиме с последующей сменой курсора на псевдо L
…	…	…	…
254	▒	Командный	Постоянный командный режим, не отключающийся после ввода единичной команды. При установке и сбросе курсора E или G, а также EDIT возвращается в предшествующий L или C.
255	B	Командный	Единоразовый вывод команды в командном режиме с последующей сменой курсора на ▒

Глава 8
Курсоры K, L и C

Краткое содержание: курсоры, Add / Remove Breakpoint, FLAGS (23611), FLAGS2 (23658)

А вот взаимоотношения курсоров K, L и C можно охарактеризовать известной фразой «всё сложно». Как видно из машинных программ, на которых написаны все эти хитросплетения, главным является курсор K. С ним проблем не будет, так как его переключение вставлено в нескольких местах. Кроме того вывод остальных курсоров на экран производится отталкиваясь от курсора K.

Отправьте Стрелочку по адресу «0», после чего наберите и введите следующую простенькую программу:

```
Debugger
Dec
Go To 23611
23611 ← 32
Trace
```

После запуска на экране замигает курсор K:

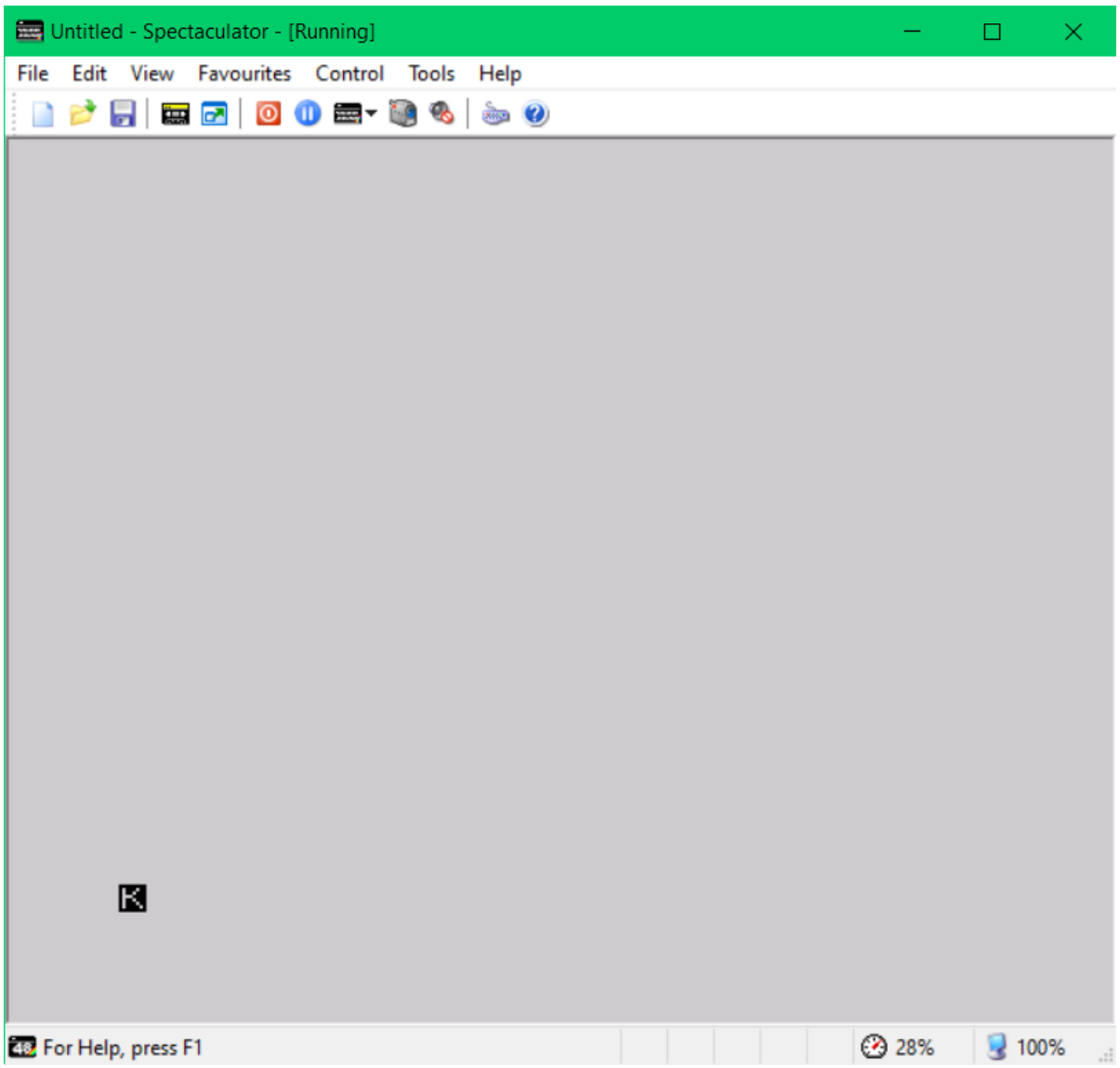


Рис. 75. Установка курсора K на чистый компьютер.

По сути, установка курсора K на голый компьютер, сводится к простому обновлению экрана путём активации бита №5 `FLAGS (23611)`.

Теперь попробуйте вывести курсор L на чистый компьютер. Поскольку у этой семейки сложные взаимоотношения, то для установки курсора L с автоматическим выводом, придётся усложнить задачу. К обновлению экрана битом №5 `FLAGS (23611)`, придётся добавить пару новых приёмов. Нужно остановить Жёлтую Стрелочку в конкретном месте, перед выводом курсора по адресу 6387, после чего добавить в `FLAGS (23611)` код курсора L (4) и отпустить Стрелочку дальше по своим делам.

Откройте «*Debugger*» по адресу 6387. Вот перед этой ячейкой нужно как-то исхитриться открыть отладчик во время выполнения программы. Естественно, случайным образом поймать стотысячные доли секунды, да еще и вслепую просто невозможно. Требуется заловить Стрелочку в этом адресе. Как это сделать? Для этого нужно выставить ловушку с приманкой в виде огромного малинового ● шарика.

Найдите кнопочку с ладошкой и наведите на неё мышку. Всплывёт подсказка «*Add / Remove Breakpoint (Ctrl+Space)*»:

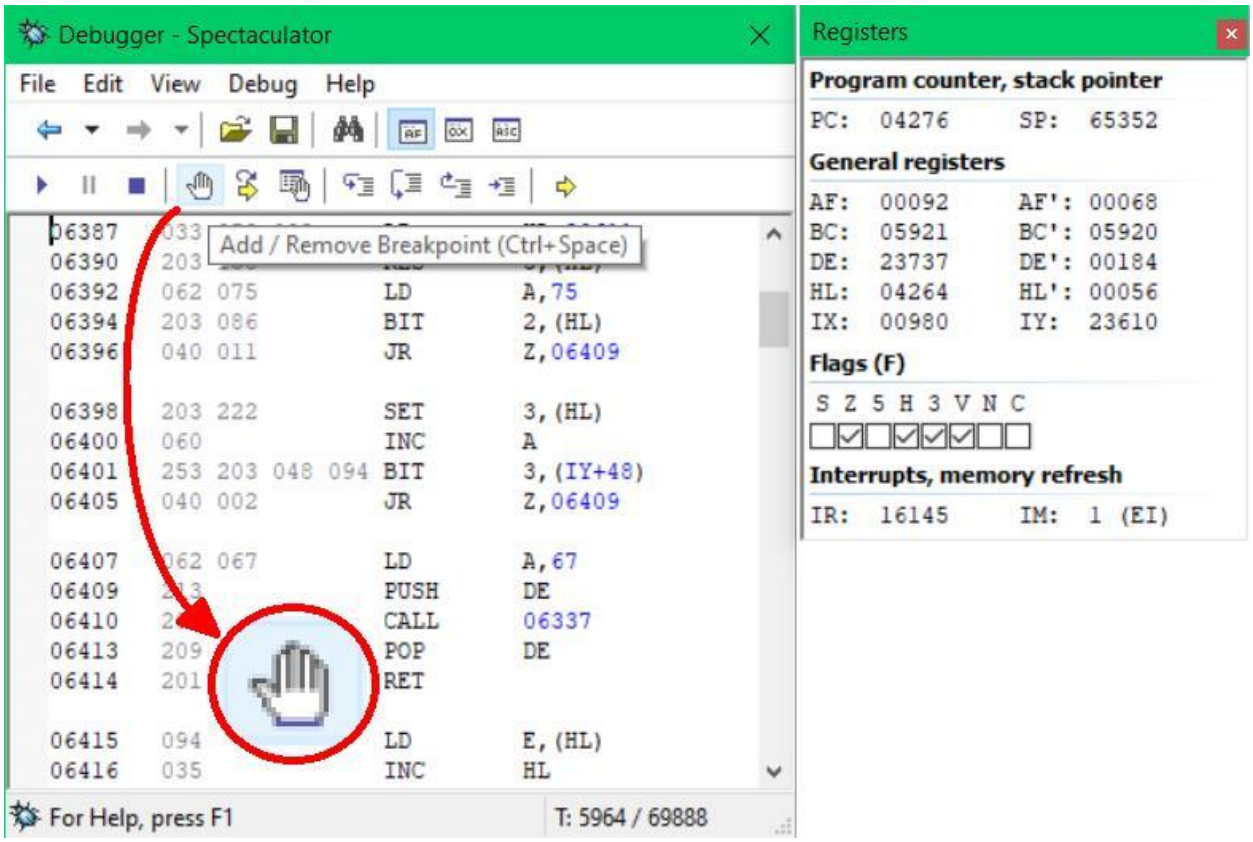


Рис. 76. Кнопка «Add / Remove Breakpoint» в отладчике.

Убедившись, что курсор «палочка» стоит на адресе 6387, нажмите на ручку , и жирная малиновая ● точка-кнопка встанет перед адресом с командой «`LD HL, 23611`»:

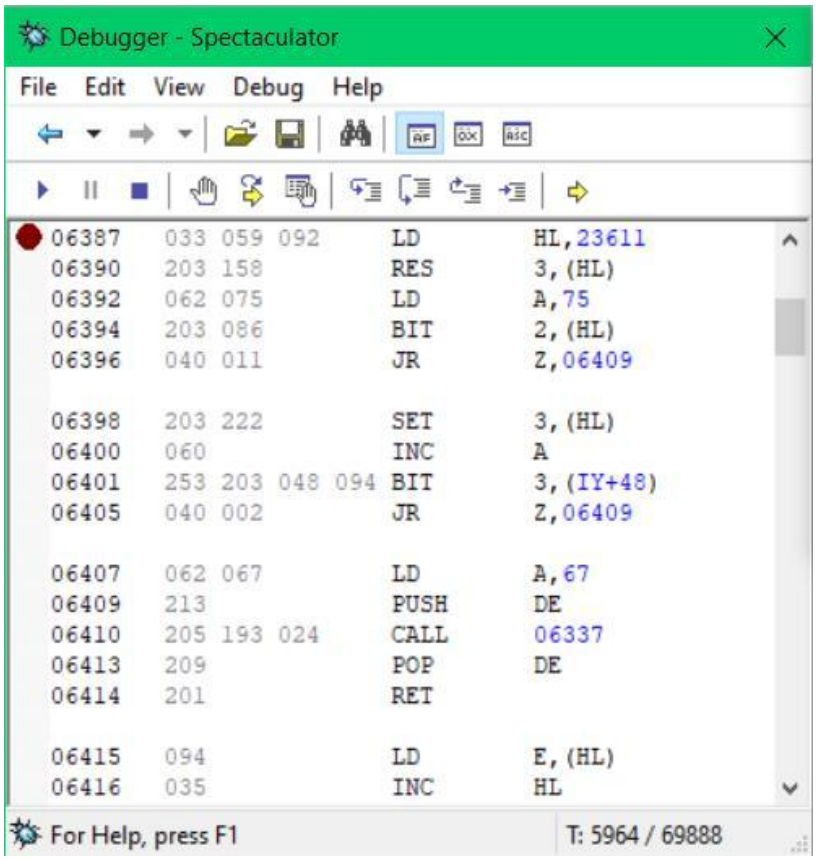


Рис. 77. Установка малиновой точки прерывания по нужному адресу памяти.

Помните про гибкость отладчика. Аналогичного результата можно также добиться, если навести курсор мышки на нужный адрес и два раза подряд щёлкнуть левой кнопкой.

Так или иначе, приманка поставлена. А теперь просто нажмите мышкой на голубенькое число 23611 в команде «`LD HL 23611`», которое для удобства сделано в виде ссылки:

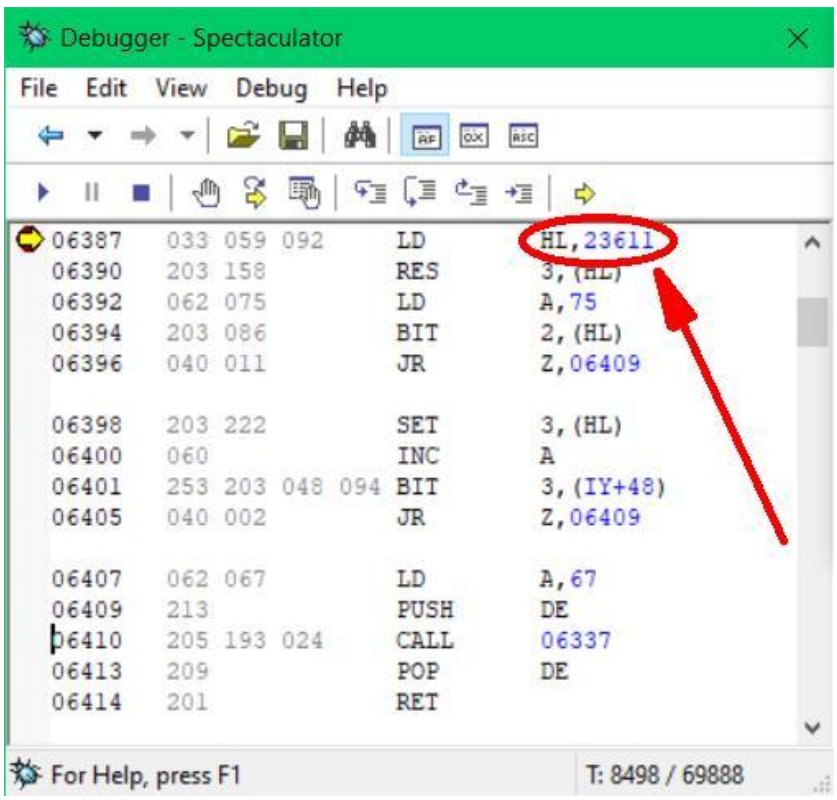


Рис. 78. Параметр со вспомогательной ссылкой.

Вы моментально переместитесь на указанный адрес без дополнительных манипуляций. Оказавшись на нужной ячейке (23611), введите туда значение 32:

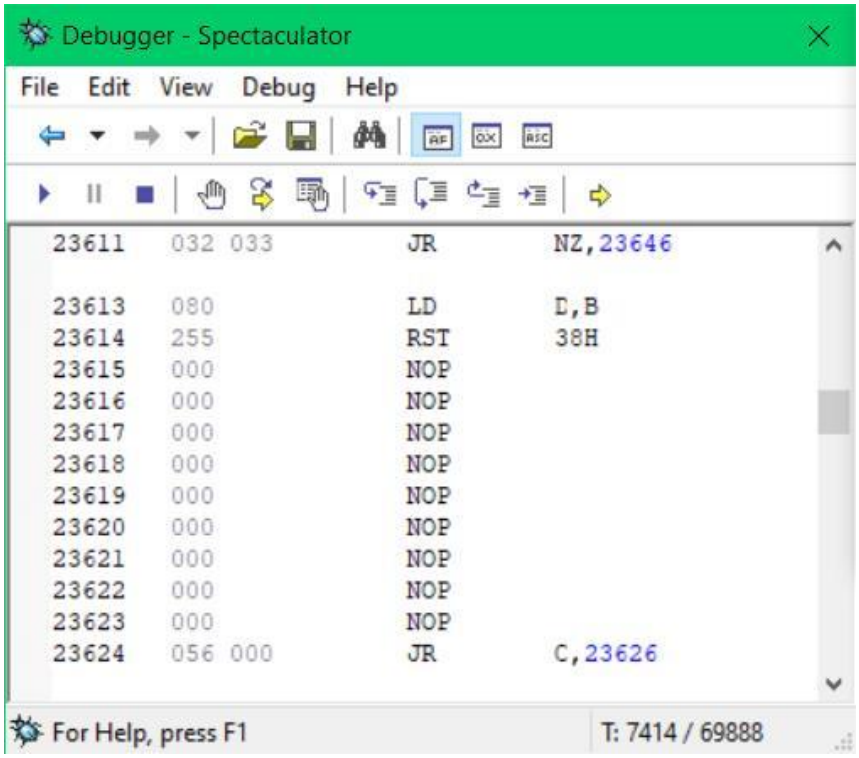


Рис. 79. Ввод значения в ячейку памяти после перехода по вспомогательной ссылке.

Смело нажимайте «*Trace (F5)*».

Окно мигнуло. Не успев закрыться, отладчик снова открылся. Жёлтая Стрелочка ожидаемо попалась на малиновый шарик, установленный по адресу 6387, и чётко легла в него:

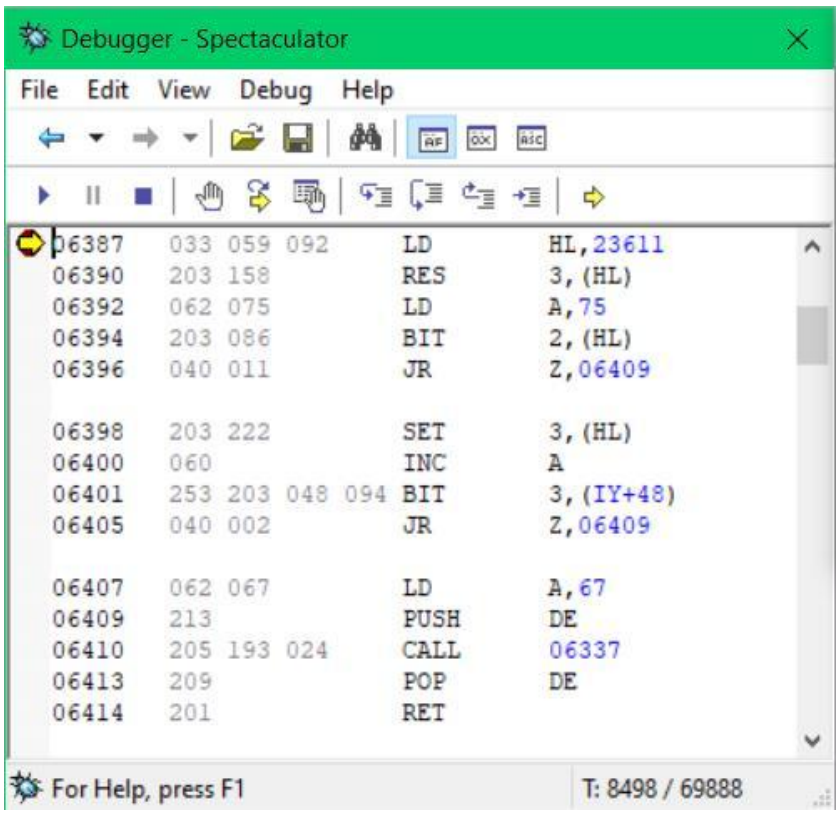


Рис. 80. Остановка Стрелочки на малиновом шарике в заданном месте.

Цель достигнута, и можно освобождать Стрелочку из ловушки.

Снова нажмите ручку , комбинацию «*Ctrl+Space*», либо наведите курсор мыши на Стрелочку с шариком и нажмите два раза подряд левой кнопкой:

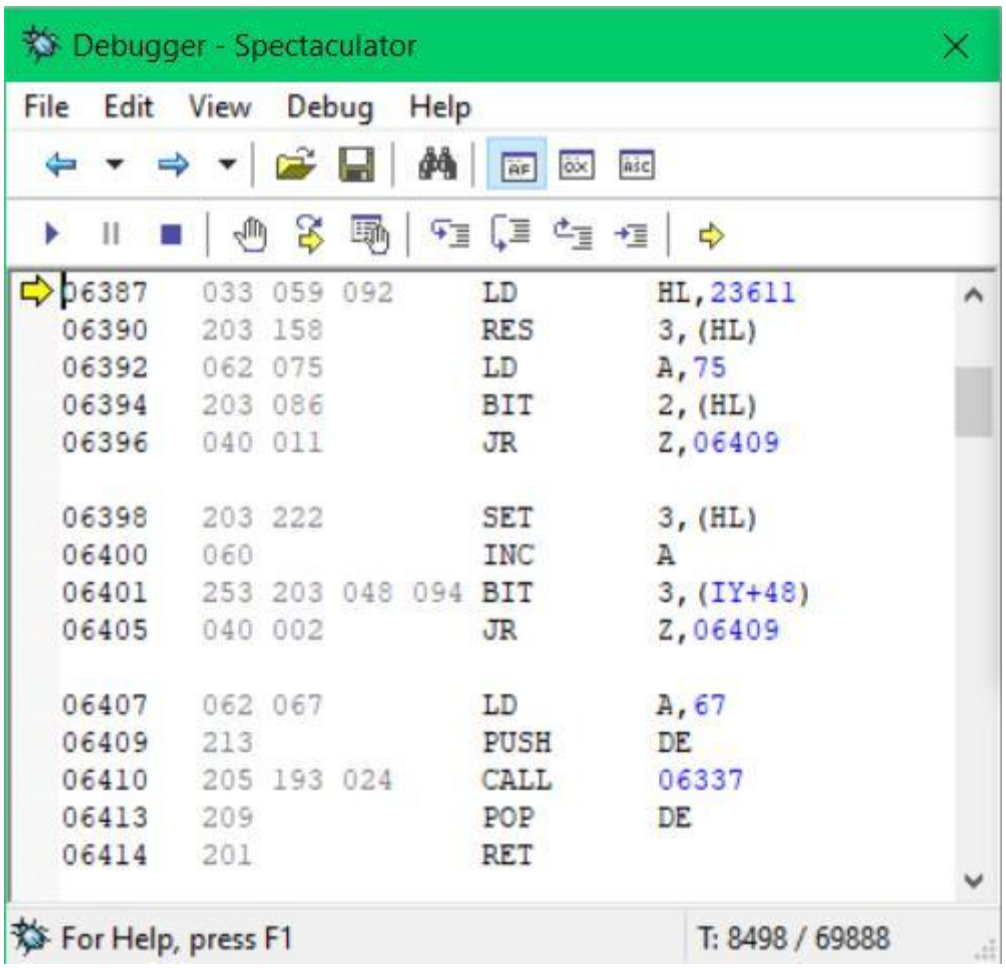


Рис. 81. Снятие точки прерывания с требуемого адреса.

Шарик-ловушка пропал, и Жёлтая Стрелочка опять свободна.

Снова нажмите на число-ссылку 23611 в команде. Переместившись на этот адрес, введите в ячейку значение 4.

Готово. Нажмите «*Trace (F5)*»:

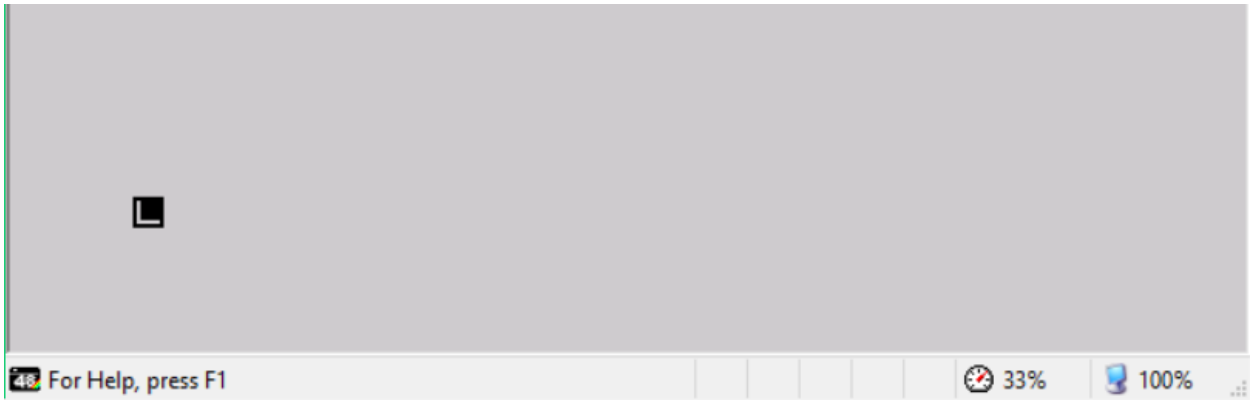


Рис. 82. Вывод курсора L искусственным методом с прерыванием. Фрагмент нижней части экрана.

Появился чистейший курсор без манипуляций в BASIC. Понажимайте буквы «qwerty» на реальной клавиатуре:

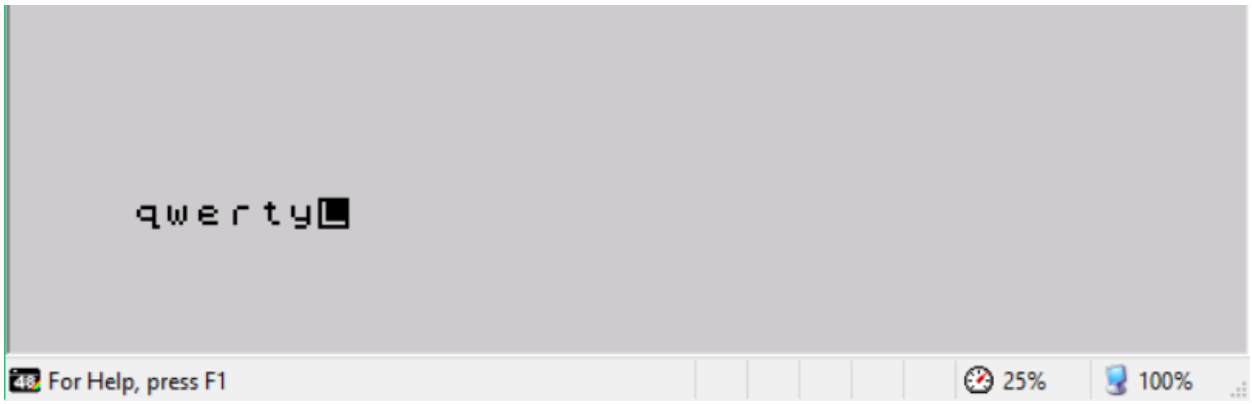


Рис. 83. Проверка работоспособности выведенного курсора L. Фрагмент нижней части экрана.

Снова придётся придумывать мнемонику новым методикам и выводить алгоритм программы. Эти действия я представляю следующим образом.

Вывод курсора L на чистый компьютер с заставкой:

```
Debugger
Dec
Go To 6387
Add Breakpoint 6387
26611 ← 32
Trace
Remove Breakpoint 6387
23611 ← 4
Trace
```

Можно опробовать сгенерировать курсор C на имеющийся текст. Для этого по адресу `FLAGS2 (23658)` нужно включить бит №3, поставив значение 8, после чего выставить 9 по адресу `TV_FLAG (23612)`.

Установка курсора C на L к имеющейся строке:

```
Debugger
Dec
Go To 23612
23612 ← 9
23658 ← 8
Trace
```

Введите и запустите эту простую программу и вот уже на экране после «qwerty» мигает курсор C:

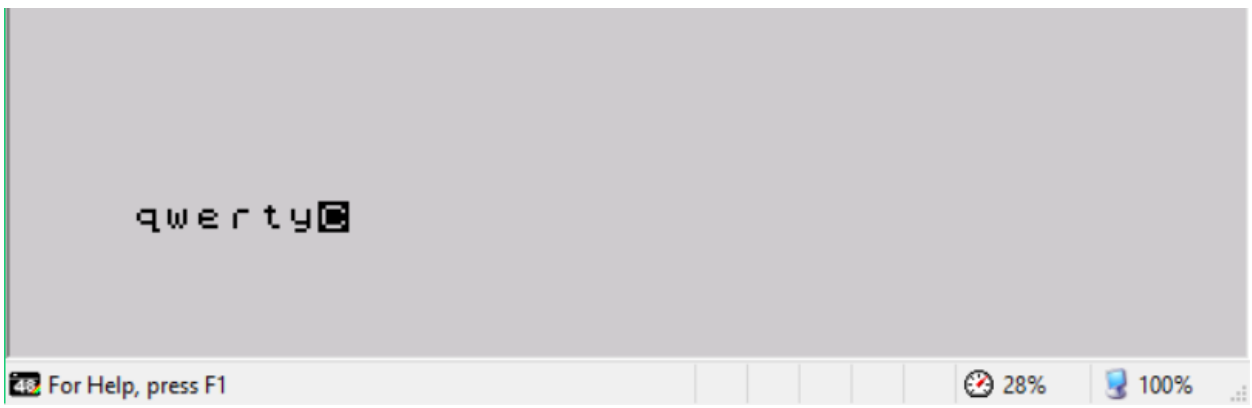


Рис. 84. Установка курсора C локальным обновлением к имеющемуся тексту. Нижний фрагмент экрана.

Теперь для теста добавьте «asdfg»:

Рис. 85. Проверка работоспособности сгенерированного курсора C. Фрагмент нижней части экрана.

Предлагаю вернуть курсор K. Если на чистый компьютер для установки достаточно всего одной строчки, то чтобы поставить курсор принудительно после текста, придётся делать алгоритм, аналогичный установке курсора L. Разница будет в том, что после остановки Стрелочки, в ячейку `FLAGS (23611)` нужно ввести 0, вместо имеющегося там значения, выключив все тумблера.

Установка курсора K вместо C или L к имеющейся строке:

```
Debugger
Dec
Go To 6387
Add Breakpoint 6387
26612 ← 9
Trace
Remove Breakpoint 6387
23611 ← 0
Trace
```

Вводите и запускайте программу:

Рис. 86. Искусственный вывод курсора K после текста. Фрагмент нижней части экрана.

Позади текста появился курсор K. Нажмите клавишу «p»:

Рис. 87. Вывод команды синтезированным курсором K. Фрагмент нижней части экрана.

На экран выскочила команда PRINT, и курсор снова стал C.

Осталось проработать вариант установки курсора C на чистый компьютер. И такой имеется. Нажмите сброс, а затем наберите и введите следующую программу:

```
Debugger
Dec
Go To 6387
Add Breakpoint 6387
26611 ← 32
23658 ← 8
Trace
Remove Breakpoint 6387
23611 ← 4
Trace
```

Рис. 88. Установка курсора C искусственным способом на чистый компьютер. Фрагмент низа экрана.

Нажмите «zxcvb»:

Рис. 89. Проверка печати синтезированным курсором C. Фрагмент нижней части экрана.

А теперь попробуйте вернуть курсор L:

Рис. 90. Вывод курсора L искусственным способом. Фрагмент нижней части экрана.

Нажмите что-нибудь для пробы:

Рис. 91. Проверка работоспособности искусственного курсора L. Фрагмент нижней части экрана.

В папке к этой главе вы найдёте еще несколько примеров переключения курсоров. На этом моменте затянувшееся ознакомление с курсорами, заканчивается. Это даже не введение получилось, а целое «*Приложение С*» с первой его частью, которое растянулось на несколько глав:

Приложение С (часть 1)

Описание микрокомпьютера ZX SPECTRUM

Клавиатура.

Каждая клавиша клавиатуры компьютера ZX SPECTRUM имеет многофункциональное назначение и позволяет вводить как отдельные символы, так и целые слова. Действие, производимое клавишей определяется частично переключающими клавишами (CAPS SHIFT и SYMBOL SHIFT) а частью режимом, в котором находится компьютер.

Режим отображается курсором, мерцающей буквой, указывающей позицию, в которую будет вводиться очередной символ с клавиатуры. Возможны следующие режимы:

K- (для ключевых слов) KEYBORDS.

Этот режим автоматически сменяет режим L, если компьютер переходит в ожидание ввода команды или строки программы. Это может быть либо в начале строки, либо после THEN, либо после ": ". И если не было нажатия переключающих клавиш, то нажатие любой клавиши будет интерпретироваться как ключевое слово (написанное на клавише) или цифра.

L- (для букв) LETTER.

Осиовной режим для компьютера. Если не было переключения регистров, то клавиша интерпретируется как основной символ, нанесенный на эту клавишу. Для обоих режимов L и K при одновременном нажатии с клавишей клавиши SYMBOL SHIFT, клавиша будет интерпретироваться как вспомогательный символ, а при нажатии CAPS SHIFT с цифрой, клавиша будет интерпретироваться как управляющая функция, написанная на белом поле клавиши. Нажатие CAPS SHIFT с любой из клавиш не вызывает ключевого слова в режимах K и L.

C- (для заглавных букв) CAPITAL.

Режим представляет собой вариант режима L, в котором используются заглавные буквы. CAPS LOCK используется для перехода из режима L в C и обратно.

E- (для расширения) EXTEND.

Используется для получения дальнейших символов, главным обра-

Рис. 92. Фрагмент приложения из самоучителя по BASIC.

Нажав Reset очередной раз, можно приступать к следующей главе.

Глава 9
Повседневная жизнь волшебного города BASIC

Из прошлых глав, вы знаете, что попав в программу ввода и редактирования BASIC строки EDITOR, Стрелочка проваливается в нижние этажи хитрой сети подпрограмм, и упирается в барьер по адресу 4276. Наступив на клеточку с командой «RET Z», Стрелочку выкидывает на пару уровней вверх. Оттуда она снова идёт к этому месту и снова её подбрасывает вверх…. Так она и ходит кругами по подземным лабиринтам комплекса программ ожидания нажатой клавиши.

После нажатия клавиши происходит проход на печать символа и возврат в редактор для добавления следующего. Благодаря прошлой главе, вы нашли Стрелочку, добыли секретную информацию и сами погуляли по подземному царству, но выманить на поверхность так и не смогли.

Чтобы мотивировать Стрелочку на прогулку по цветущему городу, требуется нажать специальное заклинание. И тогда она выйдет из подземелья, чтобы исследовать дальнейший мир BASIC. И этим тайным заклинанием станем магическое число «13», а по совместительству код клавиши ENTER. По нажатию ENTER, с набранным текстом или без, происходит возврат на поверхность в MAIN EXECUTION по адресу 4788.

Определив, что нажаты управляющие коды в интервале от 7 до 13, Стрелочка заходит в подпрограмму ED KEYS (3986) и по таблице размещения начинает вычислять и формировать адрес многоликого выхода, который осуществится с помощью команды «RET»:

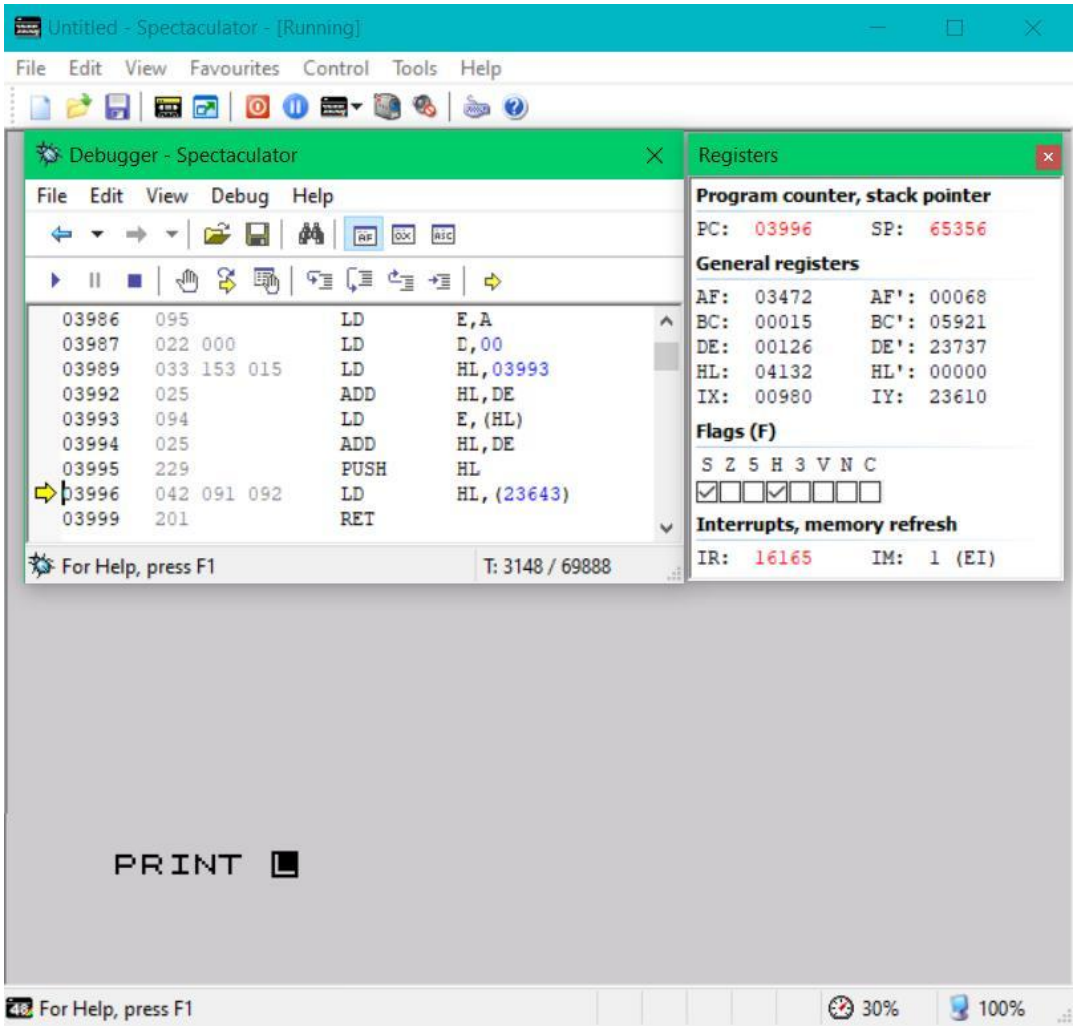


Рис. 93. Подпрограмма ED KEYS (3986).

По коду ENTER, этим адресом перехода станет подпрограмма ENTER EDITING (4142):

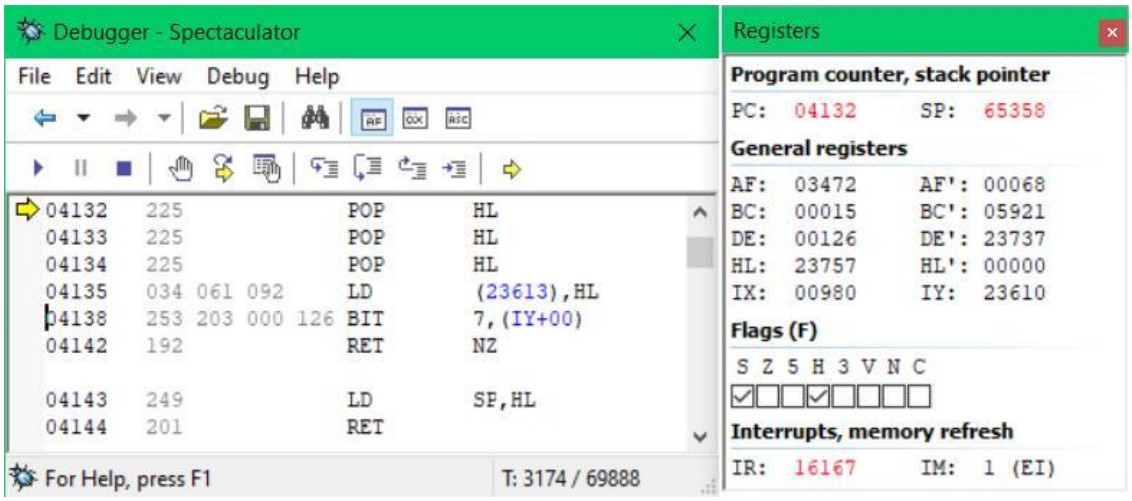


Рис. 94. На пути к выходу. Подпрограмма ENTER EDITING (4132).

Стрелочка на пути к выходу из подземелья. Откуда-то сверху пробивается яркий дневной свет. Компенсировав образовавшийся перекос, восстанавливается значение переменной `ERR_SP (23613)` до значения 65364. И вот он долгожданный выход на поверхность:

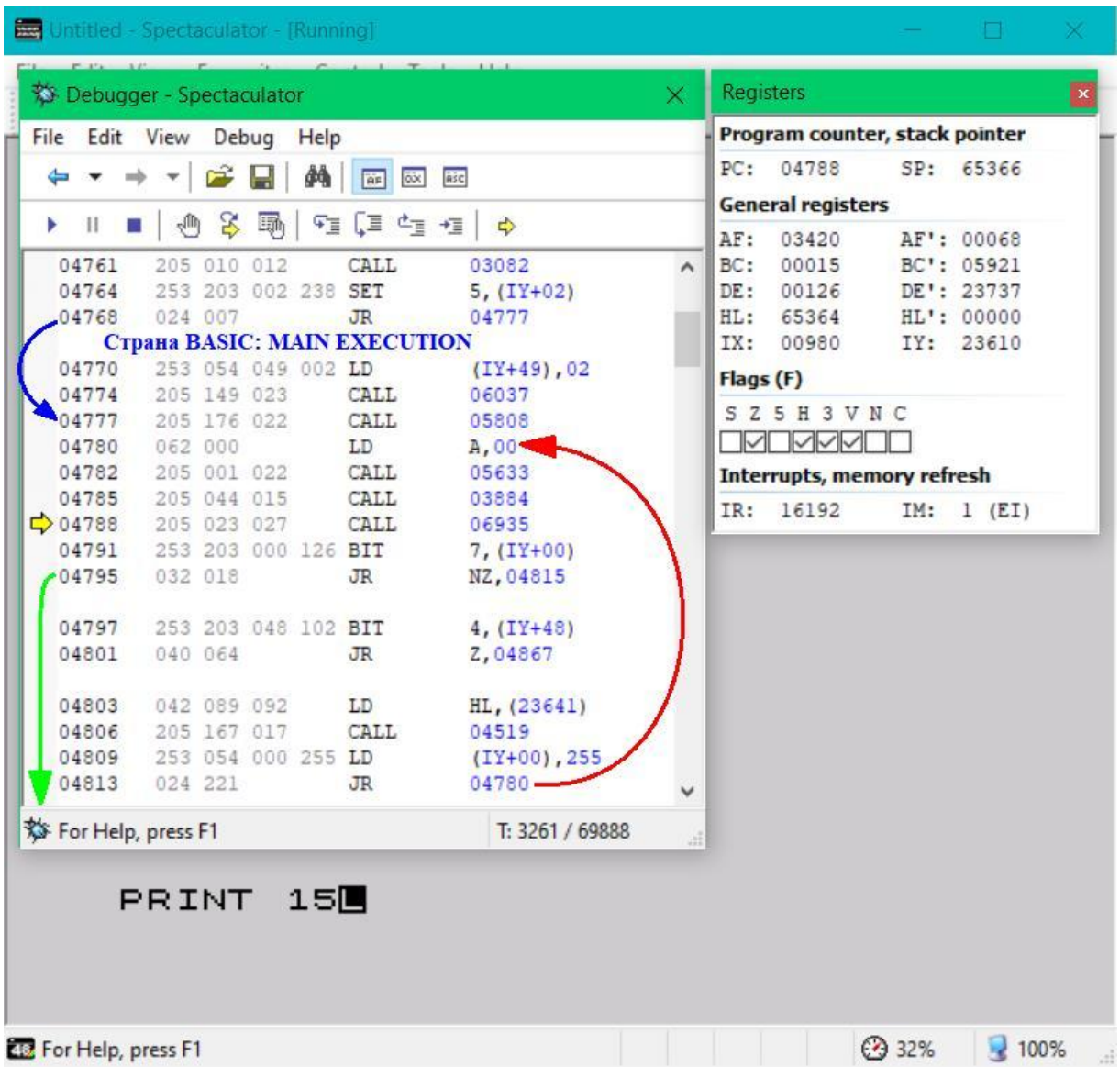


Рис. 95. Выход на поверхность в MAIN-2 по адресу 4788 после нажатия ENTER.

Стрелочка выныривает из запутанных подземных тоннелей программы EDITOR, оставив в области редактора E_LINE (23641) сырой текст программной строки.

Следом вызывается подпрограмма LINE SCAN по адресу 6935. Она преобразует строку с символами в исполняемую BASIC последовательность. Кроме того, как ОТК производит входной контроль строки на грубые ошибки (хаотичный набор символов, лишние параметры команды и т. д.). После проверки делает соответствующую пометку в одноячеечной переменной ERR_NR (23610).

После возвращения в главную программу (4791) проверяется рекомендация от инспектирующей программы. Если в ячейке ERR_NR (23610) появился 7-й бит, значит, госприемка (LINE_SCAN) одобрила качество вводимой строки, поставила высшую оценку (255) и выдала предварительное разрешение на выполнение строки. Следующей командой происходит переход на блок сортировочной базы MAIN-3 по адресу 4815.

Ну а если напечатанная строка оказалась белибердой, то инспектор выставляет низкую числовую оценку (11), в которой последнего 7-го бита не существует, и выдаёт распоряжение на устранение брака. Перед устранением дефекта, неисправной строке предстоит пройти обратное преобразование в символьный вид.

В переменной E_LINE (23641) узнаётся адрес начала текущей забракованной строки. Следом подпрограмма REMOVE-FP (4519) проводит процедуру обратной декомпрессии рыхлых чисел в простой символьный вид.

Авансом ячейке ERR_NR (23610) присваивается значение исправной строки (255), а следом происходит возврат к началу блока MAIN-2 по адресу 4780 для подготовки к повторному редактированию строки. Установив вывод в нижние строки, повторно вызывается редактор EDITOR (3484). Теперь уже в месте ошибки мигает знак вопроса  .

После исправления ошибок, строка пройдёт повторный контроль, и если недочёты исправлены, произойдёт переход по адресу 4815 в сортировочный пункт под названием MAIN-3:

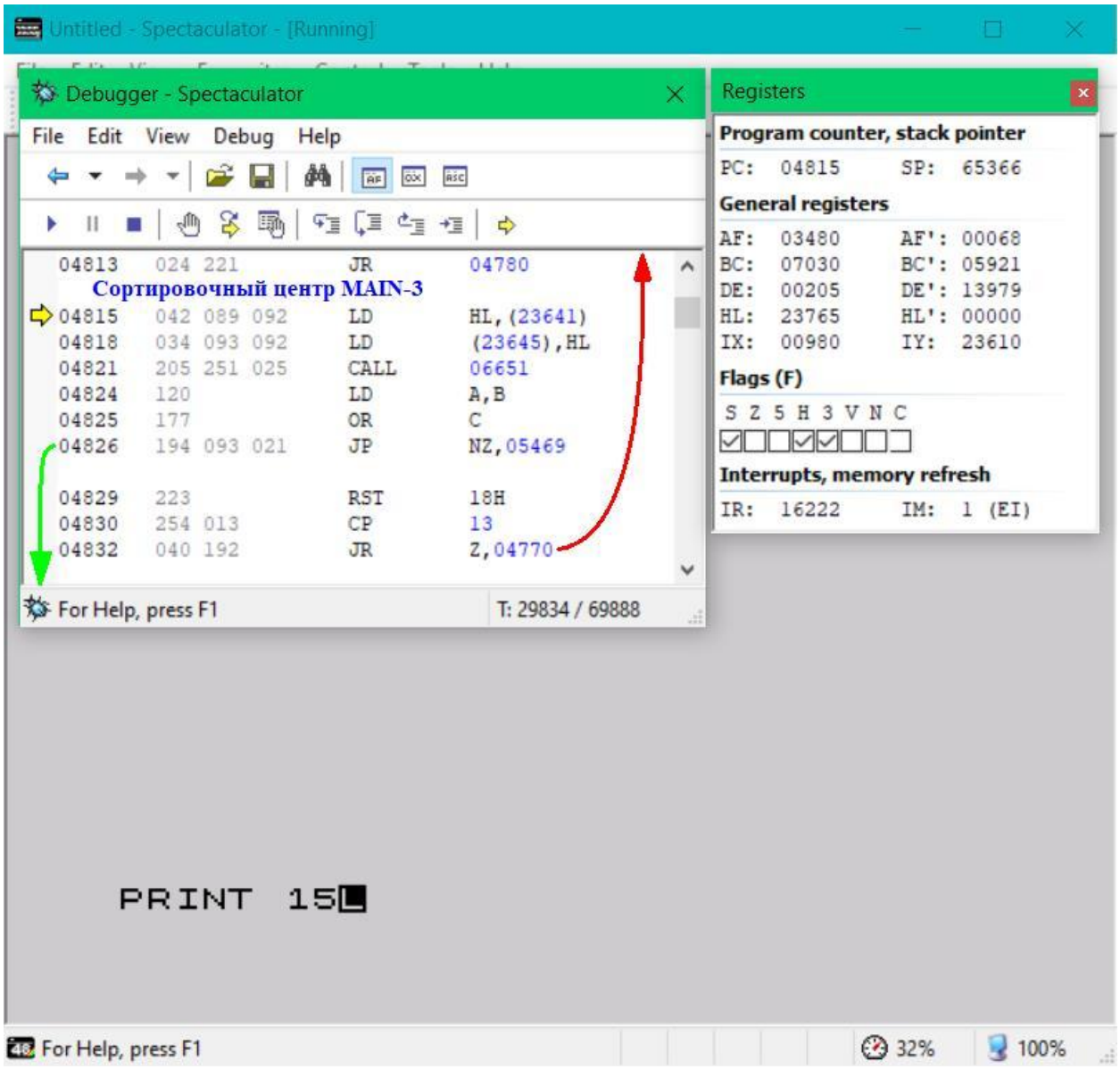


Рис. 96. Переход в сортировочный центр MAIN-3 после предварительной проверки вводимой строки.

Строка прошла предварительную проверку и была направлена в сортировочный цех, он же точка `MAIN-3 (4815)`. Он сортирует три типа поступивших массивов: строка с номером, строка без номера и холостой `ENTER`.

Взяв адрес начала проверенной строки из `E_LINE (23641)`, оно переписывается в `CH_ADD (23645)`, который является внутренним посимвольным курсором. Таким образом, задаётся первый обрабатываемый фрагмент BASIC строки. С помощью подпрограммы `E_LINE NO (6651)` производится попытка определить её номер.

Если вернулось значение больше 0, значит, строка с номером и нужно перескочить на программу `MAIN ADD (5469)` чтобы положить её на склад готовой продукции. Если получилось «0», значит, строка не имела номера и нужно двигаться вниз для подготовки к выполнению. Обратите внимание, строка 0 трактуется системой как не имеющая номера, а не с номером 0. Убрав это ограничение, любая введённая безномерная строка примет номер 0.

Если строка оказалась из голого «`ENTER`'а» (красная стрелка), то произойдёт возврат в самое начало-начал, в точку `MAIN EXECUTION` по адресу 4770.

Выделив две строки под буфер редактирования в нижней части экрана, вызывается подпрограмма обновления экрана с текстом программы `AUTO LIST (6037)`.

После перерисовки текста программы на экране вызывается подпрограмма оптимизации массивов WORKSP (23649), STKBOT (23651) и STKEND (23653). После работы со строкой эти области разрослись, а подпрограмма SET MIN (5808) ставит границы этих временных областей на место, после чего они снова указывают на одну и ту же точку сразу за маркером «128», который находится после E_LINE (23641). Всё остальное, оказавшееся за пределами границ, становится мусором, который затирается вспомогательными данными от работы следующей строки. Это и будет тот самый визуальный эффект освежения экрана по нажатию клавиши ENTER.

В итоге под конец сортировки остаётся строка без номера, которую нужно выполнить и Стрелочка перешагивает на следующую отметку с номером 4834, где начинается подпрограмма подготовки и выполнения вводимой безномерной строки:

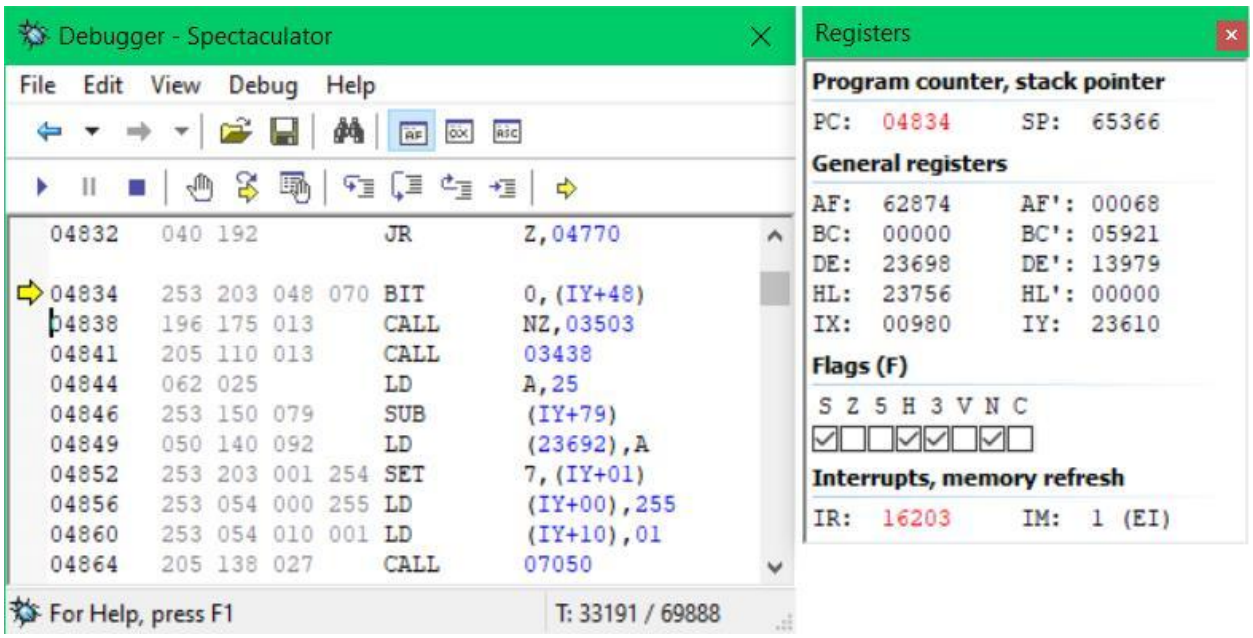


Рис. 97. Подготовка к выполнению BASIC строки без номера.

Первым делом проверяется состояние переменной FLAG2 (23658). Если бит-переключатель №0 активирован, то подпрограммой CL_ALL (3503) очищается основной экран. Следом стирается изображение взятой в работу выполняемой строки, путём очистки нижней части экрана подпрограммой CLS-LOWER (3438).

А дальше идёт подготовка к выполнению набора команд строки. Включается бит №7 FLAGS (23611), сигнализируя, что программная строка скоро будет запущена. В ERR_NR (23610) заносится код проверенной и допущенной к выполнению строки (255). В NSPPC (23620) устанавливается номер оператора 1, с которого нужно начинать выполнение BASIC программы.

После всех приготовлений строка запускается многослойной подпрограммой PROG-RUN (7050). По выходу из этой подпрограммы, в MAIN-4, с адреса 4867 подводятся итоги выполнения строки:

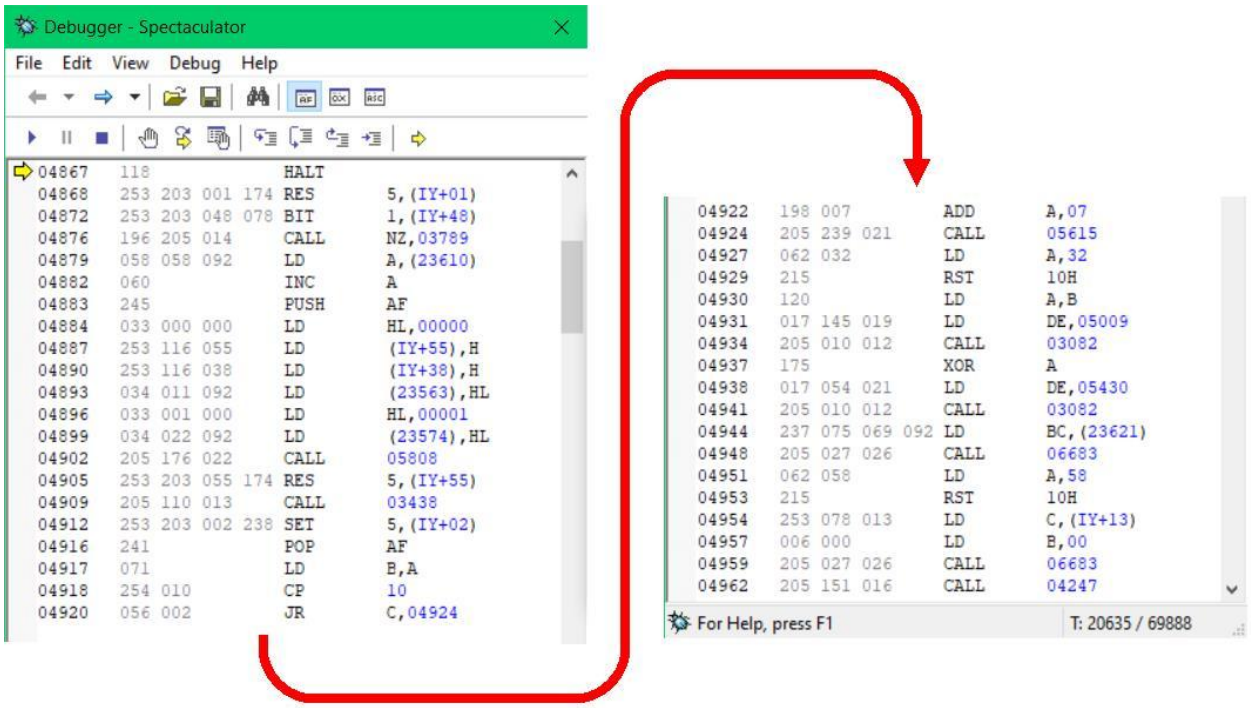


Рис. 98. MAIN-4. Итоги выполнения BASIC строки.

По итогу выполнения строки, `PROG RUN` в `ERR NR (23610)` возвращает расширенное значение оценки выполнения программы. В нём уже содержится конкретный код ошибки, увеличенный на единицу.

Отключается тумблер №5 переменной `FLAGS (23611)`, чтобы никто не повредил результат выполненной работы. Очищается буфер принтера. Из `ERR_NR (23610)` берётся уточнённая оценка результата выполнения программы, прибавляется единичка и получается номер стандартного сообщения об ошибке. Их можно посмотреть в приложении любой книги.

Сохранив его чтобы не забыть, в переменных производится чистка и уборка после выполненной команды. Восстанавливаются затронутые переменные, и ужимается до минимума временная рабочая область BASIC. Следом очищается нижняя часть экрана для вывода сообщения и включением бита №5 `TV_FLAG (23612)` устанавливается режим заморозки изображения, чтобы при выходе в редактор оно осталось висеть на экране.

Вспомнив номер выбранного сообщения, начинается анализ полученного кода с последующей его сортировкой. Если сообщение из первого блока и имеет номер меньше 10, то можно сразу перейти к его печати.

Сообщения с номером выше 10, получают буквенный код (10=A, 11=B, 12=C и т. д). Для них добавляется смещение в 7 единиц, чтобы из этого номера сформировался буквенный код ошибки.

Печатается полученный код ошибки, затем пробел. Указав адрес начала размещения текста, вспоминается код ошибки и программой `PO MSG (3082)` печатается после очередного пробела с запятой. Следом из переменных `PPC (23621)` и `SUBPPC (23623)` извлекаются значения и через двоеточие выводятся на экран с помощью программы `OUT NUM1 (6683)`. В конечном итоге получается комплексное сообщение такого плана:

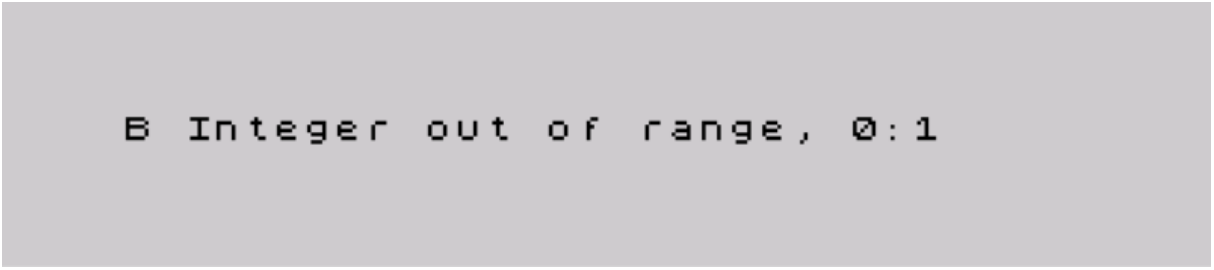


Рис. 99. Пример сформированного сообщения подпрограммой MAIN-5.

После этого с помощью CLEAR SP (4247) очищается рабочая область и устанавливается *режим* курсора K. Снова из ERR_NR (23610) берётся код сообщения, анализируется какого он характера и по результатам изменяются соответствующие системные переменные.

Сортировка начинается с успешного выполнения. Если это сообщение «0 OK», то, устанавливается курсор K. Осуществляется возврат по адресу 4764 (точка MAIN-2) в режим ожидания, чтобы оставить на экране результат выполненной программы. Цикл замыкается.

Если был нажат STOP, то конфигурируются соответствующие переменные для возобновления старта программы по команде CONTINUE. Для этого в OLDPPC (23662) и OSPPC (23664) переносятся номер строки и оператора, с которых нужно продолжить выполнение программы.

Дальше идёт проверка на неверное выполнение программы, типа «Variable not found» и прочих мелких неприятностей. Самой последней рассматривается ситуация, если нажата комбинация BREAK+CAPS SHIFT.

В зависимости от ситуации, записываются данные в переменные. Затем для любой ситуации устанавливается курсор K, заглушка «255» в переменную NSPPC (23620) и происходит возврат в 4764 (MAIN-2), оставив результат на экране:

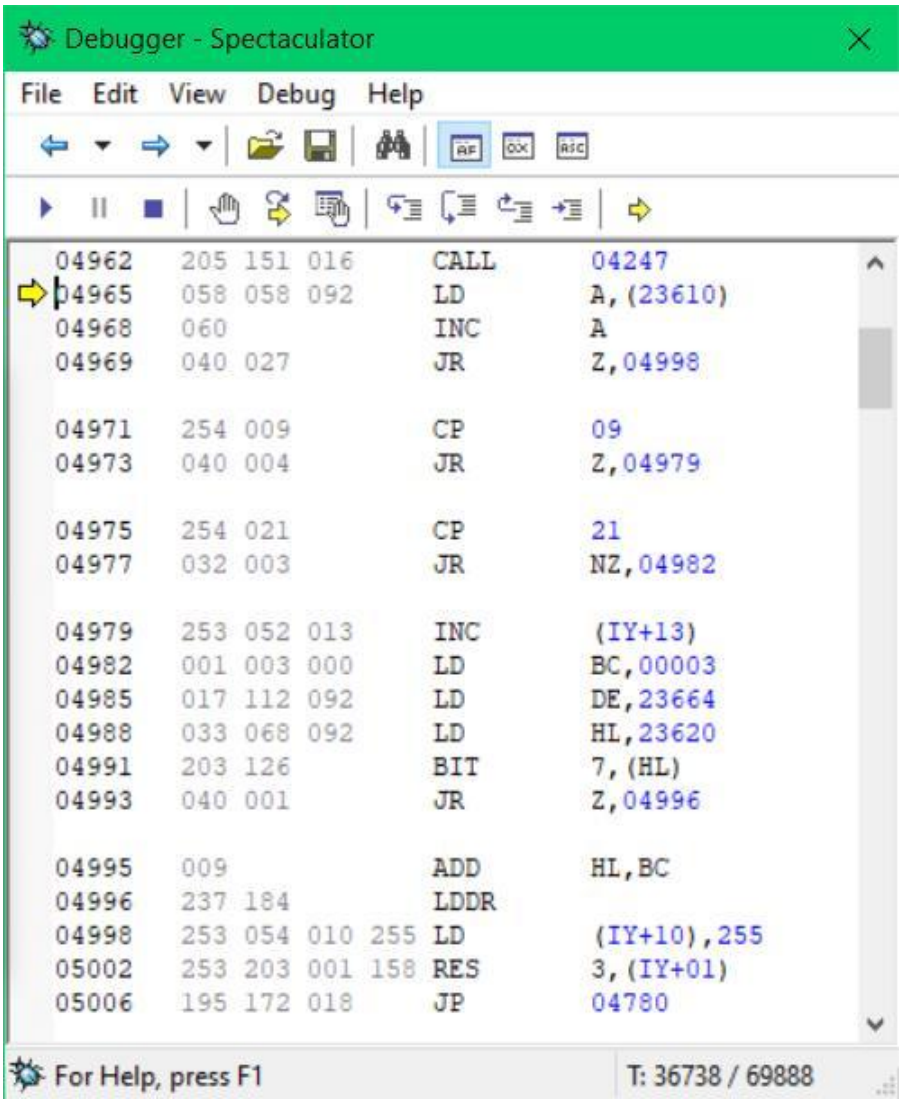


Рис. 100. Подготовка к возврату в начало цикла, в зависимости от характера сообщения.

С адреса 5009 начинается библиотека сообщений об ошибках. Заглавным стоит маркер «128». Фразы с текстом расположены подряд без разрыва. Поскольку буквенно-числовые символы не могут иметь код более 127, для экономии места в ПЗУ, маркер запаивается в конечную букву комплекта. Для этого к коду символа прибавляется «128» (или устанавливается 7-й бит). Например, в фразе «Variable not found» последняя буква «d», которая имеет код 100, смешана с маркером «128» в итоге общий код символа получился 228.

Задавая в «DE» адрес перед первым маркером, а в «A» количество переходов через следующие маркеры, можно выбрать нужное сообщение из комплекта:

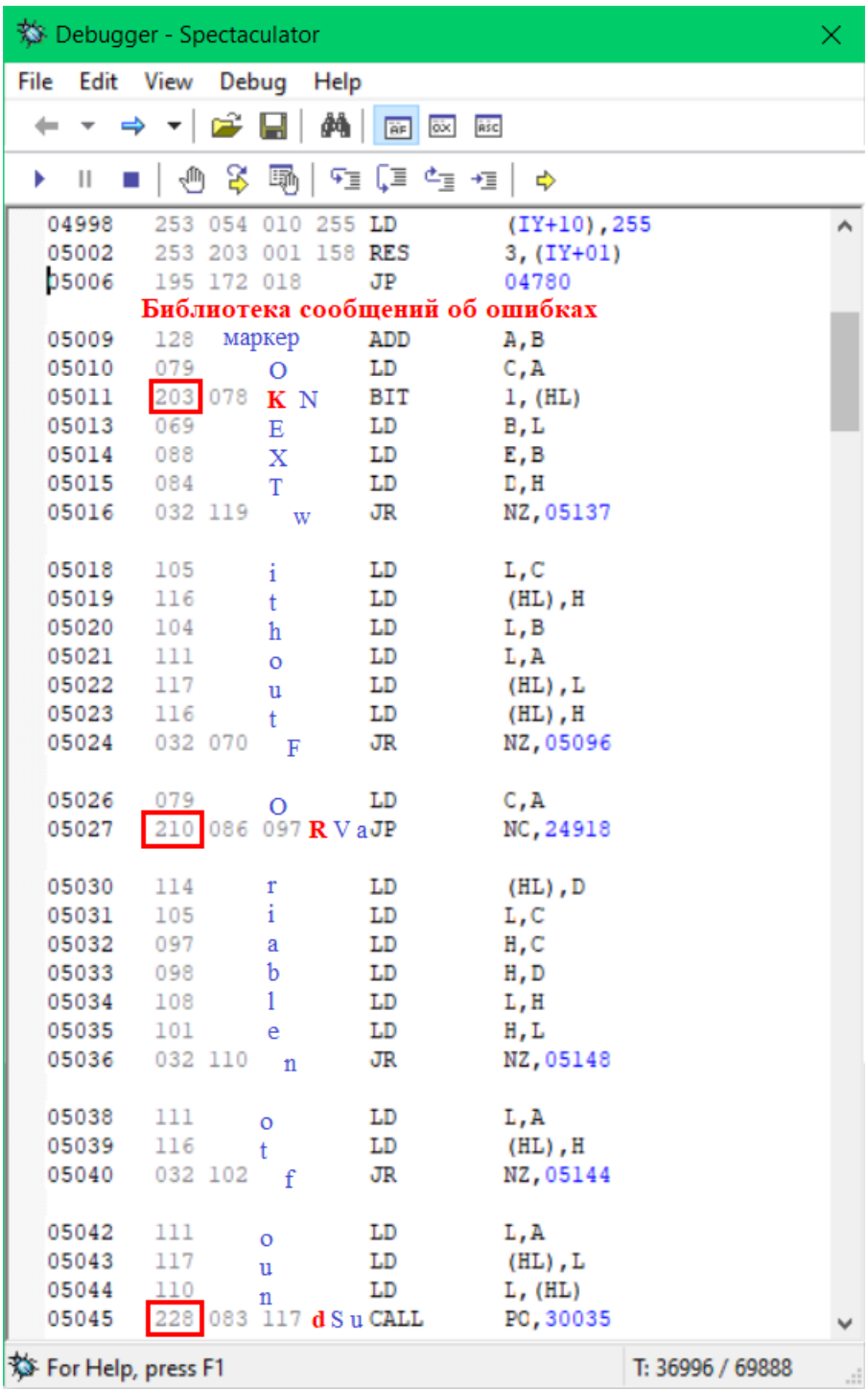


Рис. 101. Глобальная библиотека сообщений.

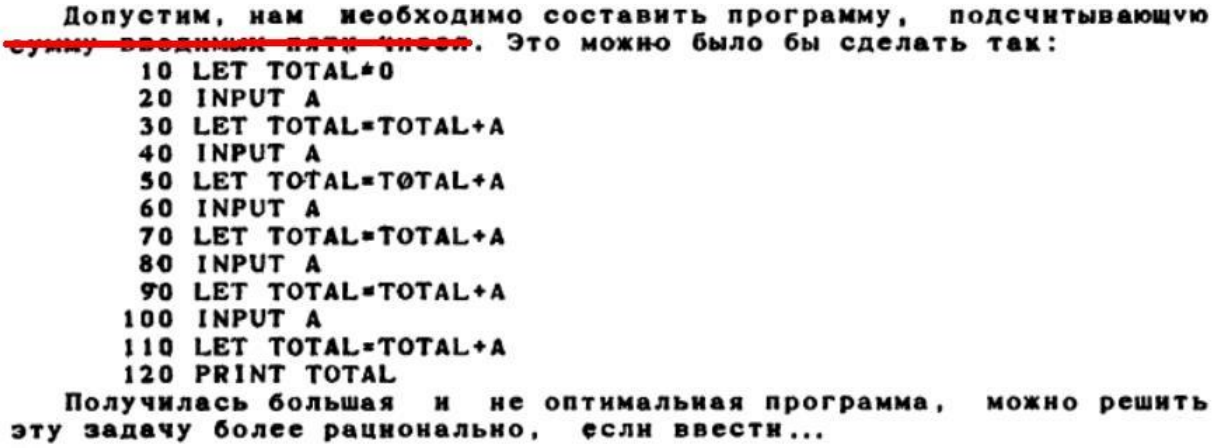
Вот такие повседневные события происходят в волшебном городе BASIC.

Глава 10
Ввод командной строки или ваш первый забористый ПРИНТ

Краткое содержание: ввод текстовой последовательности, ввод пятизначных чисел, свойства кавычек, ASCII вид, E_LINE (23641), WORKSP (23649), K_CUR (23643), синтез BASIC-строки, искусственный запуск

Ознакомившись с бытом и культурой волшебного города BASIC, можно попробовать синтезировать и выполнить простую BASIC строку.

В позапрошлой главе вы научились переключать курсоры, выводить управляющие комбинации, и печатать любые символы с командами. Такой метод удобен, когда требуется вывести 1-3 символа. Но,


Допустим, нам необходимо составить программу, подсчитывающую сумму вводимых пяти чисел. Это можно было бы сделать так:

```
 10 LET TOTAL=0
 20 INPUT A
 30 LET TOTAL=TOTAL+A
 40 INPUT A
 50 LET TOTAL=TOTAL+A
 60 INPUT A
 70 LET TOTAL=TOTAL+A
 80 INPUT A
 90 LET TOTAL=TOTAL+A
100 INPUT A
110 LET TOTAL=TOTAL+A
120 PRINT TOTAL
```

Получилась большая и не оптимальная программа, можно решить эту задачу более рационально, если ввести...

Рис. 102. Фрагмент из главы старого самоучителя по BASIC.

Перефразирую: допустим, вам необходимо выполнить программную строку в пару десятков символов. Можно после каждого символа из `LAST_K (23560)` включать бит №5 `FLAGS (23611)` и нажимать «*Trace*». Полностью согласен, что получится большая и неоптимальная программа.

Эту задачу можно решить более рационально, но вводить пока ничего не буду, ибо то чего имею, слишком забористое, поэтому вводить нужно осторожно, а не абы как.

Итак, предлагаю синтезировать и выполнить такую программную строку:

```
PRINT "В КОМНАТЕ АНТОНА ЗРЕЕТ ТРАВКА"
```

…Панельный девятиэтажный дом брежневской застройки. Утреннее солнышко освещает уютную детскую комнату. На внутреннем подоконнике стоят ящички с землёй, из которых пробивается зелёная травка. Мальчик заботливо поливает молоденькие всходы из красной пластмассовой леечки. Лениво пощипывая молоденькие росточки, на подоконнике сидит белый кот с большущими глазами. Жмурясь от весеннего солнышка, зверёк поглядывает вниз на двор и верхушки деревьев с набухающими почками…

Согласно таблице символов, команда `PRINT` имеет код «245» и на этом можно остановиться. При наличии смекалки всё остальное… забота отладчика.

Набираемая строка всегда начинается по адресу, указанному в комплекте ячеек `E_LINE (23641)` (*жарг. ельник, ёлка*). В режиме 48K на чистом компьютере это значение обычно 23756. Откройте «*Debugger*» по нужному адресу:

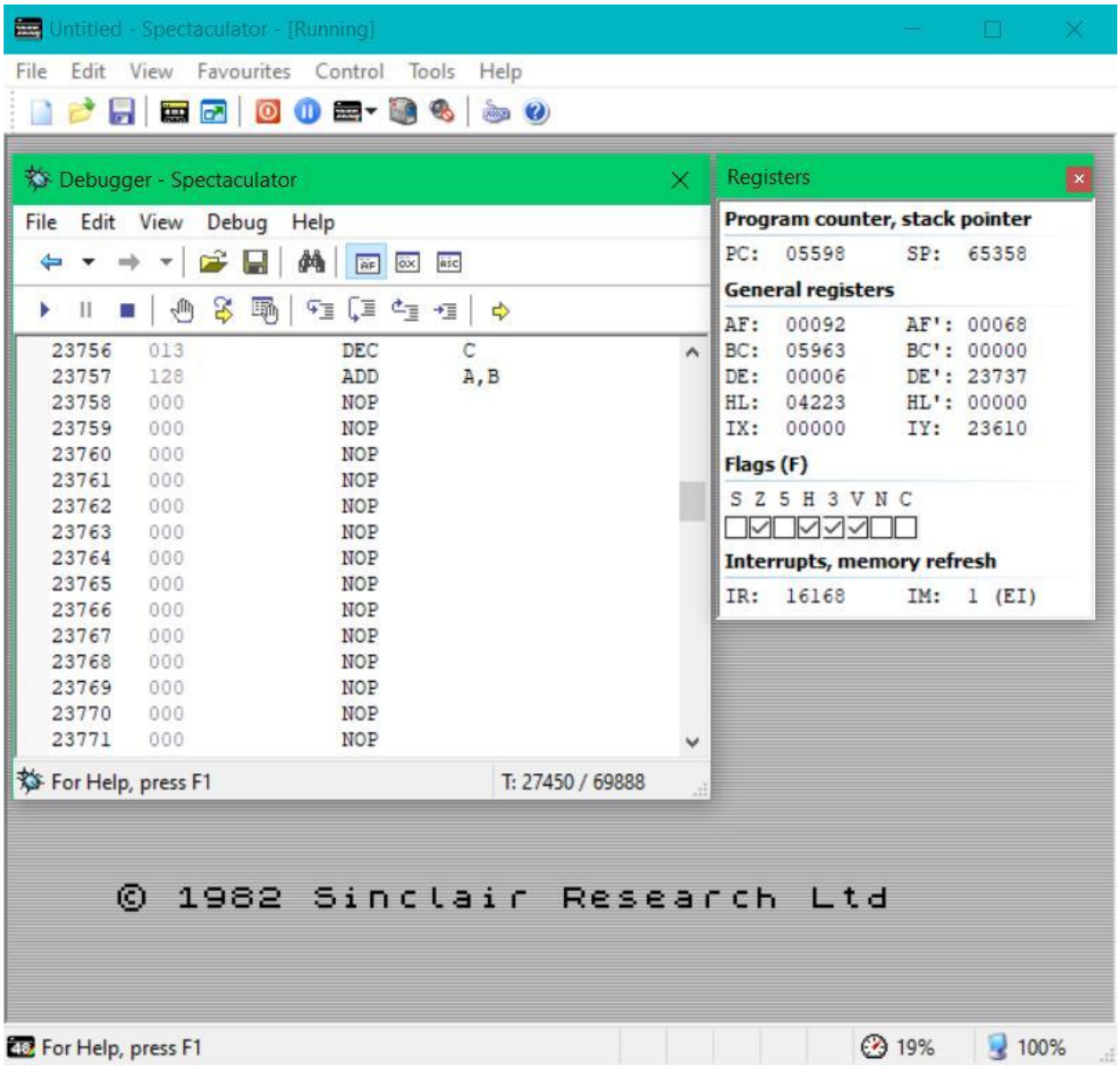


Рис. 103. Область вводимой строки.

Сейчас эта область пустует и состоит из кода ENTER (13) и заглушки «128». Введите в ячейку 23756 число 245:

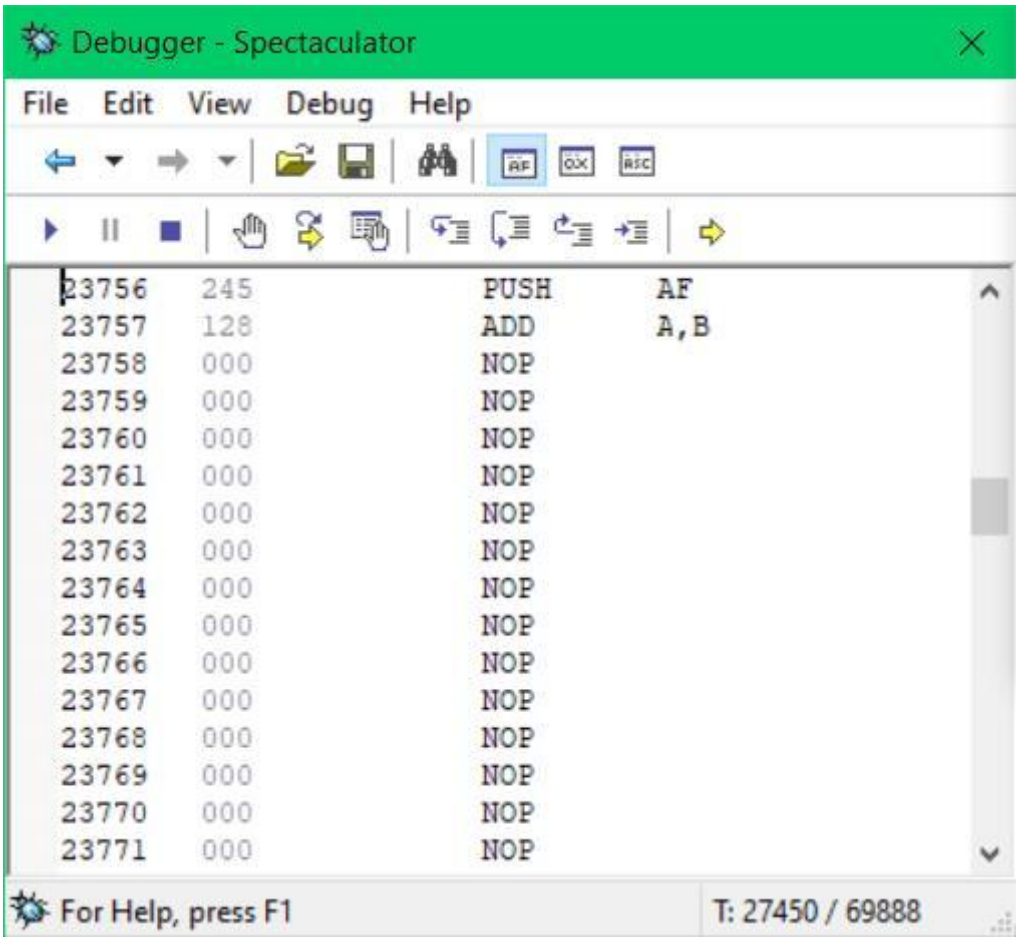


Рис. 104. Ввод кода команды PRINT.

А в ячейку 23757 попробуйте пакетом скормить… всю остальную строку. Но прежде, чем это начинать, небольшое и важное лирическое отступление.

Первым делом переключитесь на ***АНГЛИЙСКУЮ*** раскладку и запомните следующее:

Ни в коем случае не вводите в окошечки адресов памяти кириллицу! В Spectrum нет таких кодов символов, а англоязычные разработчики не предусмотрели защиту от случайного введения юникода. Любая русская буква вызывает сбой и зависание всего приложения с выводом информационного окна от Windows и потерей всех данных после его принудительного закрытия:

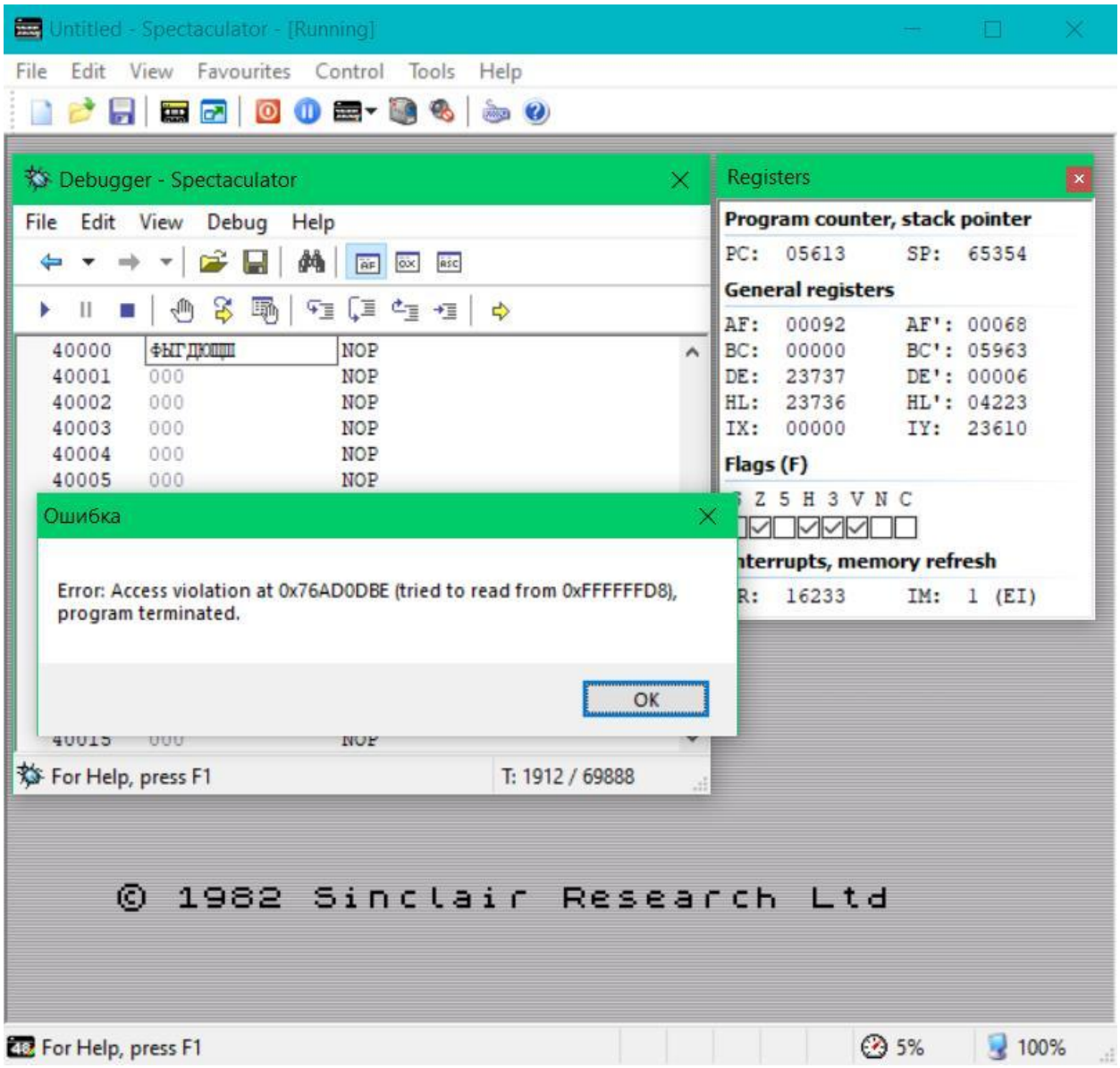


Рис. 105. Сбой «Access violation at…» при попытке ввода русских букв.

Вообще, если вы не хотите внезапного вылета программы с потерей набранных данных, забудьте не только про русский, но и другие языки. В отладчике работайте только с однобайтовыми ЛАТИНСКИМИ БУКВАМИ и знаками препинания, имеющимися в наборе ZX-SPECTRUM.

Тогда может возникнуть вопрос: как ввести рекомендуемую фразу? Очень просто. Обратите внимание, текст «В КОМНАТЕ АНТОНА ЗРЕЕТ ТРАВКА» написан заглавными английскими буквами.

Теперь по поводу кавычек. Отладчик принимает только стандартные ровные кавычки " с однобайтным кодом 34. Никаких вот этих «» юникодовских двухбайтных (C2h ABh и C2h BBh), а уж тем более трёхбайтных “” (E2h 80h 9Ch и E2h 80h 9Dh).

Если всё понятно, можно приступать к делу. Введите, а лучше скопируйте нижеследующую строчку в ячейку с адресом 23757.

И снова внимание! Это не ошибка, впереди предложения ДВЕ ОТКРЫВАЮЩИЕ КАВЫЧКИ, а сзади только одна:

```
""В КОМНАТЕ АНТОНА ЗРЕЕТ ТРАВКА"
```

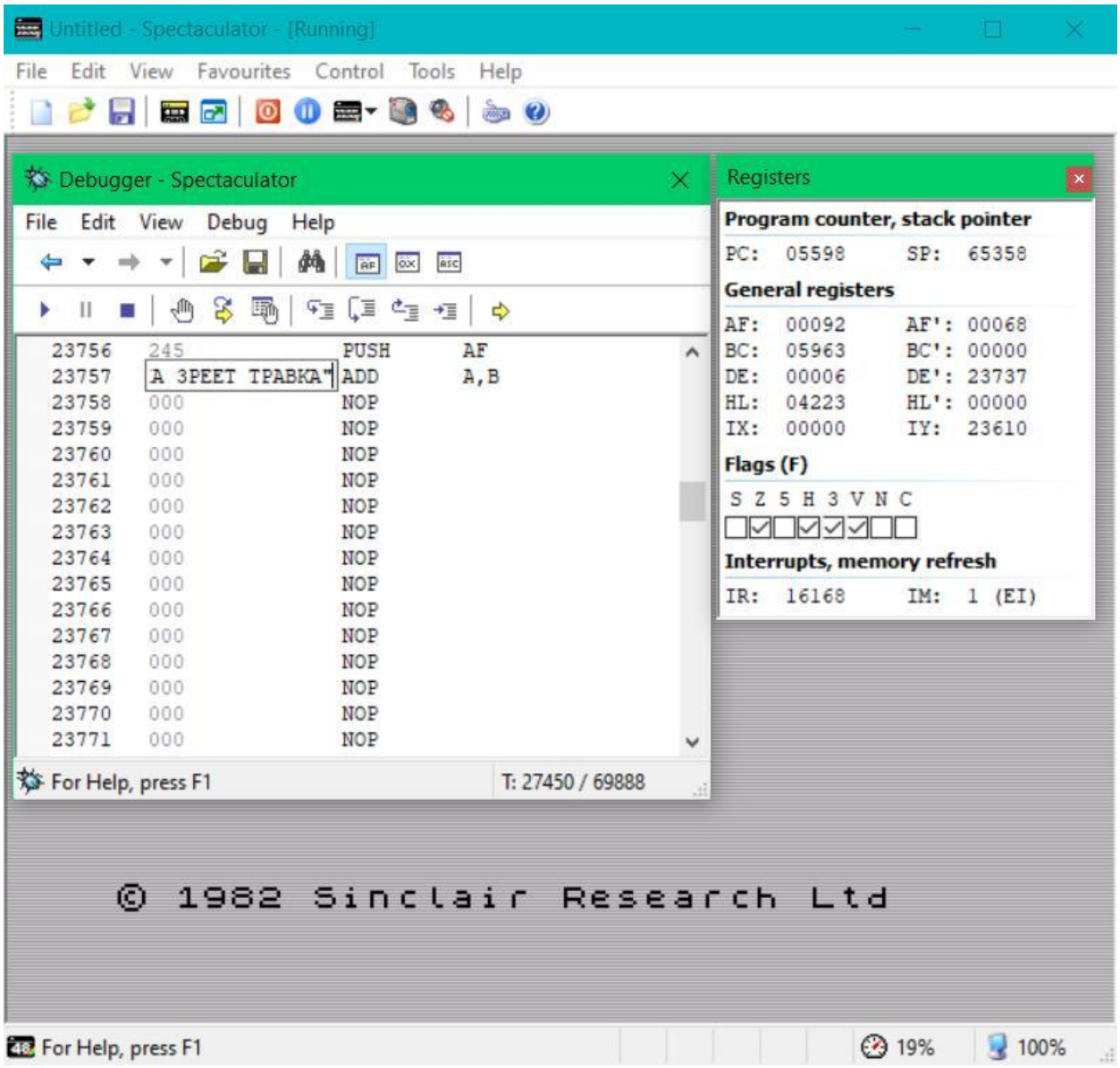


Рис. 106. Ввод длинной последовательности символов с кавычками в одну ячейку.

Вводите строку:

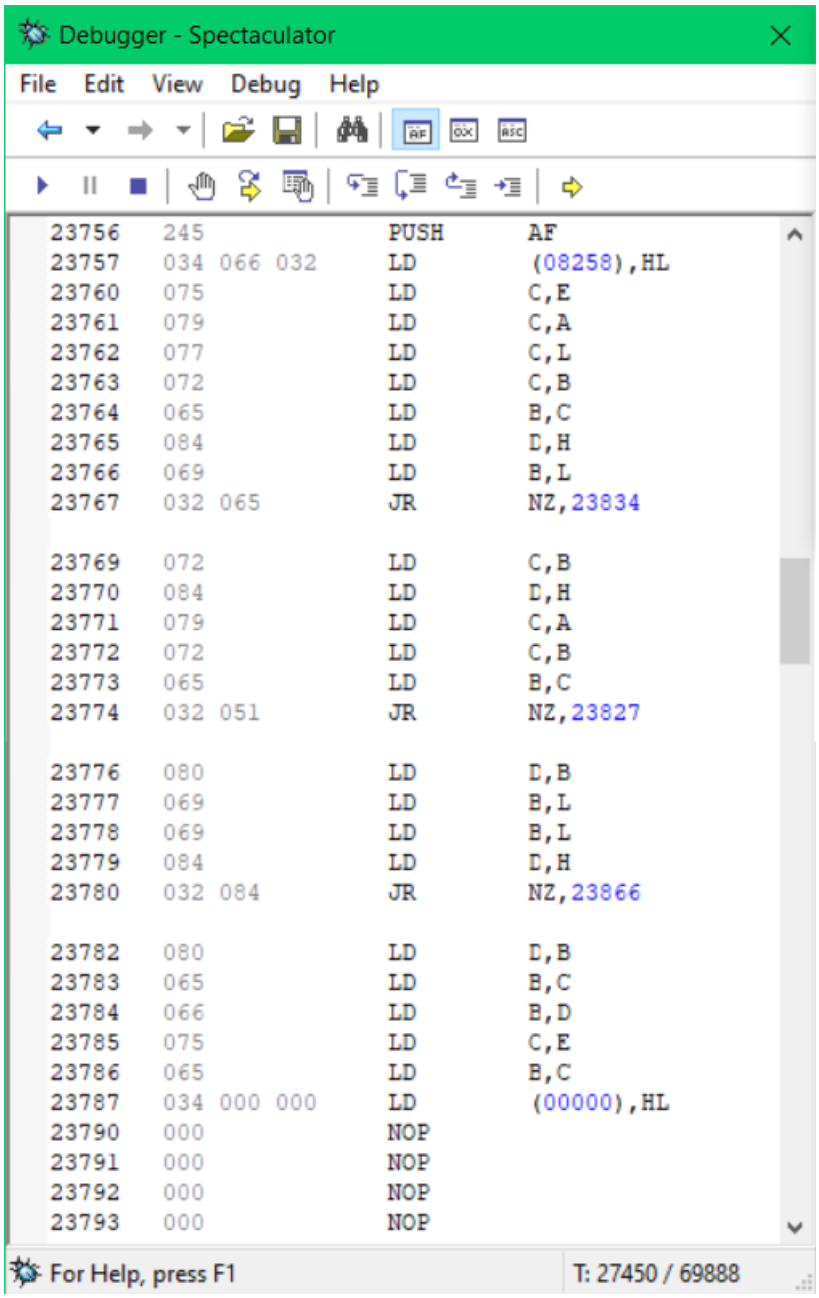


Рис. 107. Преобразование цепочки символьных данных в памяти.

Буквы в предложении автоматически перевелись в коды и встали друг за другом в нужном порядке в нижеследующие ячейки, несмотря на то, что вы всё вводили в одну. Зеците, данное свойство отладчика в разы упростило работу. Не нужно ничего переводить в числовой формат и поштучно вводить в ячейки.

Обратите внимание, подобно BASIC команде `VAL`, первую кавычку съел отладчик, а вторая уже записалась в память. Следом за ней расположилась цепочка текста. Это правило тоже следует запомнить:

При вводе символьной последовательности, начинающейся с кавычек, её требуется защитить ещё одной, которая при вводе исчезнет.

Приёмы схожи с программированием в командной строке (по этой теме рекомендую книгу *Яркий мир «Волшебного DOS»*). Вообще в одну ячейку можно ввести последовательность, длиной до 40 буквенно-числовых символов. Всё что свыше, будет автоматически обрезано.

Для самоконтроля проверьте правильность автопреобразования. Найдите кнопочку «*ASCII (Ctrl+H)*»:

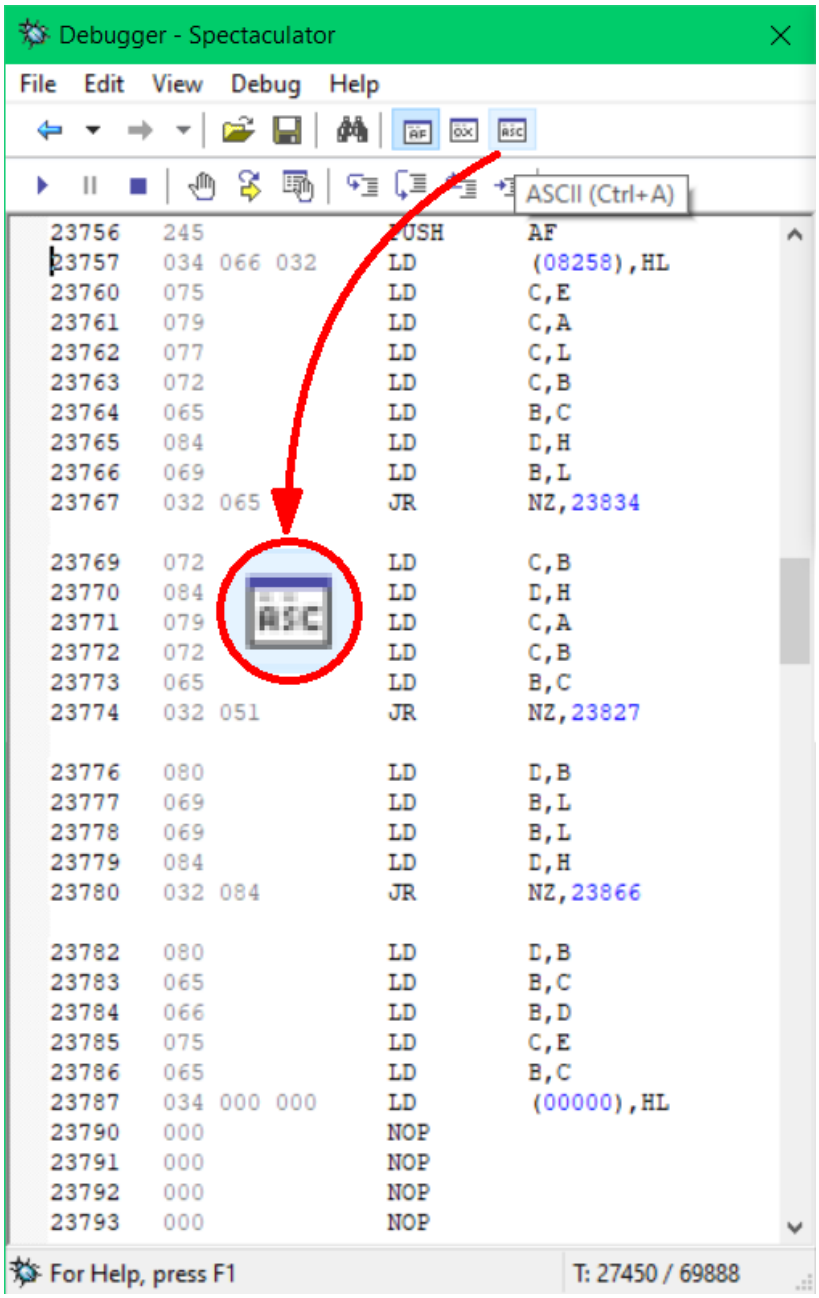


Рис. 108. Кнопочка преобразования данных ASCII.

Нажмите её, и все значения ячеек превратятся в символьные:

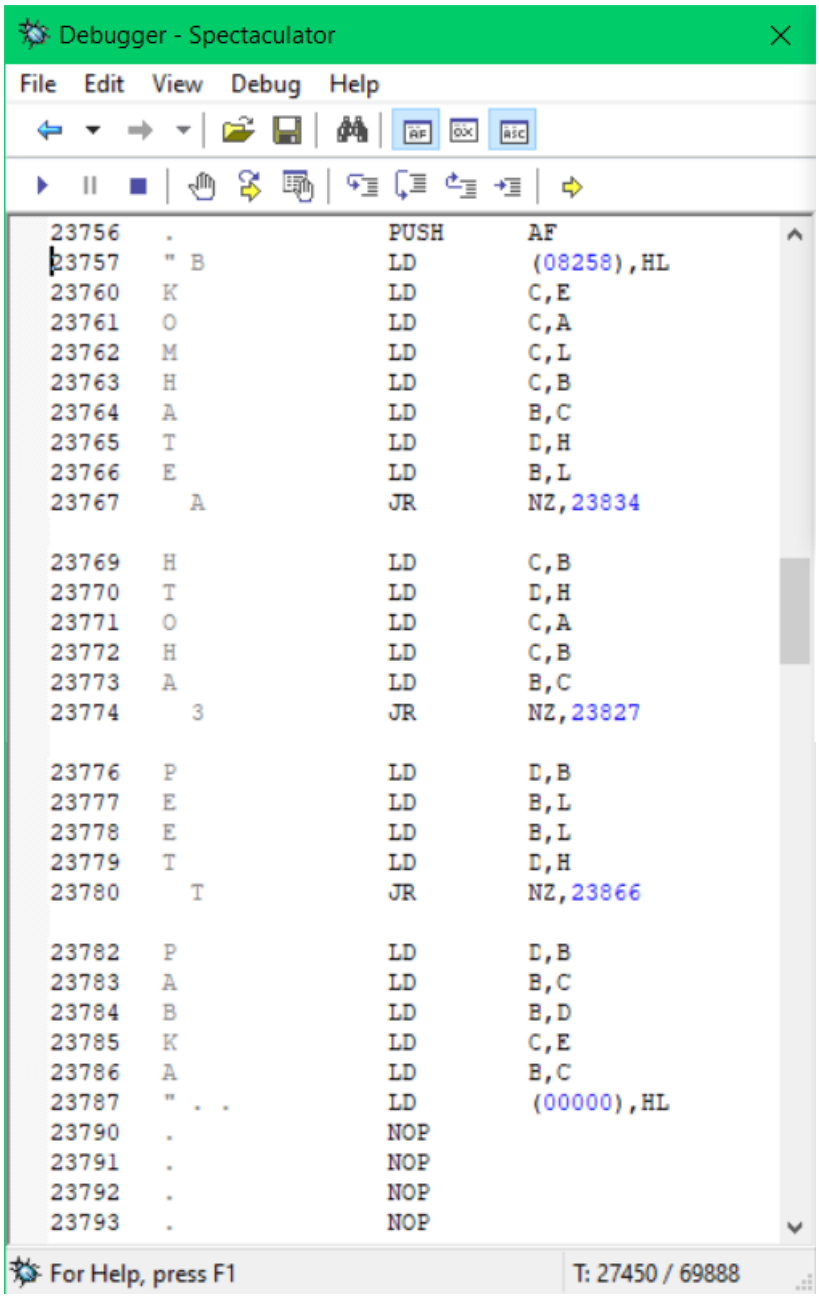


Рис. 109. Отображение содержимого ячеек памяти в символьном виде.

Ну вот, всё в порядке. Отожмите кнопочку обратно, и всё числовые значения возвратятся назад.

Итак, вы ввели строку и всё получилось? Отлично. Теперь в конце строки осталось поставить код ENTER (13) и маркер «128» в ячейку 23788, сразу за дальними кавычками.

Переводите вы взгляд на адрес 23788, а там такой подвох:

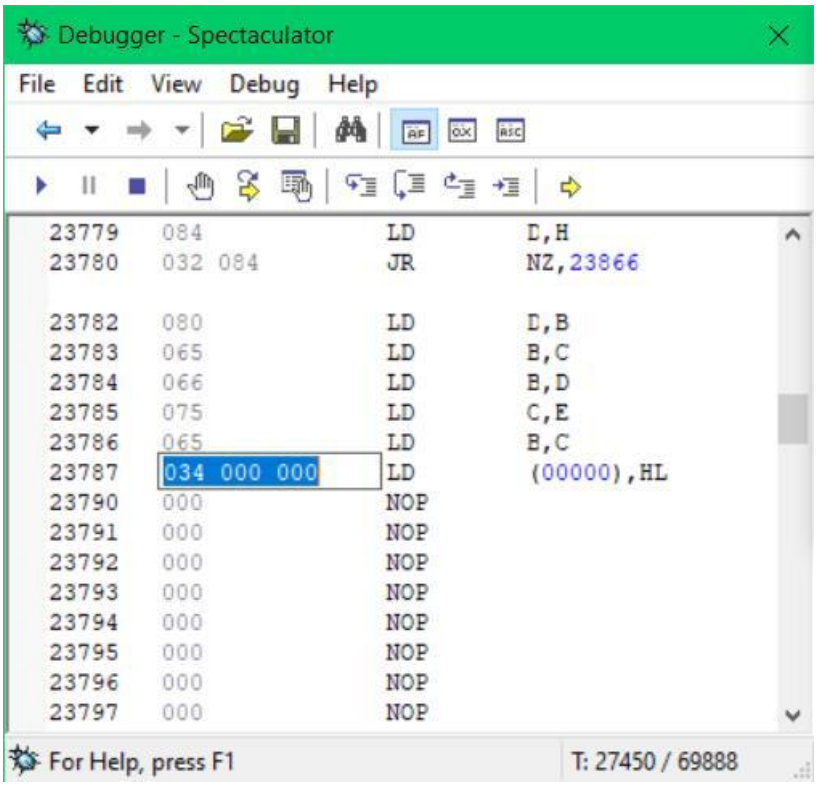


Рис. 110. Группировка чисел в неудобной последовательности по предполагаемым командам ассемблера.

Когда-нибудь это должно было произойти. Значения расположились в строчку с 23787, а нужные числа стоят вторым и третьим, а не первым. Как тут быть? Можно снять синее выделение, с двух последних чисел, стереть их и дописать 13 128.

Но есть вариант гораздо проще. Никуда не отходя, выполните «*Go To 23788*» на языке алгоритма (*нажмите CTRL+G и в окошко введите 23788*). Старая группировка чисел разрушится и первым адресом сверху встанет нужный:

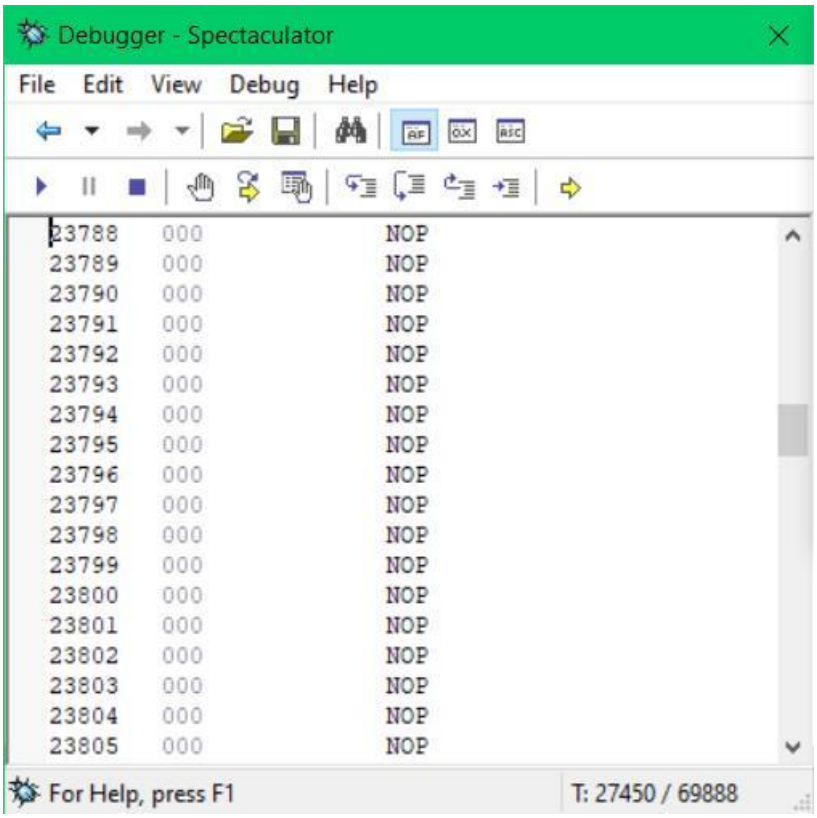


Рис. 111. Принудительное разрушение неудобно сгруппированной числовой последовательности.

Введите в самую верхнюю строку по адресу 23788 последовательность 13 128 и вопрос закрыт. Полная BASIC строка стала выглядеть так:

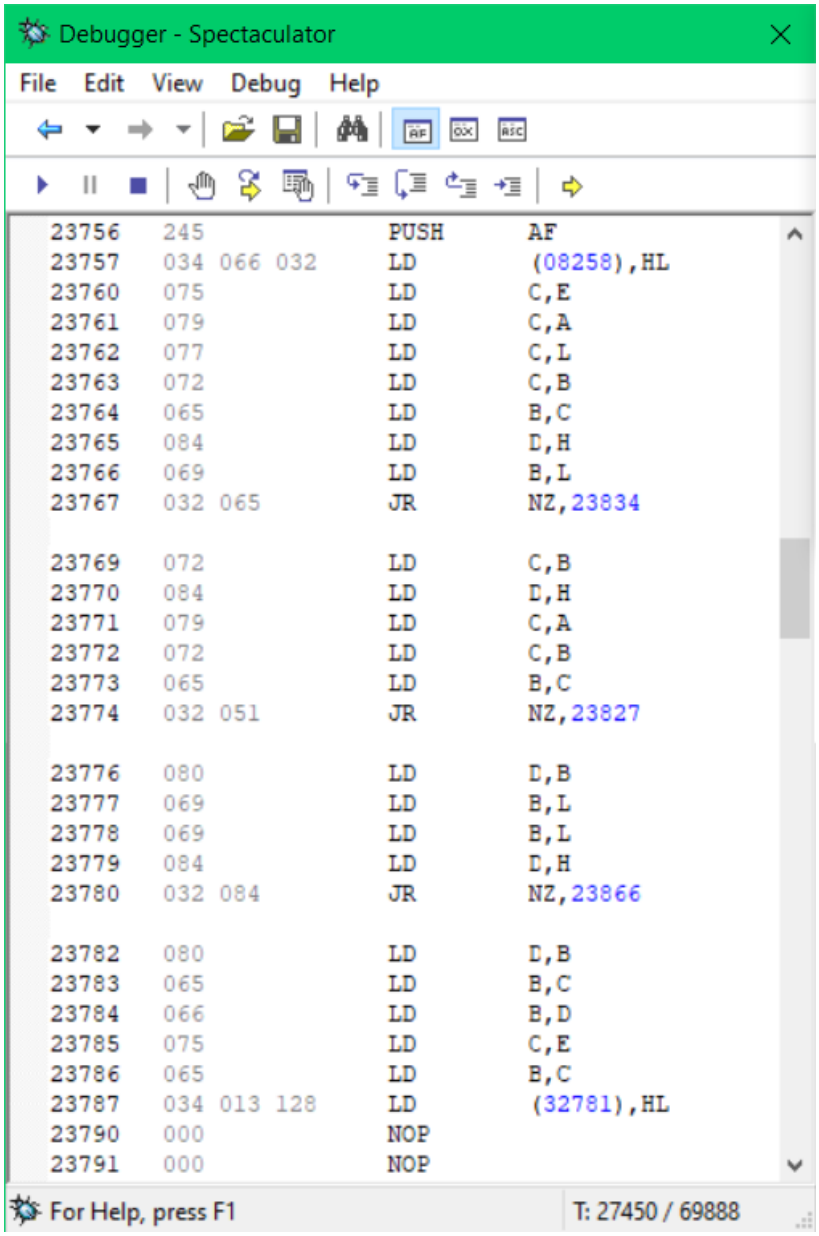


Рис. 112. Расположение полной BASIC строки с маркерами в памяти «Псевдоспектрума».

С BASIC частью всё. Следующим шагом нужно отодвинуть области `WORKSP (23649)`, `STKBOT (23651)`, и `STKEND (23651)` которые расположены сразу за вводимой строкой. Как видите из картинки, эти указатели нужно передвинуть на пустую ячейку, которая расположена сразу за концом BASIC строки. Этим адресом будет 23790.

Подойдите к ячейке 23649 и нажмите на значение для редактирования:

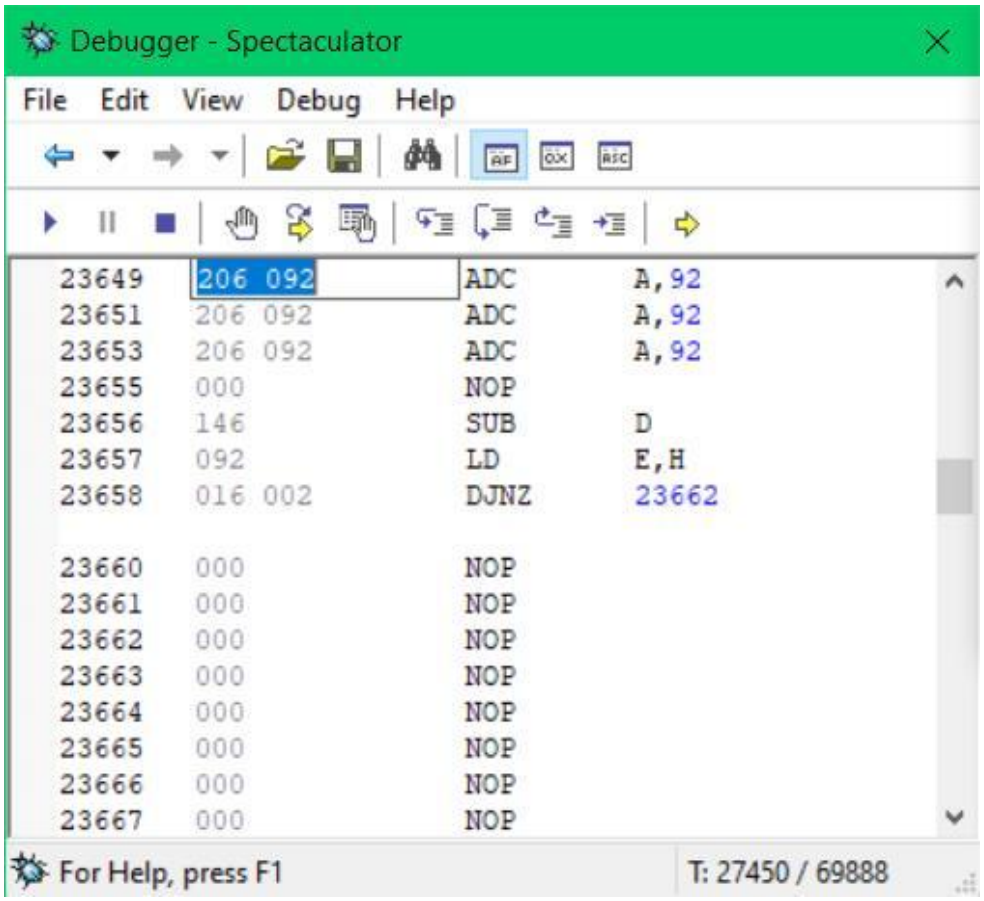


Рис. 113. Группа указателей WORKSP (23649), STKBOT (23651), и STKEND (23651).

А теперь вставьте в 23649 цепочку из трёх пятизначных чисел 23790 23790 23790

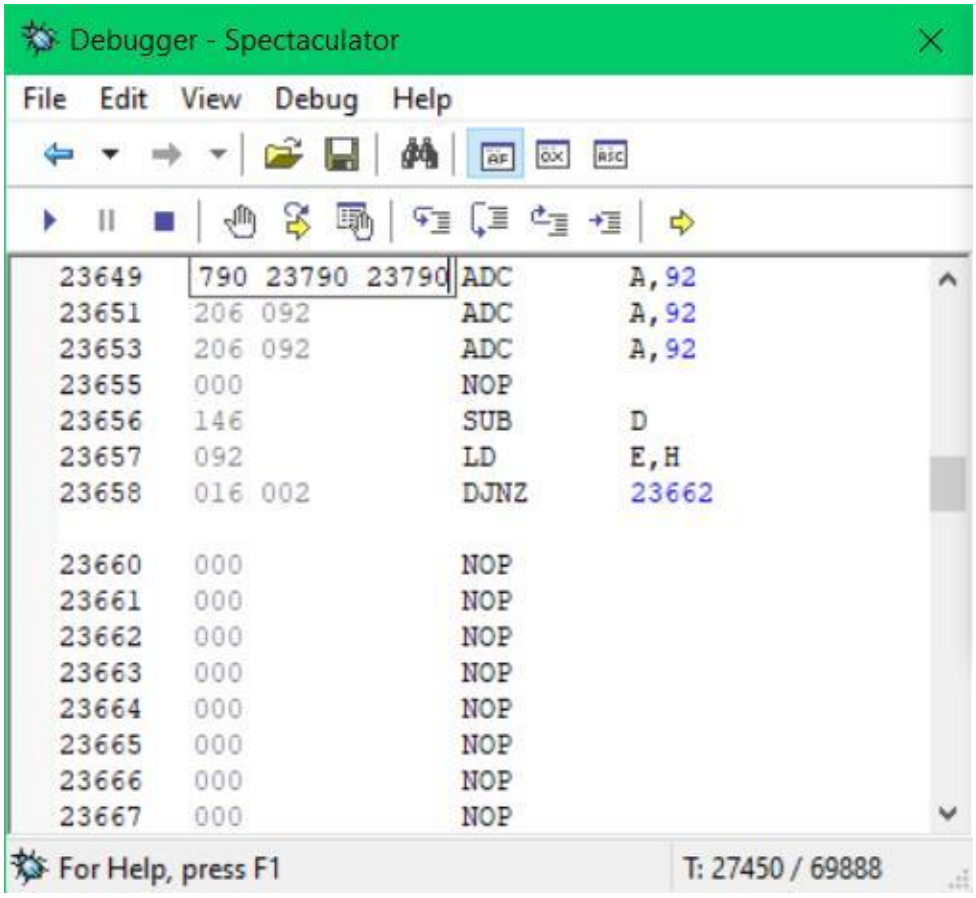


Рис. 114. Ввод цепочки пятизначных чисел в ячейку.

Вводите:

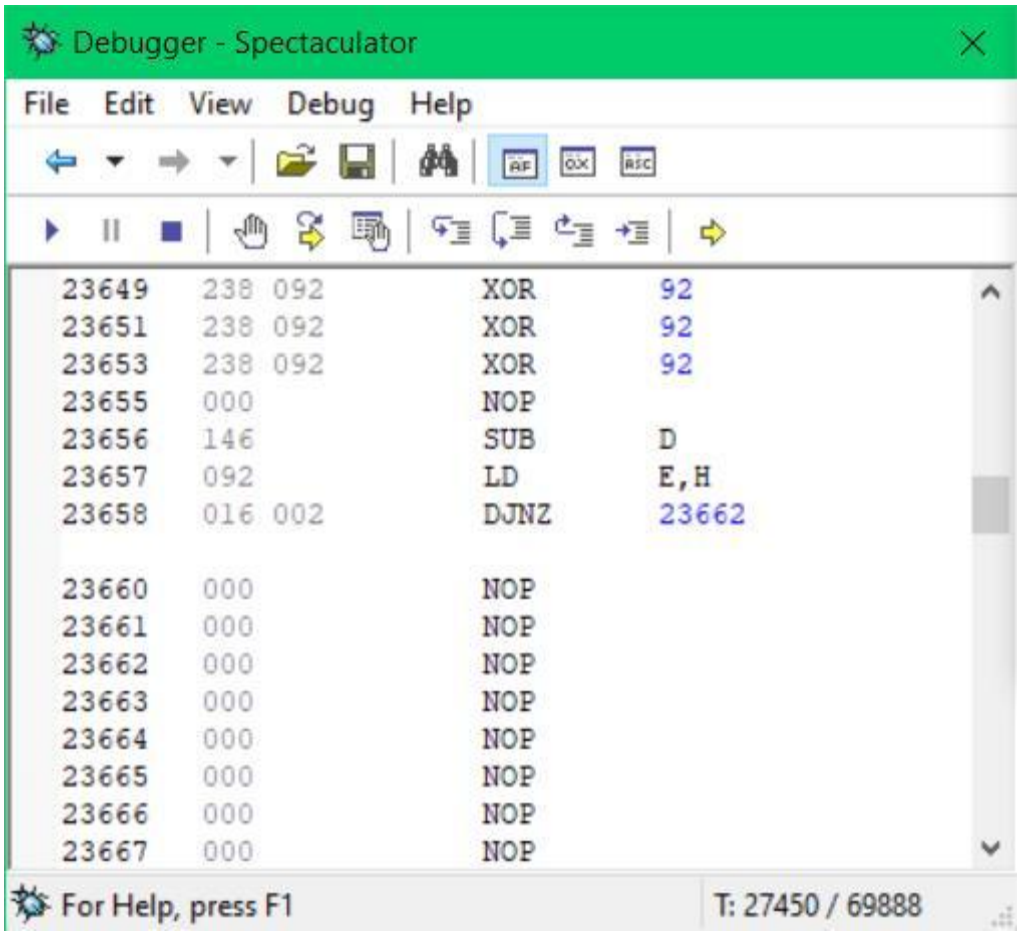


Рис. 115. Преобразование вводимого пятизначного числа в двухбайтное.

Вот и строчку из пятизначных чисел чудо отладчик перевел в машинный двухбайтный формат и расставил так, как требуется 23790=238+(92*256).

Вроде всё должно быть правильно, но проверить не помешает.

Для самоконтроля предлагаю обновить содержимое и посмотреть, как это выглядит в реальности. Пользуясь навыками из прошлых глав, это сделать не сложно. Выставьте по адресу `LAST_K (23560)` какое-нибудь управляющее значение курсора, например «11». Ну а в `FLAGS (23611)` впишите 32. После ввода этих значений нажмите «*F5*» или «*Trace*» для выхода в *Spectaculator*:

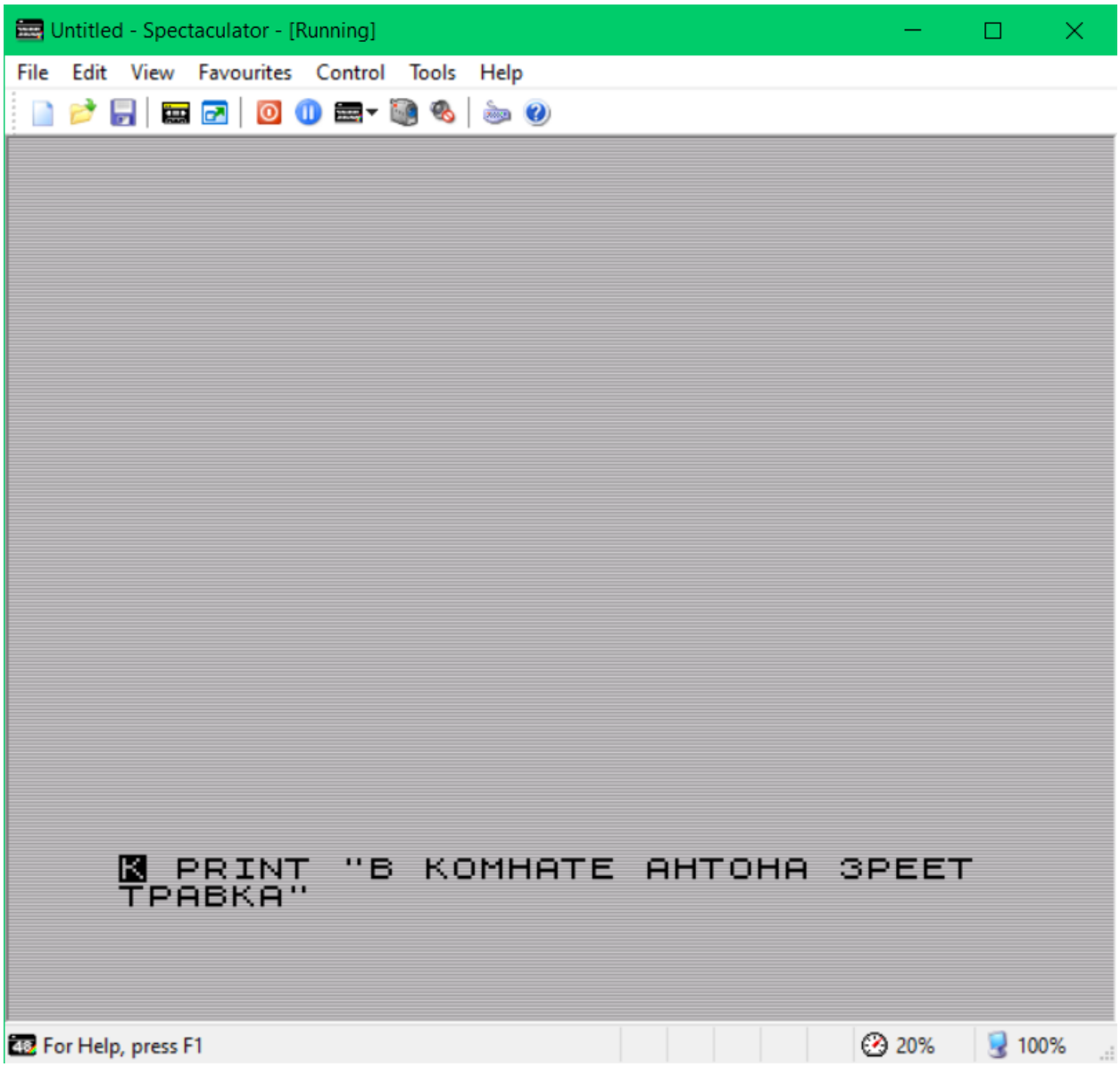


Рис. 116. Неправильная позиция курсора после искусственно синтезированной строки.

Оригинально, но портит эстетику один момент. Курсор K мигает в начале строки. Нужно переставить его в конец, как положено. За быструю перестановку курсора отвечает переменная K_CUR по адресу 23643.

Предлагаю внести последний штрих и поставить курсор по адресу 23788 на ENTER, как и положено. Возвращайтесь в «*Debugger*». Введите 23643 ← 23788 и 23612 ← 9:

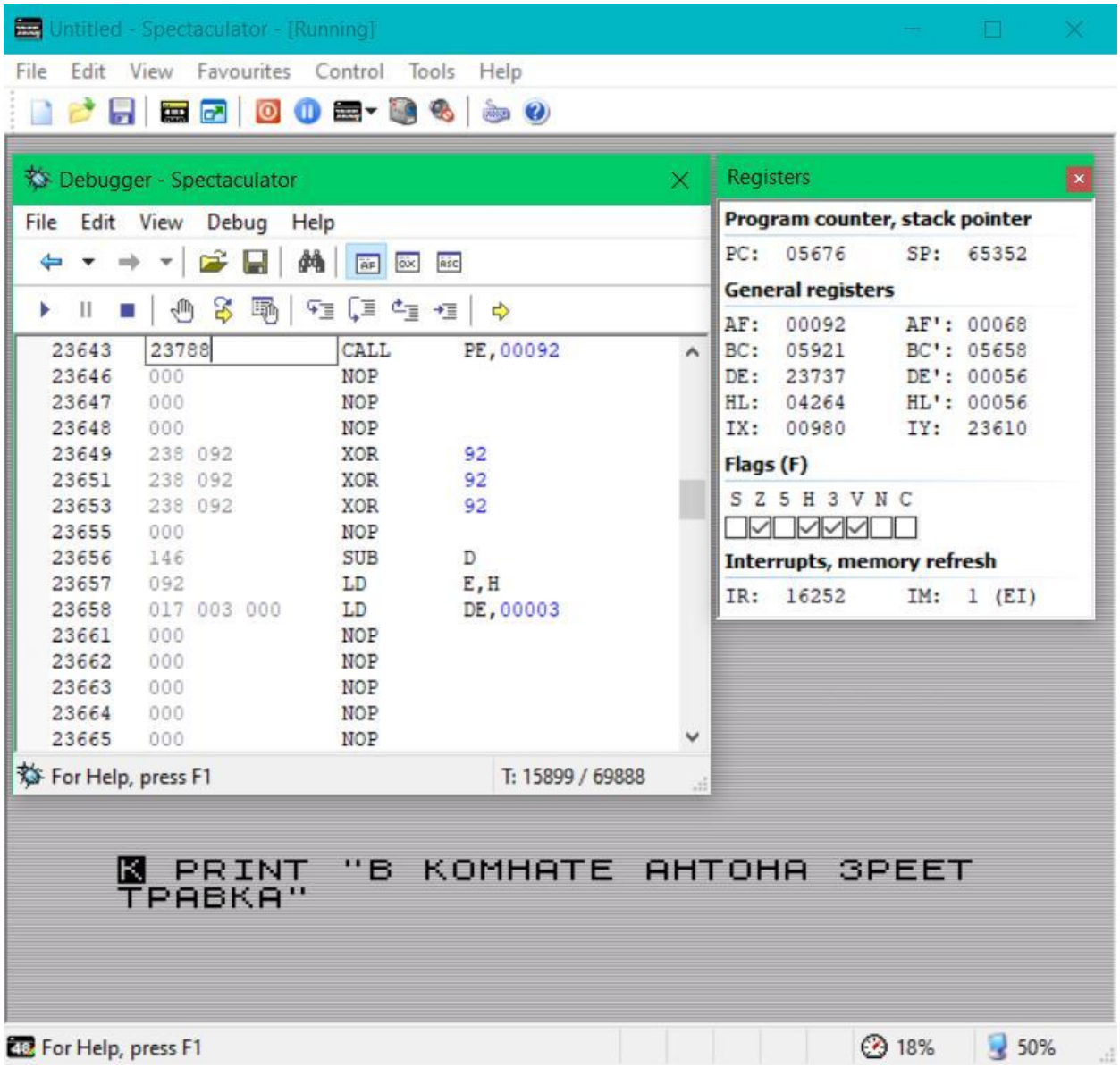


Рис. 117. Ввод значения в ячейку для переустановки мигающего курсора в конец BASIC строки.

Нажимайте ▸ «*Trace (F5)*»:

Рис. 118. Переустановка мигающего курсора в конец BASIC строки. Фрагмент нижней части экрана.

Ну вот всё. Курсор стоит на нужном месте. Теперь осталось заставить выполняться эту цепочку чисел.

В «*Debugger*» подойдите к нужным адресам в `LAST_K` с `FLAGS` и выставите требуемые значения (23560 ← 13 и 23611 ← 32).

Нажимайте «*Trace (F5)*»:

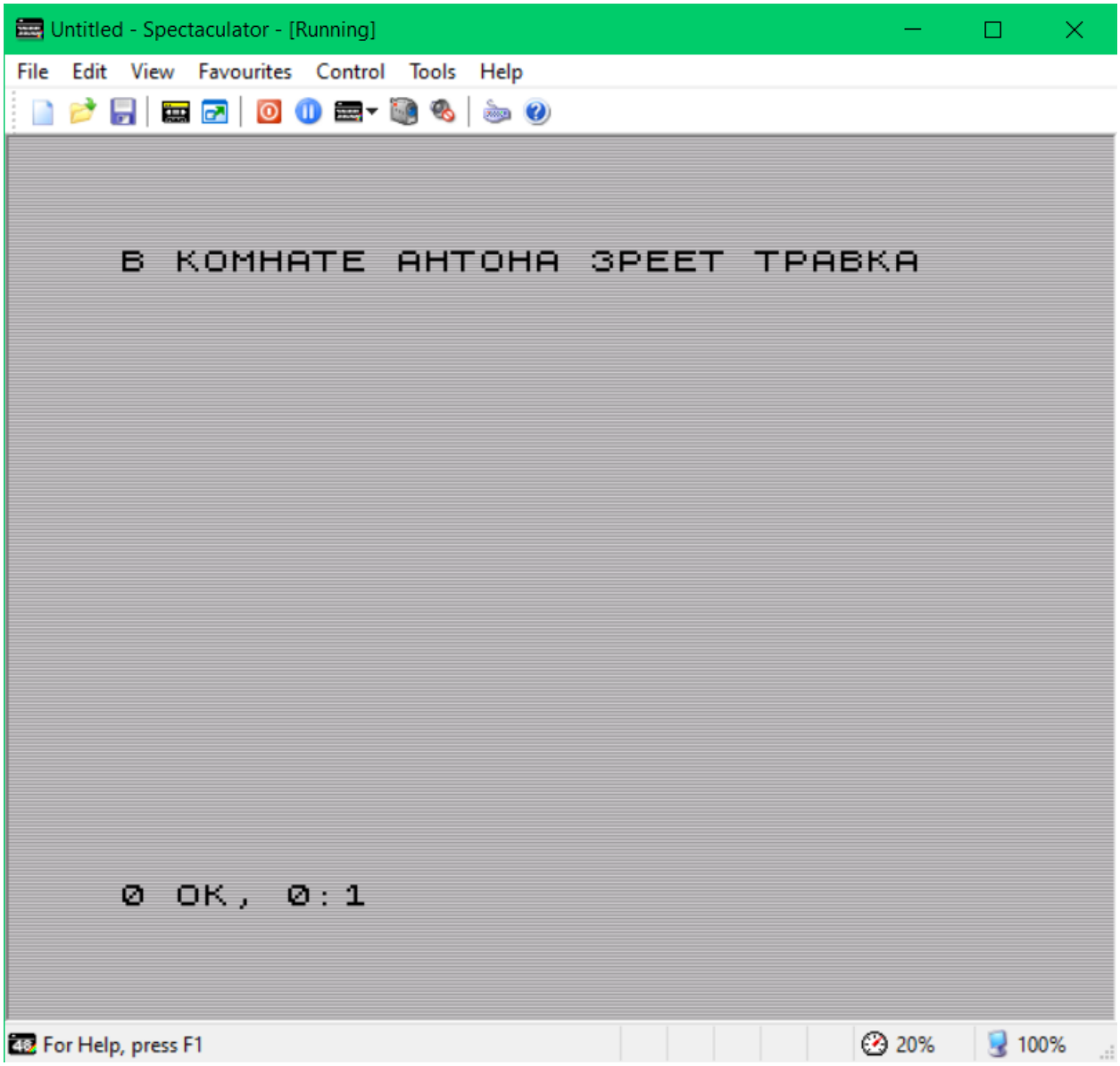


Рис. 119. Результат выполнения синтезированной строки.

Невероятно, но это получилось. Поздравляю себя и вас! Несмотря на забористую кошачью траву, поставленная задача завершена на все 146,47%. Искусственно синтезированная BASIC строка выполнилась даже без прикосновений к стандартным внутренним средствам.

Попробую представить программу ввода строки на языке «*God Mode*»:

```
Debugger
Dec
Go To 23560
23560 ← 13
23611 ← 32
23649 ← 23790 23790 23790
23756 ← 245
23757 ←""B KOMHATE AHTOHA ЗРЕЕТ TPABKA"
Go To 23788
23788 ← 13 128
Trace
```

В виде программы выглядит не так уж и страшно. Между делом появилась ещё одна команда – «*ASCII*».

Глава 11
Закольцовка безномерной строки. Суета спасёт мир

Краткое содержание: NEWPPC (23618), NSPPC (23620), PPC (23621), SUBPPC (23623) запуск фрагмента строки в реальном времени, закольцовка строки

После главы с основами программирования обычно предлагают изучить циклы. Это конечно, не совсем цикл, но вещь эксклюзивная. Почему-то ни в одной книге по BASIC данную тему не раскрывали.

Наберите и запустите следующую программу:

```
Debugger
Dec
Go To 23611
23611 ← 32
23643 ← 23781
23649 ← 23783 23783 23783
23756 ← 245 34 67 121 101 109 97 34 58 245
23766 ← 34 67 110 97 99 101 109 32 77 117 112
23777 ← 34 58 242 48 13 128
Trace
```

На экране в буфере редактора появилась следующая BASIC строка:

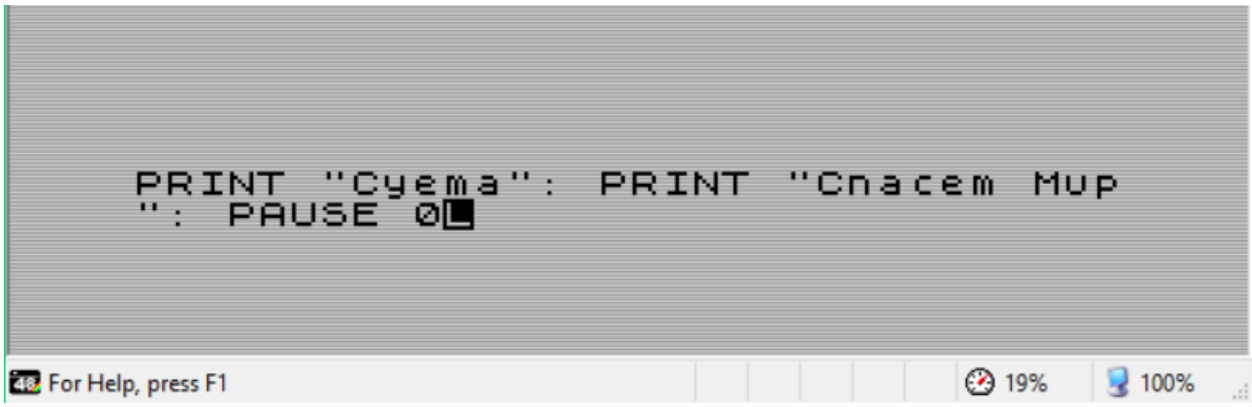


Рис. 120. Строка набранной программы. Фрагмент нижней части экрана.

Не, ну а чего? На лоне природы, вдали от людской хуеты, царит суета и умиротворение. Только тут, за тысячи километров от цивилизации, можно часами наслаждаться горной рекой, как из глубокой подземной пещеры, кристально чистые потоки воды, несутся по вычурным продолговатым камням. А эта красивая величественная гора! С её живописного склона открываются потрясающие виды на бескрайние леса.

Иными словами, ~~Суета~~ красота спасёт мир:

Рис. 121. Надпись «Суета» на восточном портале коллектора Муринского ручья под станцией Мурино.

Так вот ты какая Суета! Суетологи всех стран – объединяйтесь! Короче, заходите в «*Debugger*» и виртуально запускайте программу (23560 ← 13 и 23611 ← 32):

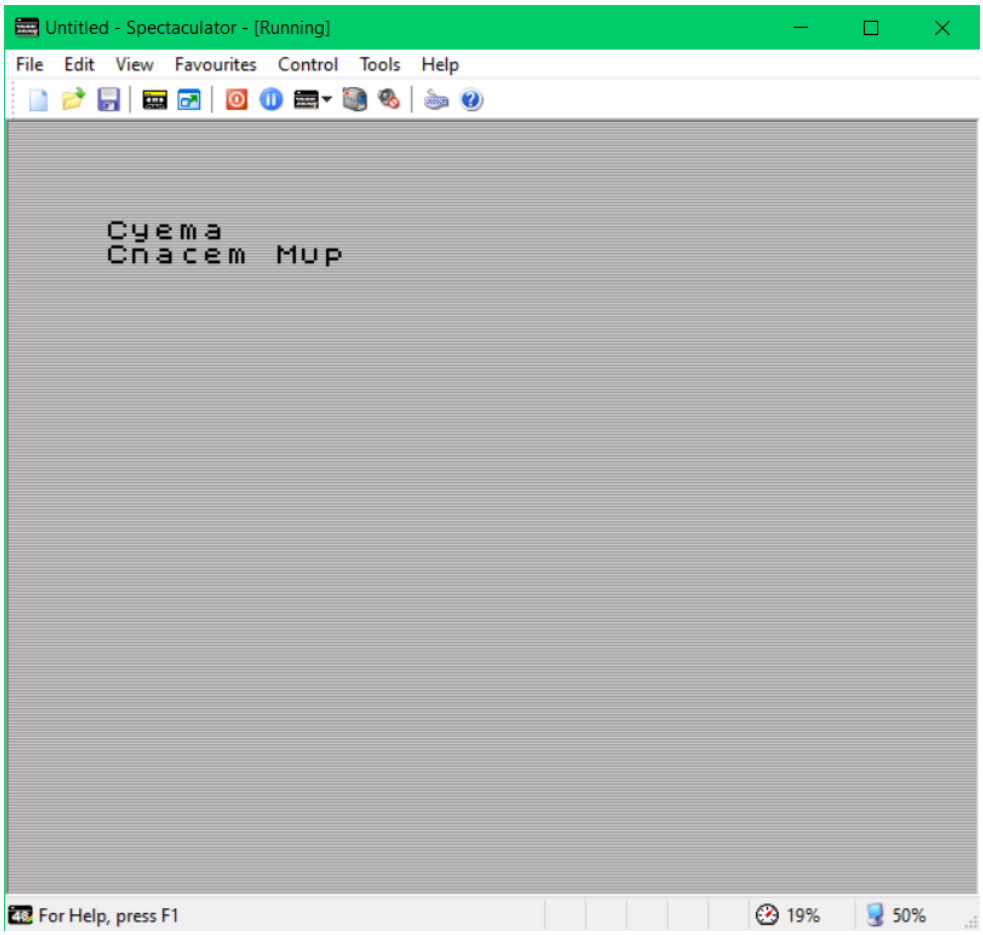


Рис. 122. Вывод текста «Суета спасёт мир» безномерной строкой.

После запуска строки в правом верхнем углу экрана гордо появилась «Суета». Строчкой ниже скромно приютилось продолжение «Спасёт мир». *Spectaculator* застыл в ожидании нажатия любой клавиши, после чего выполнение программы гарантированно закончится выдачей сообщения «0 OK».

Давайте попробуем повторить выполнение определённого куска, не завершая работу BASIC программы. На этот раз, не прибегая к долгому анализу программ, предлагаю просто посмотреть в старый справочник:

NEWPPC 23618/19 IY+8 (#5C42/43)

Номер строки бейсик-программы, в которой расположен следующий выполняемый оператор.

NSPPS 23620 IY+10 (#5C44)

Номер следующего зыполняемого оператора в строке бейсик-программы. Переменные NEWPPC и NSPPS можно использовать для перехода к произвольному оператору в бейсик-программе.

PPC 23621/22 IY+11 (#5C45/46)

Номер строки бейсик-программы, в которой расположен выполняемый оператор. При выполнении оператора в непосредственном режиме в переменную записано число 65534 (#FFFE).

SUBPPC 23623 IY+13 (#5C47)

Номер выполняемого оператора в строке бейсик-программы.

Рис. 123. Набор переменных для управления выполняемыми строками (из старого справочника).

Несмотря на то, что информация о «ПэПэЦэшных» ячейках не являлась секретом со времён первого издания справочника Ларченко и Родионова «*ZX-Spectrum для пользователей и программистов*», развитие идеи с использованием этих значений я так и не встретил.

Над подобным запуском вводимой строки я бился в конце 1992 года, постигая основы языка BASIC, но до ума не довёл. Пришло время подчищать хвосты. Стоит попытаться решить задачу свежим взглядом 2024 года? Однозначно!

Откройте «*Debugger*» на адресе `NEWPPC (23618)` и внимательно посмотрите все четыре переменные:

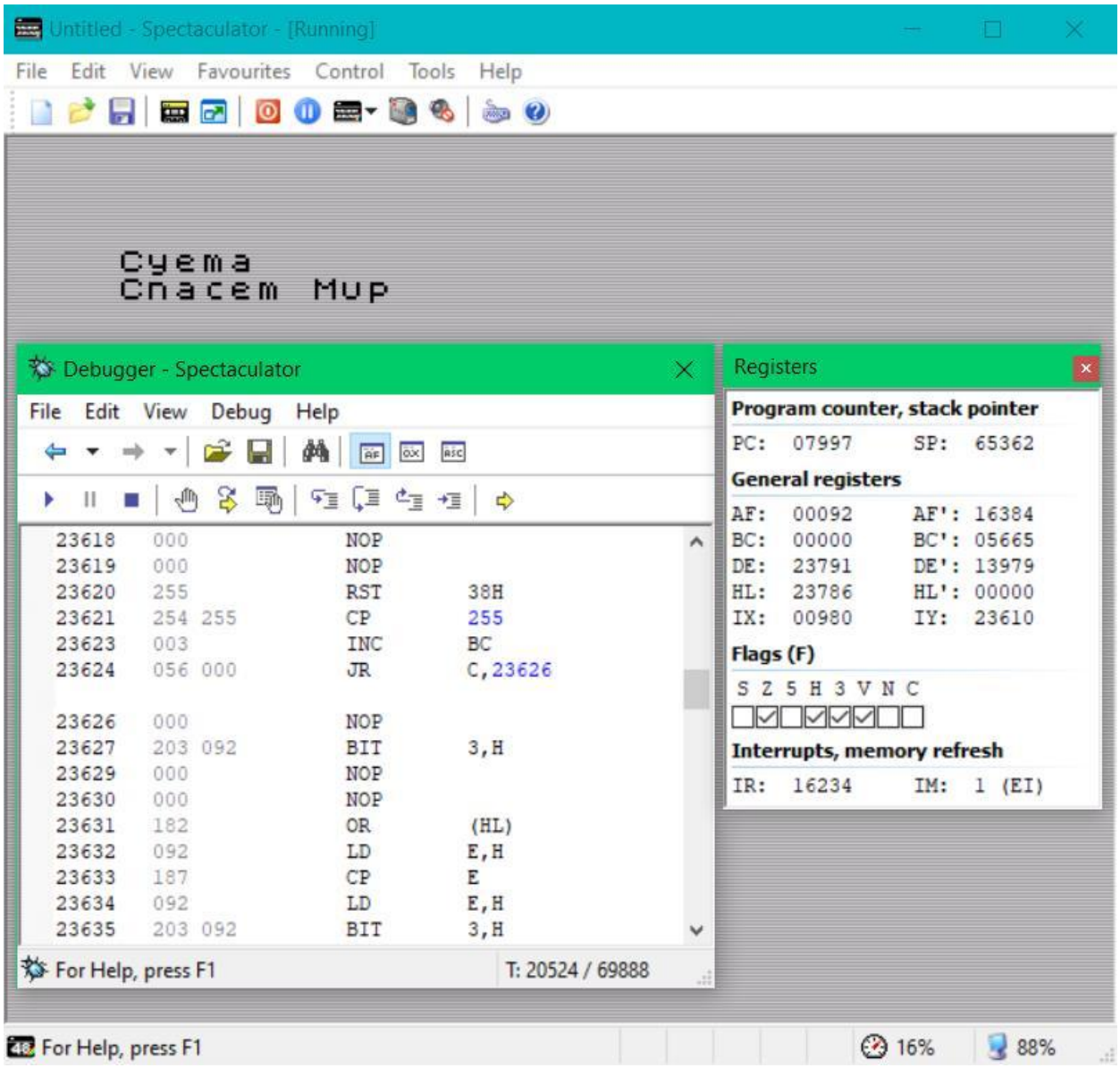


Рис. 124. Ячейки NEWPPC, NSPPC, PPC, SUBPPC в области системных переменных.

О текущих процессах выполнения сигнализируют переменные `PPC (23621)` и `SUBPPC (23623)`. Обратите внимание, «безномерная» строка в момент выполнения имеет номер «65534». Номер выполняемого оператора в `SUBPPC (23623)` забит значением «3», что действительно соответствует команде `PAUSE`, на которой в данный момент остановилось выполнение строки.

Чтобы без выхода в редактор повторить фрагмент выполняемой строки, достаточно в переменную `NEWPPC (23618)` скопировать 65534. В `NSPPC (23620)`, которая отвечает за следующий выполняемый оператор, вместо заглушки «255» поставить 1 или 2. Для

вывода полученного результата включить бит №5 в переменной FLAGS (23611). Учитывая, что программа в процессе выполнения, кроме бита №5 следует оставить включённым и бит №7. Итого в 23611 нужно записать 128+32=160.

Проще говоря, находясь в отладчике, выполните следующий алгоритм, вписав три значения в ячейки:

23611 ← 160
23618 ← 65534
23620 ← 2

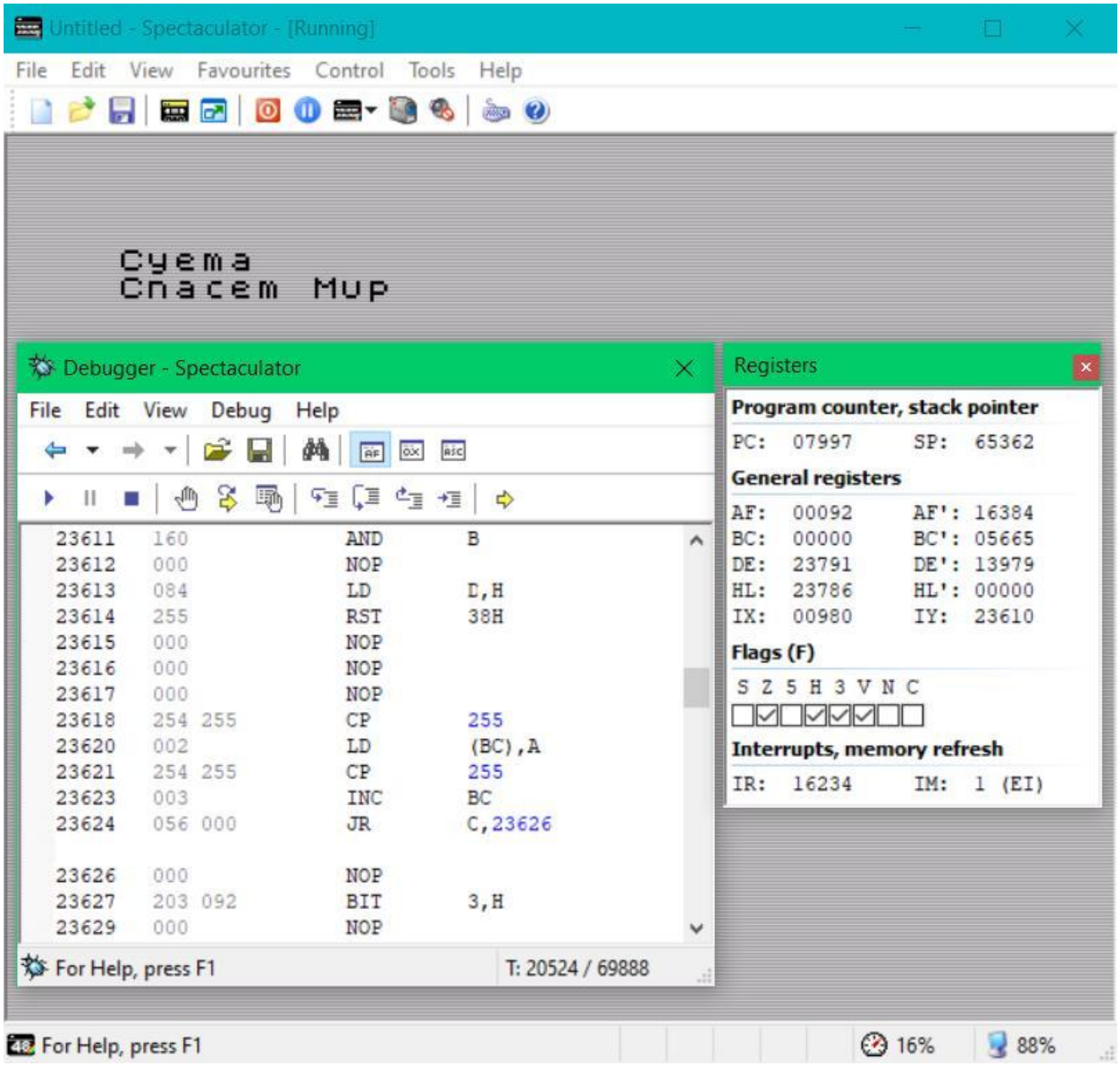


Рис. 125. Внесение значений для повтора выполнения BASIC строки.

Запускайте командой «*Trace*»:

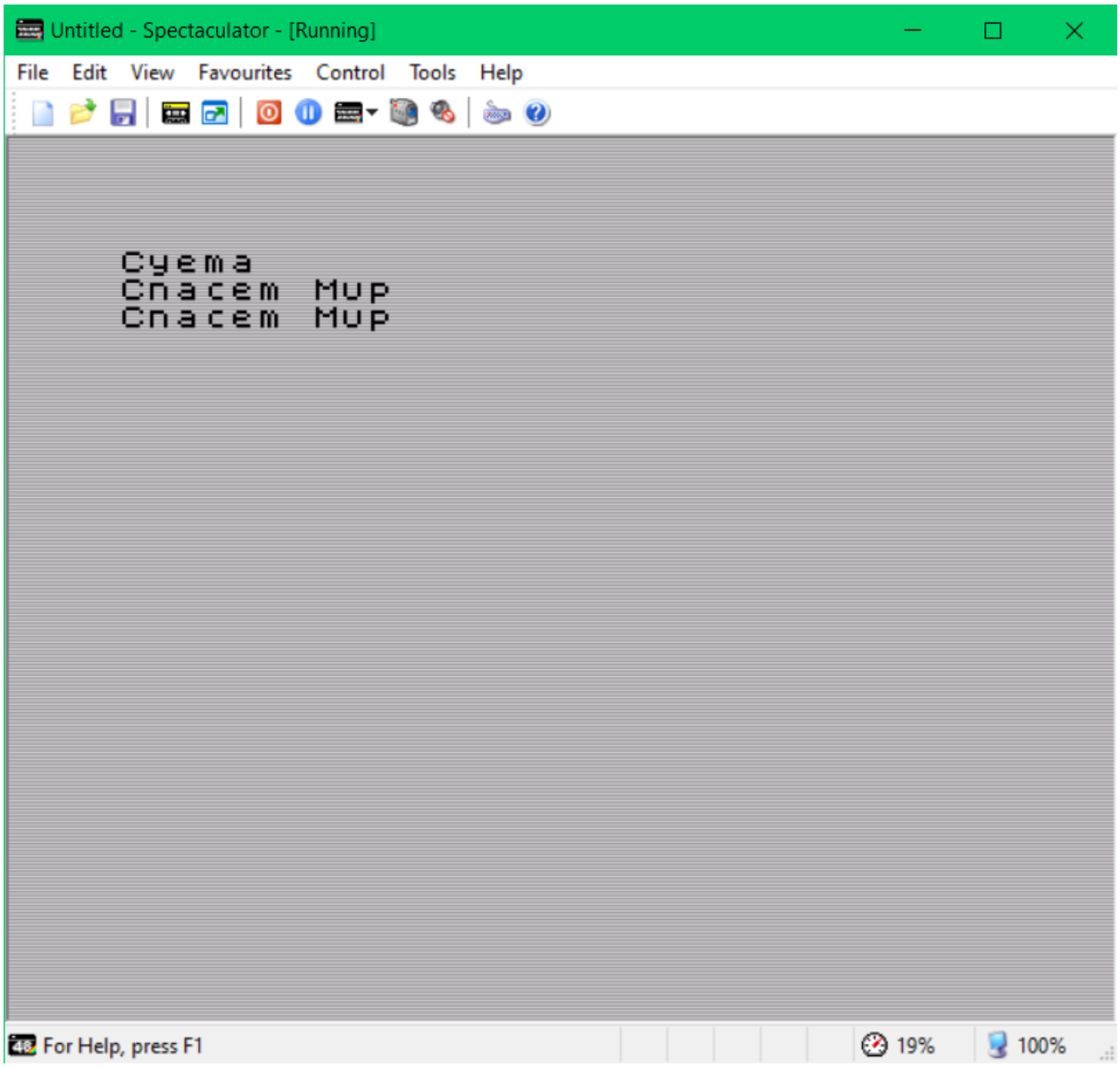


Рис. 126. Повтор второй части фразы после модификации NEWPPC и NSPPC.

Произошла повторная попытка спасти мир от безумия. На экран вывелась соответствующая запись, после чего опять программа перешла в режим `PAUSE 0`.

Откройте Debugger в `NEWPPC (23618)` и посмотрите, что изменилось:

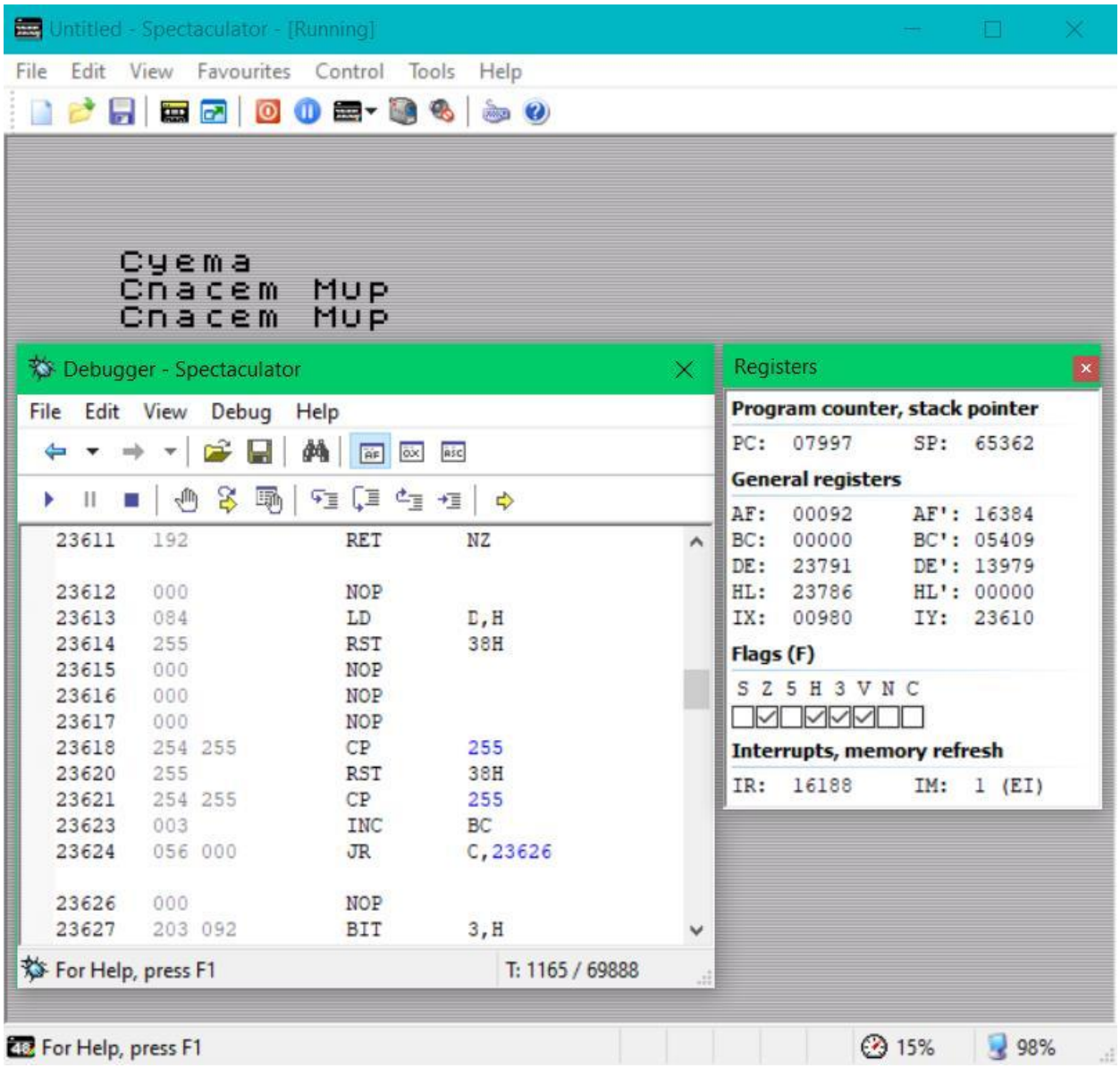


Рис. 127. Появление заглушки в NSPPC (23620) после повтора фрагмента BASIC строки.

В `NSPPC (23620)` снова появилась затычка «255», а вот значение в `NEWPPC (23618)` осталось без изменений с предыдущего вмешательства. Ну и отлично, значит, задача упростилась.

Еще раз прервите выполнение программы и для завершения эксперимента запустите строку с самого начала. Выполните 23620 ← 1 и 23611 ← 160, после чего нажмите ▸ «*Trace (F5)*»:

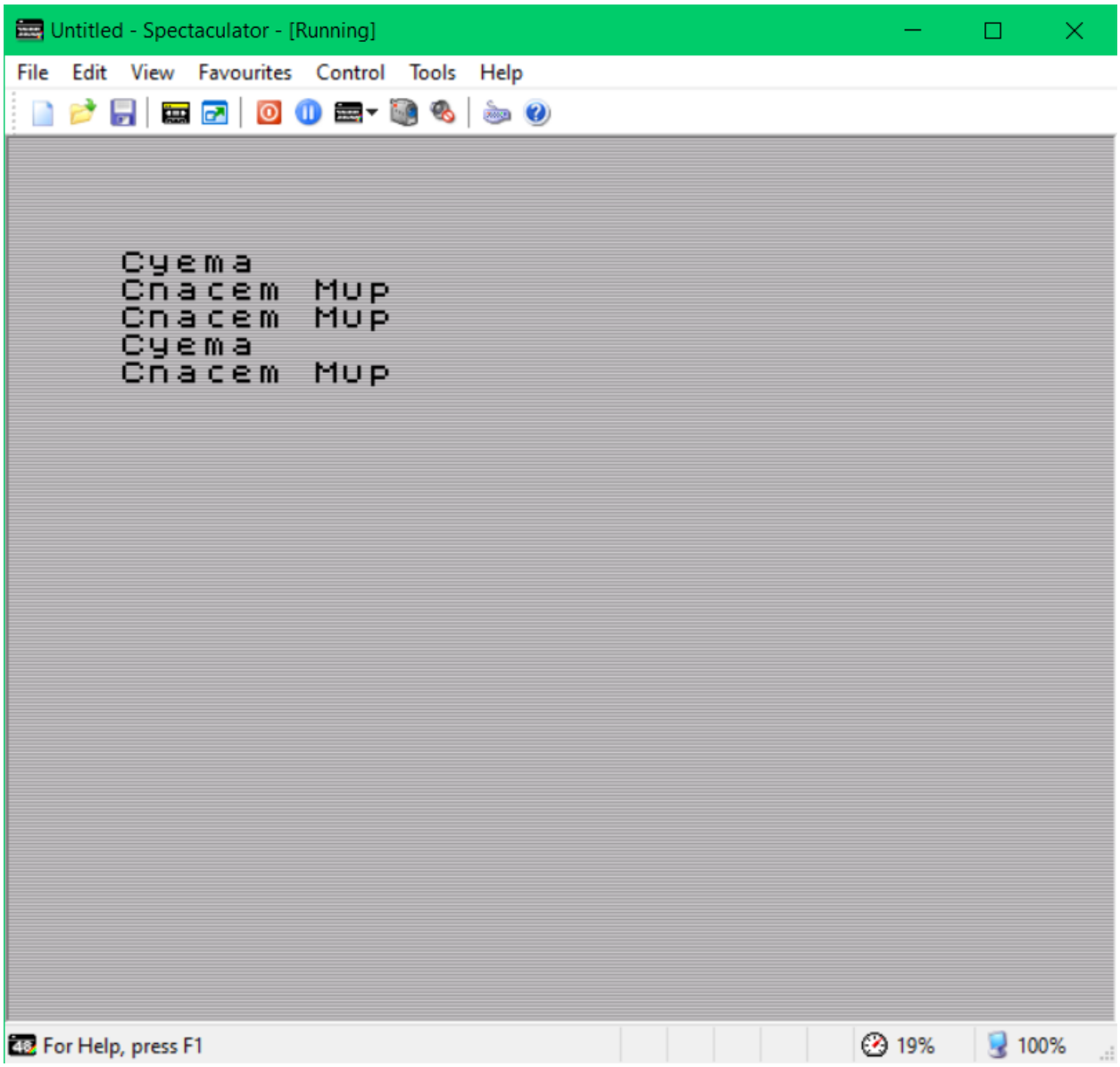


Рис. 128. Повтор выполнения полной BASIC строки с оператора 1.

Перед уходом в режим ожидания, Суета предприняла третью попытку спасти мир.

А если без шуток, то изменив значение в `NSPPC (23620)`, эмулятор очередной раз прервал выполнение текущего куска BASIC строки и перешёл к указанному месту.

Войдите в отладчик и имитируйте нажатие любой клавиши, убедившись, что программа не зависла и успешно завершилась после повтора. Поскольку в `LAST_K (23560)` ещё остался лежать `ENTER`, достаточно просто открыть «*Debugger*» и в адрес 23611 очередной раз записать 32. После чего можно выйти из отладчика через «*Trace*», или просто его закрыв:

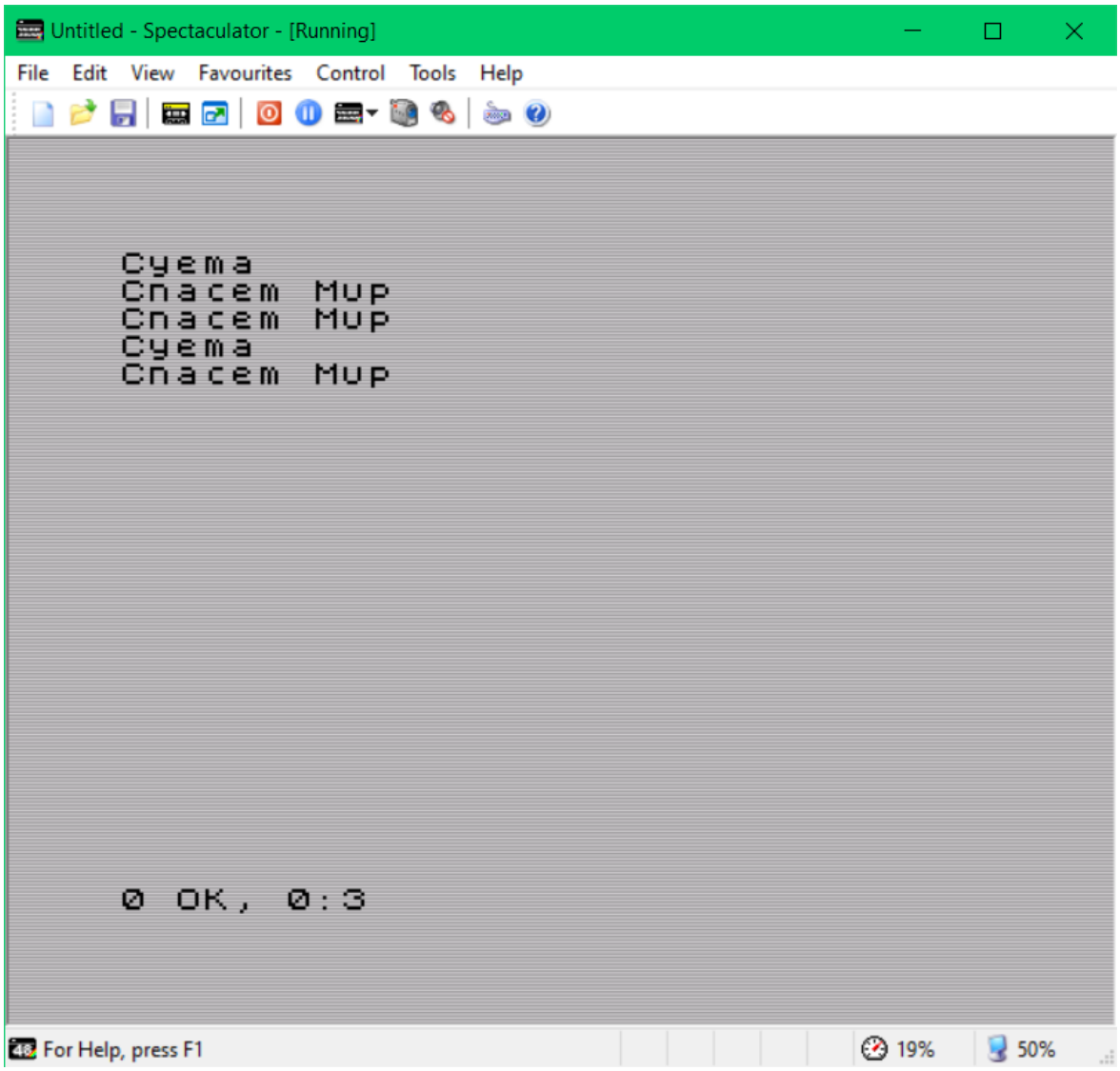


Рис. 129. Выход в BASIC после нажатия любой клавиши.

Идея понятна? Если нет, подскажу. На этом свойстве вполне реально создать автономную BASIC программу, которая будет состоять из введенной строки с бесконечным циклом. Строка с «`GO TO ХЕРЗНАЕТКУДА`».

BASIC строка будет следующей:

```
    BORDER 1: PAUSE 0: BORDER 2:  PAUSE 0: POKE
23618, 255: POKE 23619, 254: POKE 23620, 1
```

Для набора в *Spectaculator* предлагаю выполнить следующий алгоритм:

```
Debugger
Go To 23560
23560 ← 13
23611 ← 32
23649 ← 23800 23800 23800
23756 ← 231 49 58 242 48 58 231 50 58 242 48 58
23768 ← 244 50 51 54 49 56 44 50 53 53
23778 ← 58 244 50 51 54 49 57 44 50 53 52
23789 ← 58 244 50 51 54 50 48 44 49 13 128
Trace
```

Запустив её, на экране появится синяя рамка и ожидание нажатой клавиши:

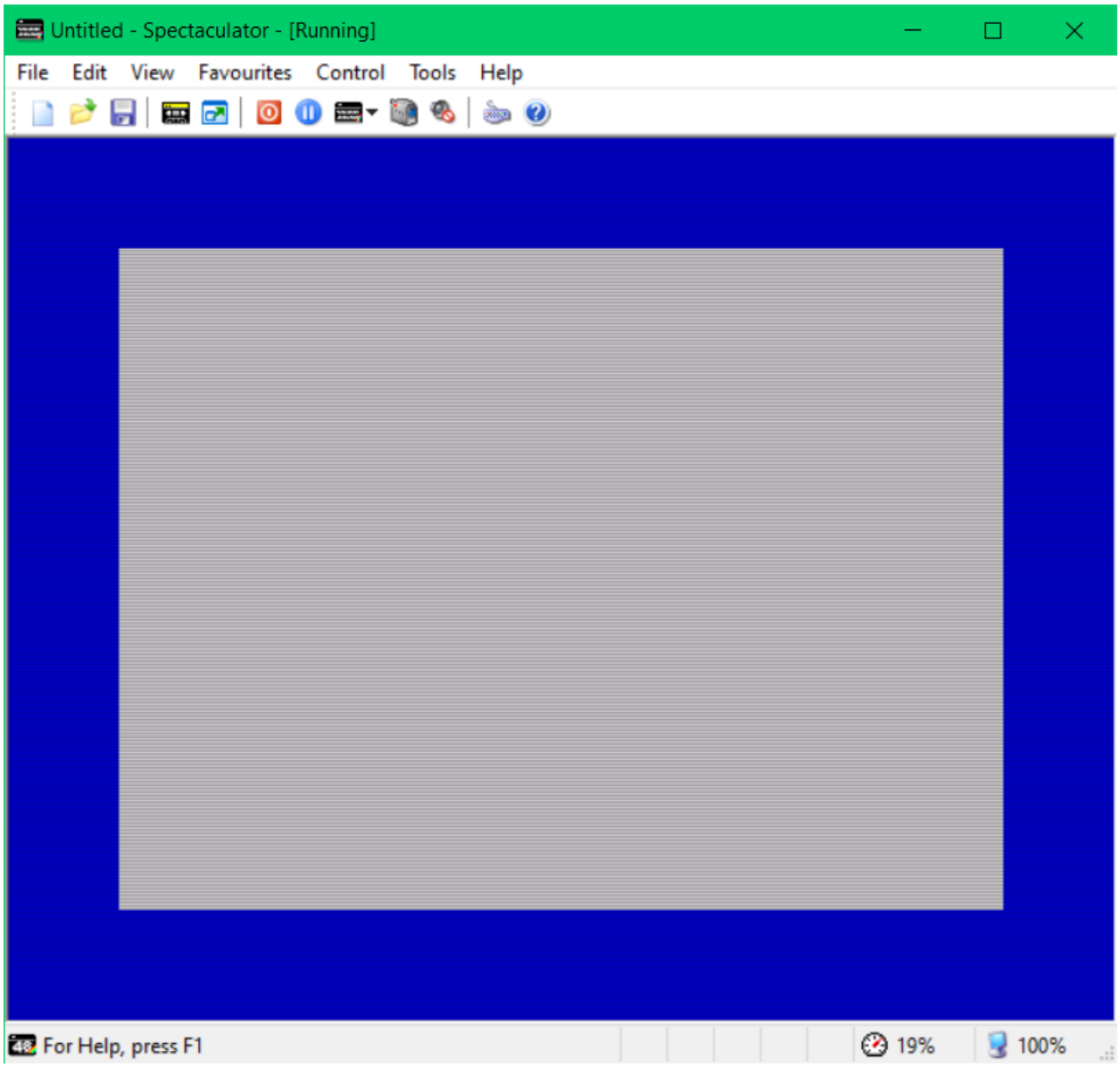


Рис. 130. Ожидание нажатия клавиши в закольцованной безномерной BASIC строке.

Нажимая любую клавишу, цвет рамки будет чередоваться с синего на красный. Выйти из программы можно нажатием CAPS SHIFT + BREAK SPACE:

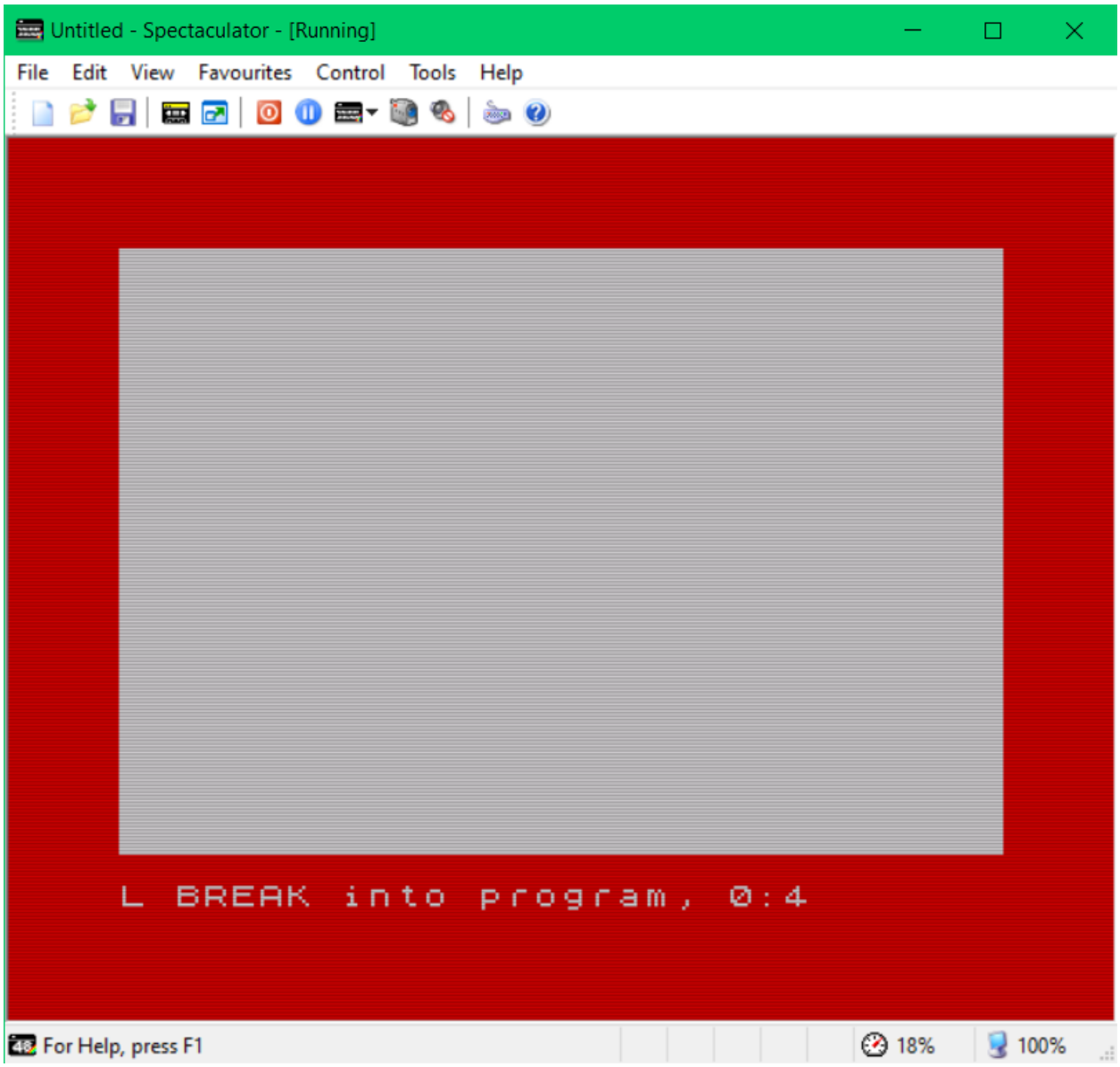


Рис. 131. Вскрытие закольцованной безномерной строки нажатием CAPS SHIFT + BREAK SPACE.

На этом всё и до встречи в следующей главе.

Глава 12
VARS: переменные, массивы и... Питерская повседневность

Краткое содержание: переменные, массивы, VARS (23627)

Этажом выше буфера ввода строки расположена область переменных «VARS». (*жарг. варка, конфорка*). Почти что от слова «Варить». При этом слове сразу вспоминается белая газовая плита середины 1960-х годов от завода Ленгазаппарат №4. Вот такой агрегат (только вроде четырёхконфорочный) стоял в моей старой купчинской квартире, аж до 1990 года:

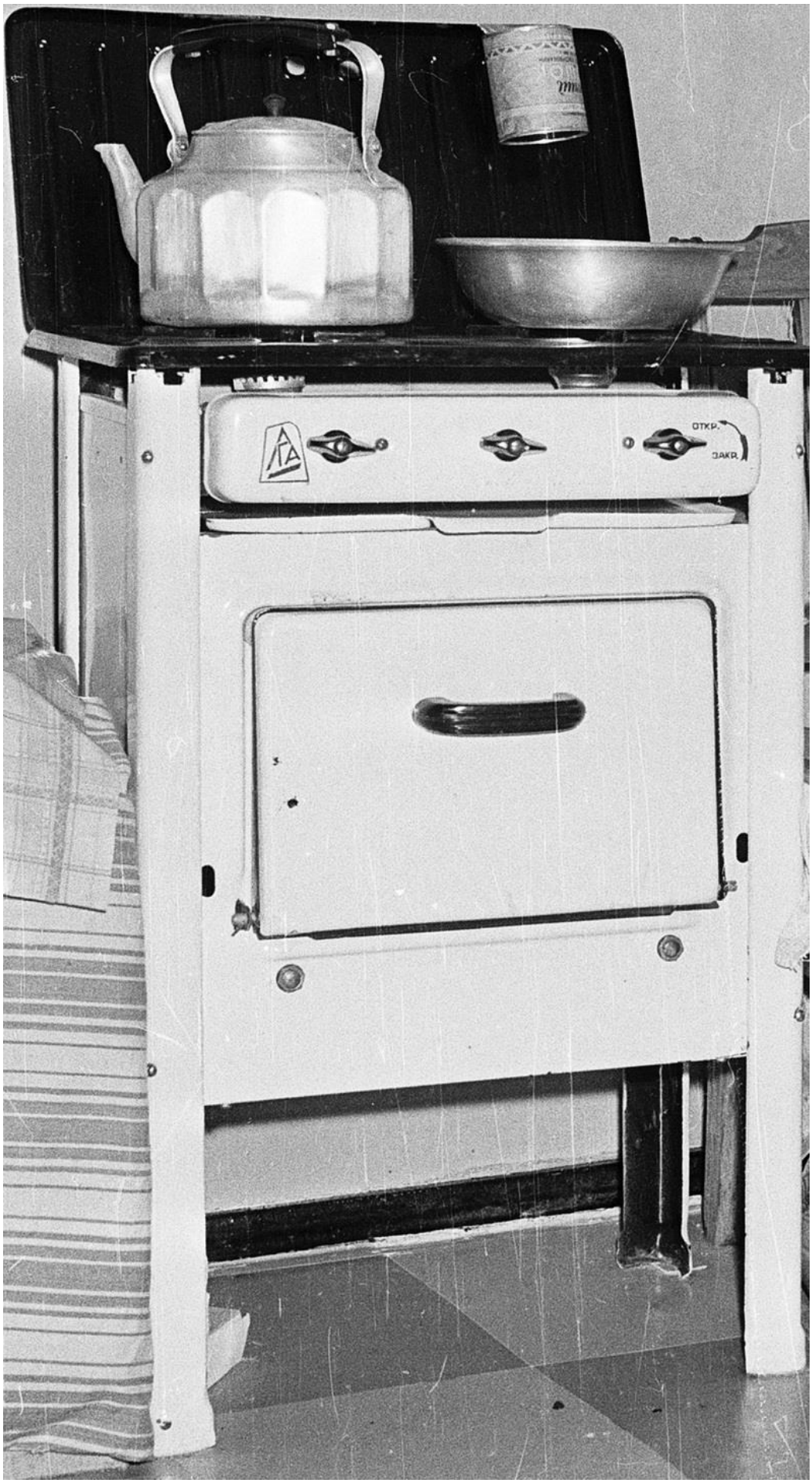


Рис. 132. Газовая плита 1960-х годов завода Ленгазаппарат №4. Фото из интернета.

А на самом деле, в тех ячейках хранятся результаты работы всех BASIC команд, которые используют или формируют переменные. Например: LET, INPUT, READ, FOR, NEXT, DIM... На чистом компьютере ячейки VARS (23627) указывают на адрес 23755, в котором стоит заглушка «128».

Как формируется переменная? После ввода строки, например:

```
LET a$="TEKCT"
```

По нажатию ENTER происходит выход из редактора и вход в анализатор строки LINE SCAN по адресу 6935. После всех приготовлений строка проверяется на двоеточие с «ENTER'ом» и рассматривается первый оператор. Дальше методом дробления на группы и классы происходит подборка программы действия для соответствующего оператора. Определив, что требуется обработка переменных, запускается подпрограмма распознавания переменных LOOK-VARS по адресу 10418:

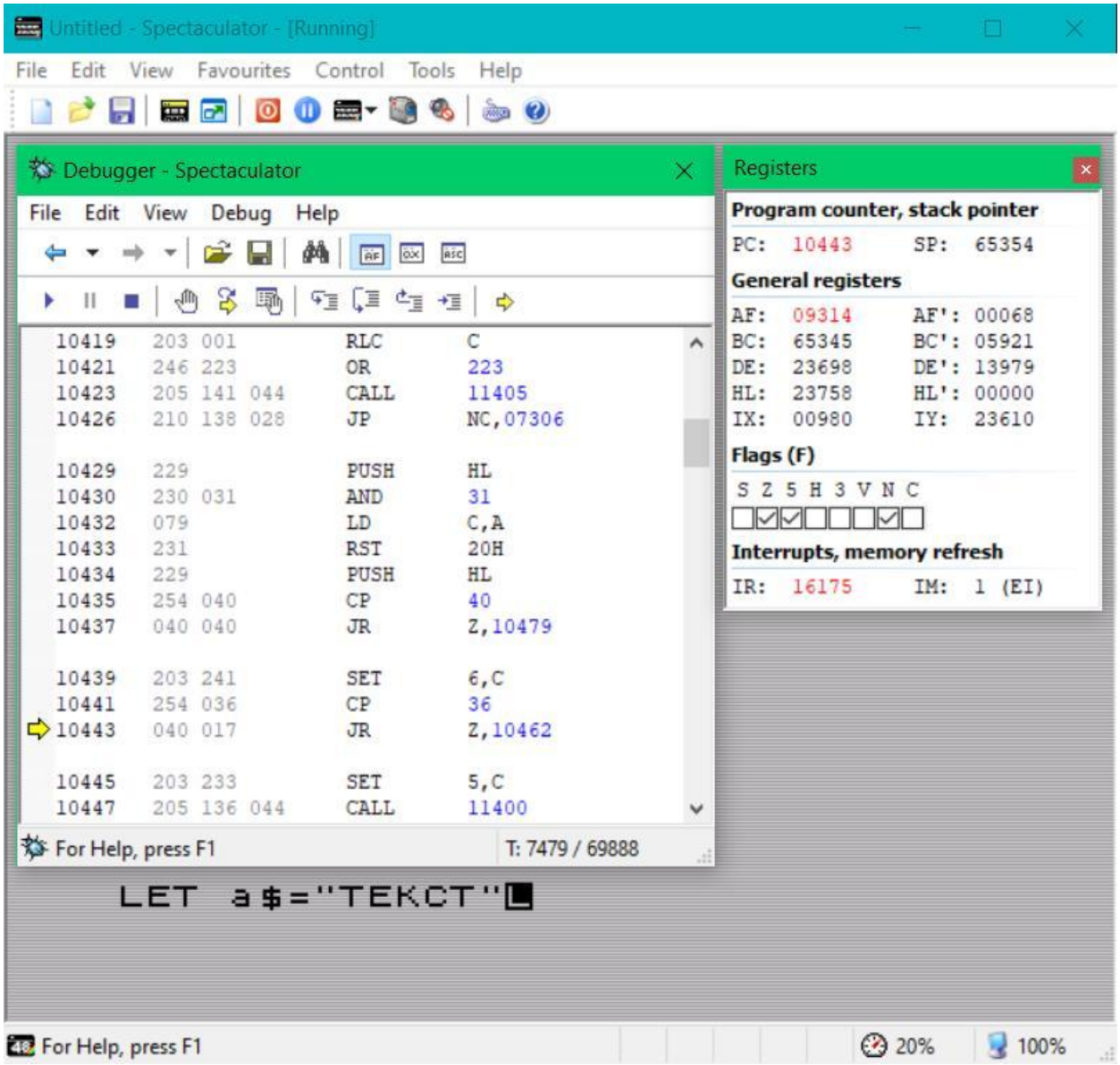


Рис. 133. Программа LOOK VARS. Отсортировка типа переменной.

Именно она и подсказывает, каким образом формируются основные переменные. Установив бит-тумблер №6 FLAGS (23611), для обработки числового выражения проверяется первый символ. Если он является кодом числового символа, то происходит

выход в главный цикл и редактор для исправления ошибки, потому что любая переменная должна начинаться с буквы.

В случае если символ буквенный, выполнение программы продолжается. Убираются три верхние бита символа буквы. Таким образом, буквы любого регистра становятся числовым кодом от 1 до 26 в алфавитном порядке (A или a = 1…Z или z = 26).

Смотрится следующий символ и по его характеру происходит рассортировка на типы переменных с переходом на соответствующие программы формирования массивов.

В зависимости от типа, к коду буквы добавляется код типа переменной:

+64 – однобуквенная символьная переменная (65-90)
+96 – однобуквенная числовая переменная (97-122)
+128 – однобуквенный числовой массив (129-154)
+160 – многобуквенная числовая переменная (161-186)
+192 – однобуквенный символьный массив (193-218)
+224 – однобуквенная переменная команды FOR или последняя буква многобуквенной числовой переменной (225-250).

Итак, признаком символьной переменной будет число от 65 до 90 в первом байте переменной по адресу, на который указывает VARS. Проще принять, что символьная переменная начинается с заглавных букв A-Z. Структура строки в общем виде будет выглядеть так:

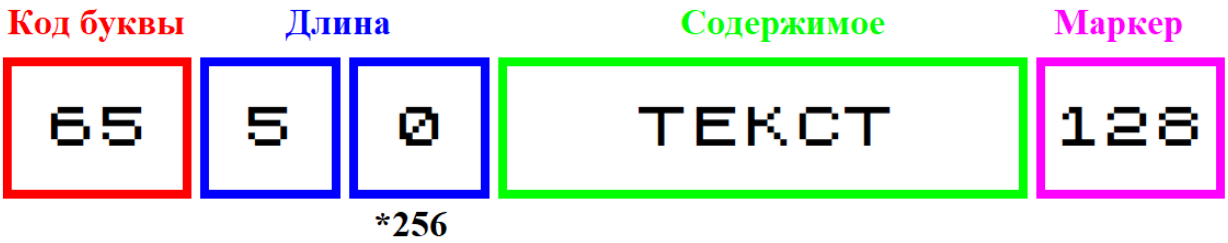


Рис. 134. Структурная схема символьной переменной.

Выглядит не очень хитро и сложно. Предлагаю ввести в область переменных результат работы цветной BASIC строки:

```
LET a$="СТАС РАССЕКАЕТ НА МЕРСЕ В ОЗЕРКАХ"
```

Для того, чтобы вклинить переменную в нужное место, снова придётся двигать нижестоящие области. На этот раз к трём предыдущим добавляется E_LINE (23641), расположенная сразу за переменными. Туда же должна указывать и следом стоящая переменная K_CUR (23643) иначе случится кое-что нехорошее в виде глухого зависания.

Никаких новых команд и алгоритмов тут уже не будет. Программа на алгоритмическом языке выглядит достаточно просто, только не забывайте, что все буквы латинские:

```
Debugger
Dec
Go To 23641
23641 ← 23794 23794
23749 ← 23796 23796 23796
23755 ← 65 35 0 16 3
23760 ←СТАС РАССЕКАЕТ НА МЕРСЕ В ОЗЕРКАХ
23793 ← 128 13 128
Trace
```

Запустив программу, отладчик закрылся и вроде бы ничего не произошло. На экране продолжает висеть заставка. Для проверки наберите с клавиатуры команду `PRINT a$` или `PRINT A$` любым удобным способом:

Рис. 135. Ввод команды для проверки синтезированной переменной. Фрагмент нижней части экрана.

Вводите и убедитесь, что такой нехитрой программой вы только что создали настоящую текстовую переменную:

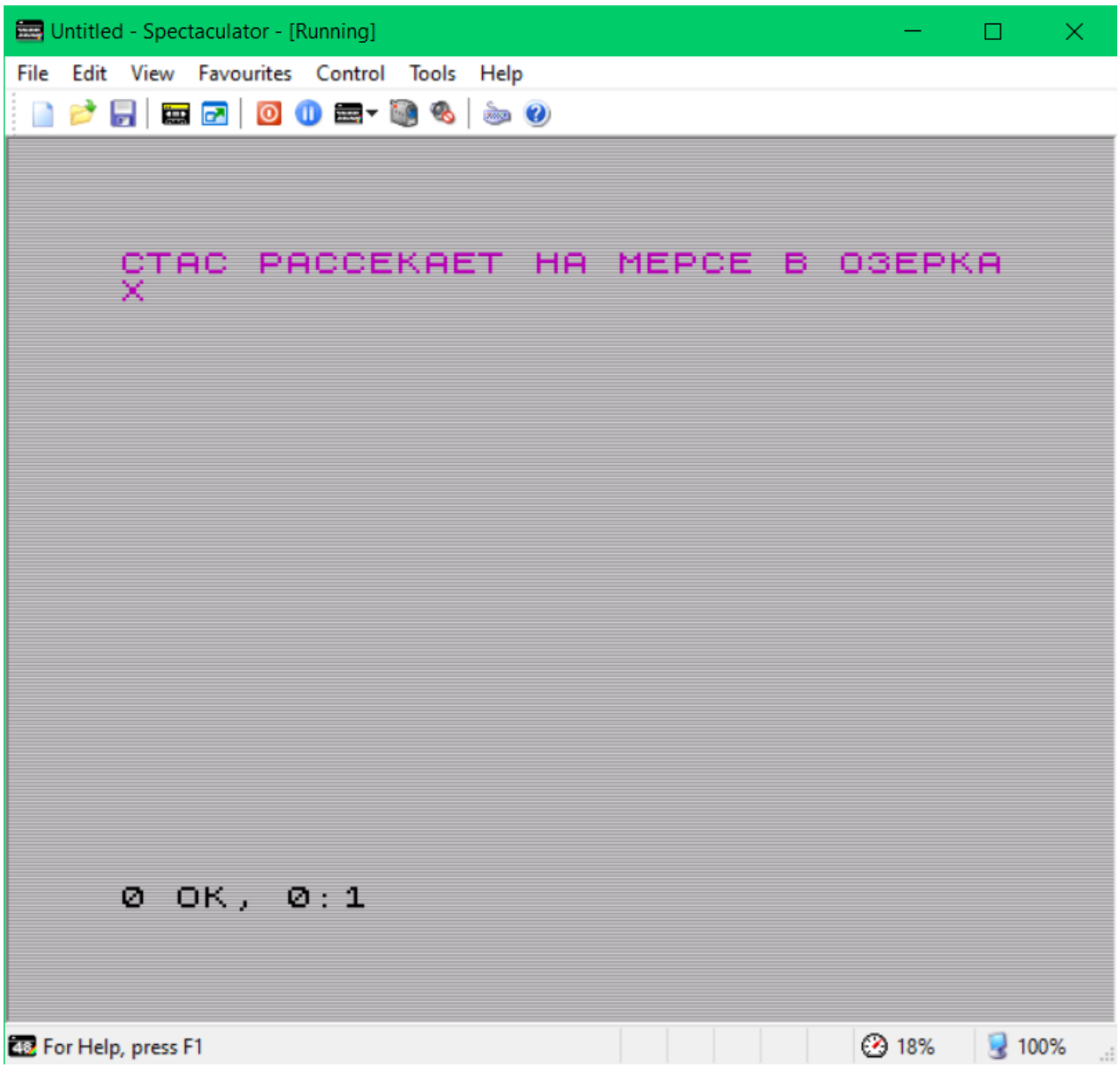


Рис. 136. Вывод синтезированной переменной натуральным методом по команде PRINT a$.

Вот, собственно, и весь вопрос.

Можно усложнить задачу к существующей символьной, добавить числовую переменную. Ведь в реальных BASIC программах, разные типы переменных могут десятками стоять друг за другом.

В прошлом примере скорость движения автомобиля так и не была раскрыта. Этот важный вопрос нужно выяснять. Правило «90-60-90», касается простых смертных на «Жигах» и «Москвичах». Дачники на «Запорах» в свои садоводства за Зеленогорск ездят по линейной схеме «40-40-40». Это явно параметры не для «Нового Русского» на Мерседесе, который гоняет по Выборгскому району Санкт-Петербурга. Как минимум по «Просвету» или Луначарского ночью можно втопить под 160, а то и более. На худой конец, соточку по трамвайным путям залудить. А на недовольных в багажнике всегда имеется спортинвентарь профессионального бейсболиста.

Вернёмся из счастливых девяностых к программированию. Следом за текстовой переменной, введите многосимвольную числовую, которая будет эквивалентна работе BASIC строки:

```
LET СКОРОСТЬ=160
```

В первую и последнюю букву таких комплексных имён подмешивается маркер начала и конца. В данном случае, к первой букве имени нужно добавить число 160, к последней 224. Числовое значение переменной также должно быть закодировано в рыхлом 5-ти байтном виде. Сейчас подробно на нём останавливаться не буду. Тем более, что в справочниках есть сносная раскладка структуры таких чисел.

Не стирая предыдущую, добавите и запустите следующую программу:

```
Debugger
Dec
Go To 23641
23641 ← 23807 23807
23749 ← 23809 23809 23809
23793 ← 163 107 111 112 111 99 116 226 0 0 160
23804 ← 0 0 128 13 128
Trace
```

После того, как отладчик закрылся, обновите экран вводом ENTER (23611 ← 32) и для самоконтроля наберите обычным способом следующую строку:

```
PRINT a$: PRINT СКОРОСТЬ
```

Рис. 137. Подготовка к интеграции и проверке дописанной многобуквенной переменной.

Введите:

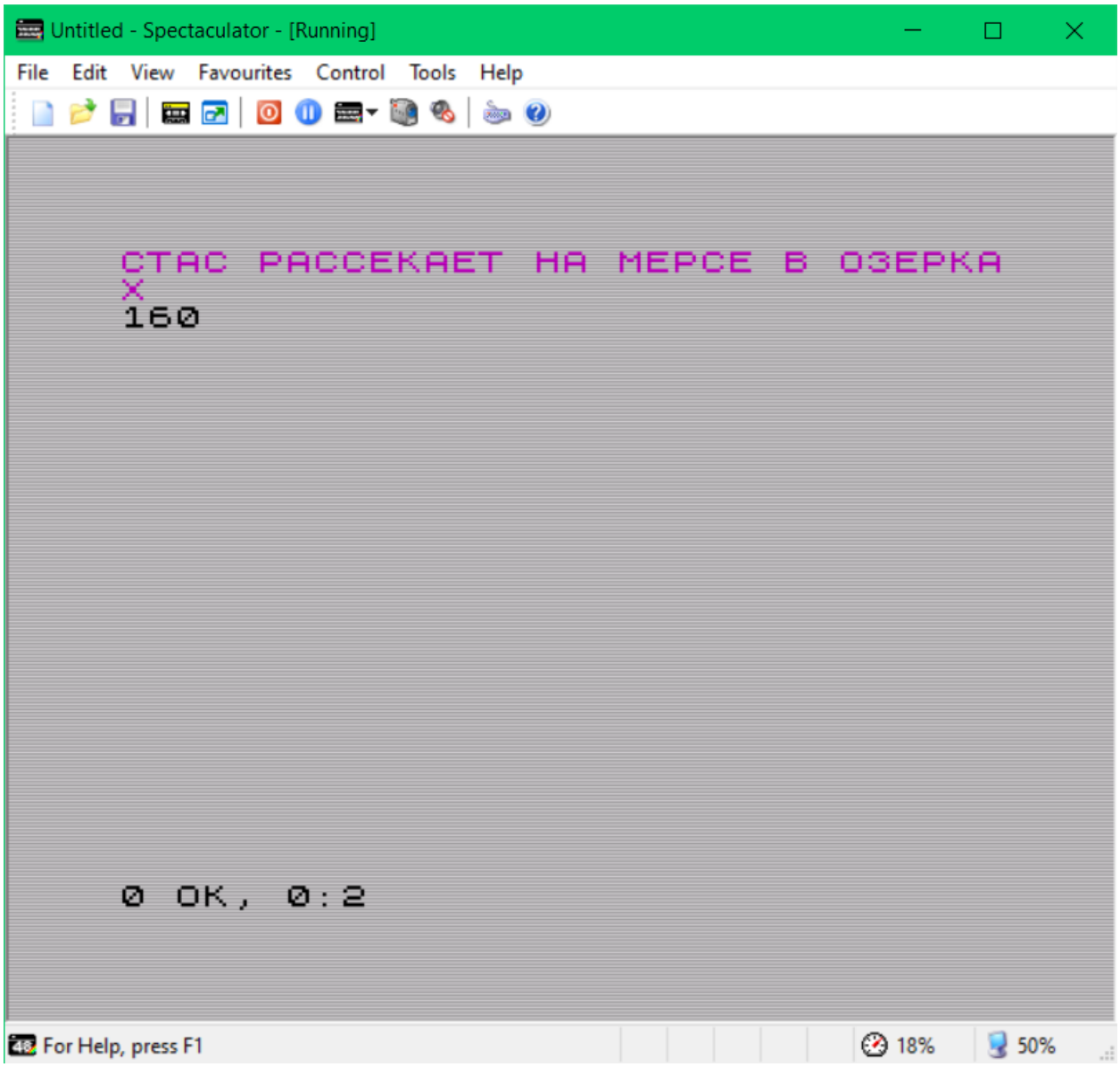


Рис. 138. Вывод на экран многобуквенной числовой переменной, добавленной искусственным способом.

Обе переменные вывелись в столбик. Для очистки области можно обойтись без сброса. Достаточно набрать и ввести `RUN` и вся область готова к экспериментам с очередными типами переменных.

Из этого комплекта осталось рассмотреть однобуквенную числовую переменную. Пусть это будет результат выполнения строки:

```
LET a=69
```

Одиночные числовые переменные имеют номера в интервале 97-122 и соответствуют строчным буквам a-z. Соотносимые к ним числа сохраняются в «рыхлом» 5-ти байтном распакованном виде. Учитывая вышесказанное, последовательность для «`LET a=69`» будет 97 0 0 69 0 0:

```
Debugger
Dec
Go To 23641
23641 ← 23762 23762
23749 ← 23764 23764 23764
23755 ← 97 0 0 69 0 0 128 13 128
Trace
```

Запустите программу, а следом наберите `PRINT a`:

Рис. 139. Подготовка к проверке вывода синтезированной числовой переменной. Фрагмент низа экрана.

Вводите строчку:

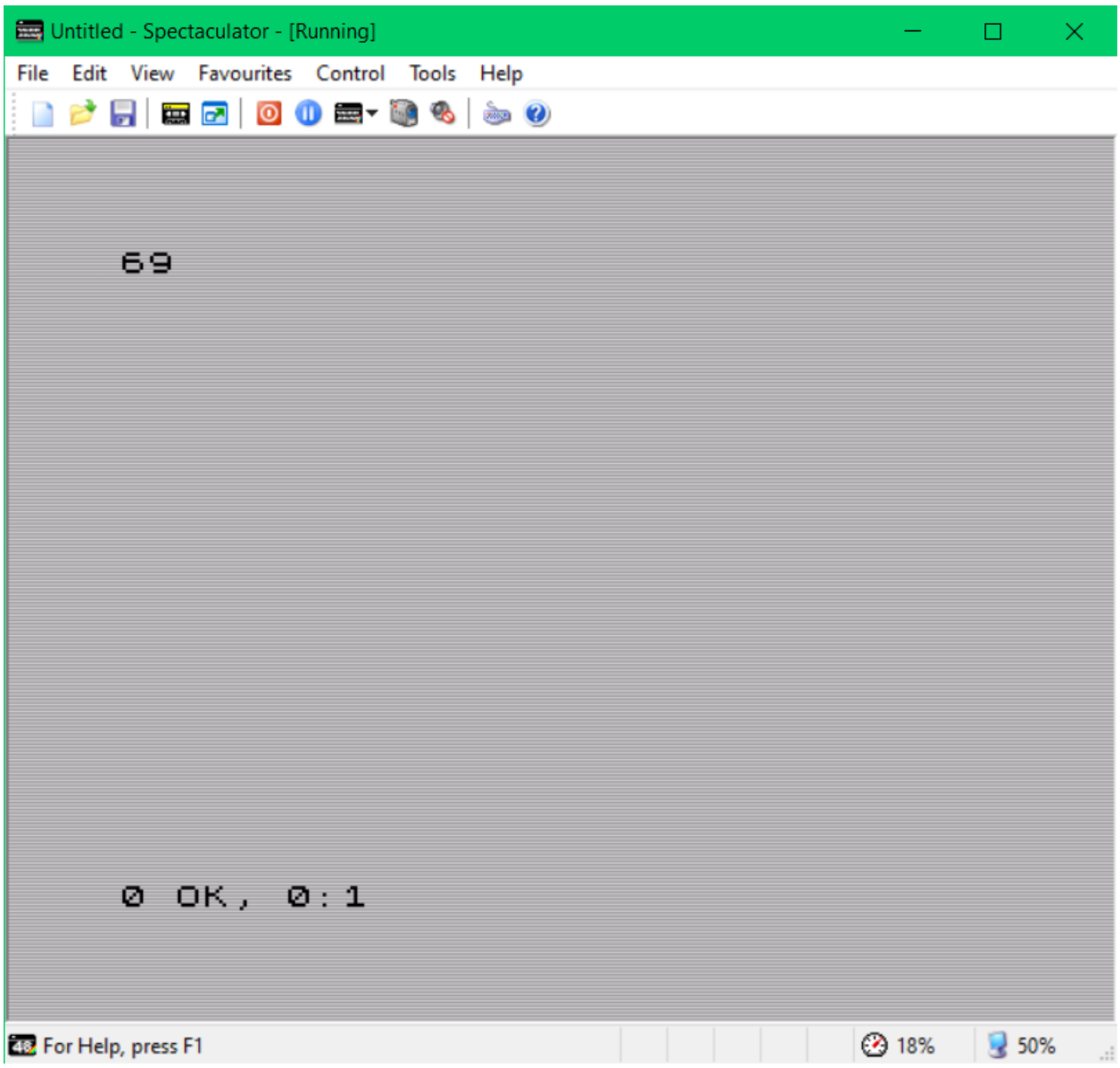


Рис. 140. Вывод синтезированной однобуквенной переменной.

А вот теперь можно смело нажимать Reset и приступать к массивам данных.

Первая буква символьного массива получается путём добавления числа 192, поэтому она будет иметь код от 193 по 218. Его структура на примере простейшего массива

```
DIM a$ (1,3):LET a$(1)="КС3"
```

будет выглядеть следующим образом:

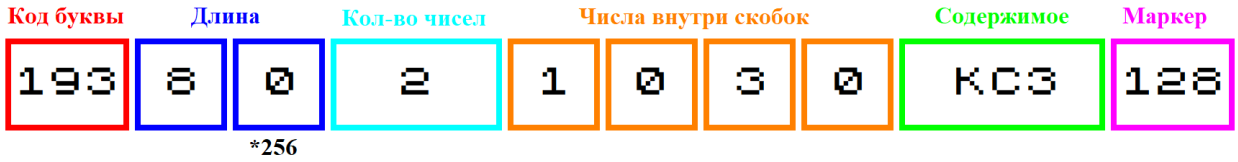


Рис. 141. Схема символьного массива.

где:
193 – код массива +192 с первой буквой алфавита «а» (192+1=193)
8 0 – общая длина чистого массива, начиная со следующего байта и до маркера 128.
2 – количество чисел внутри скобок, разделённых запятыми.
1 0 3 0 – сами числа без запятых, по 2 байта на каждое (их может быть 3 и более).
75 67 51 – тело массива с символами (по умолчанию после DIM заполнено пробелами).
128 – маркер конца области.

Для примера предлагаю эквивалент работы более сложной программы:

```
DIM a$(2,6): LET a$(1)="▒▒▒▒▒▒": LET
a$(2)="▒▒▒▒▒▒"
```

На алгоритмическом языке она выглядит более простой:

```
Debugger
Dec
Go To 23641
23641 ← 23776 23776
23749 ← 23778 23778 23778
23755 ← 193 17 0 2 2 0 6 0
23763 ← 16 1 99 97 103 111 16 2 77 65 51 79 128
23776 ← 13 128
Trace
```

Введите и запустите программу. Как обычно после выхода из отладчика «ничего не происходит». Для теста содержимого массива наберите следующую строку натуральным способом:

```
PRINT a$(1); "-"; a$(2)
```

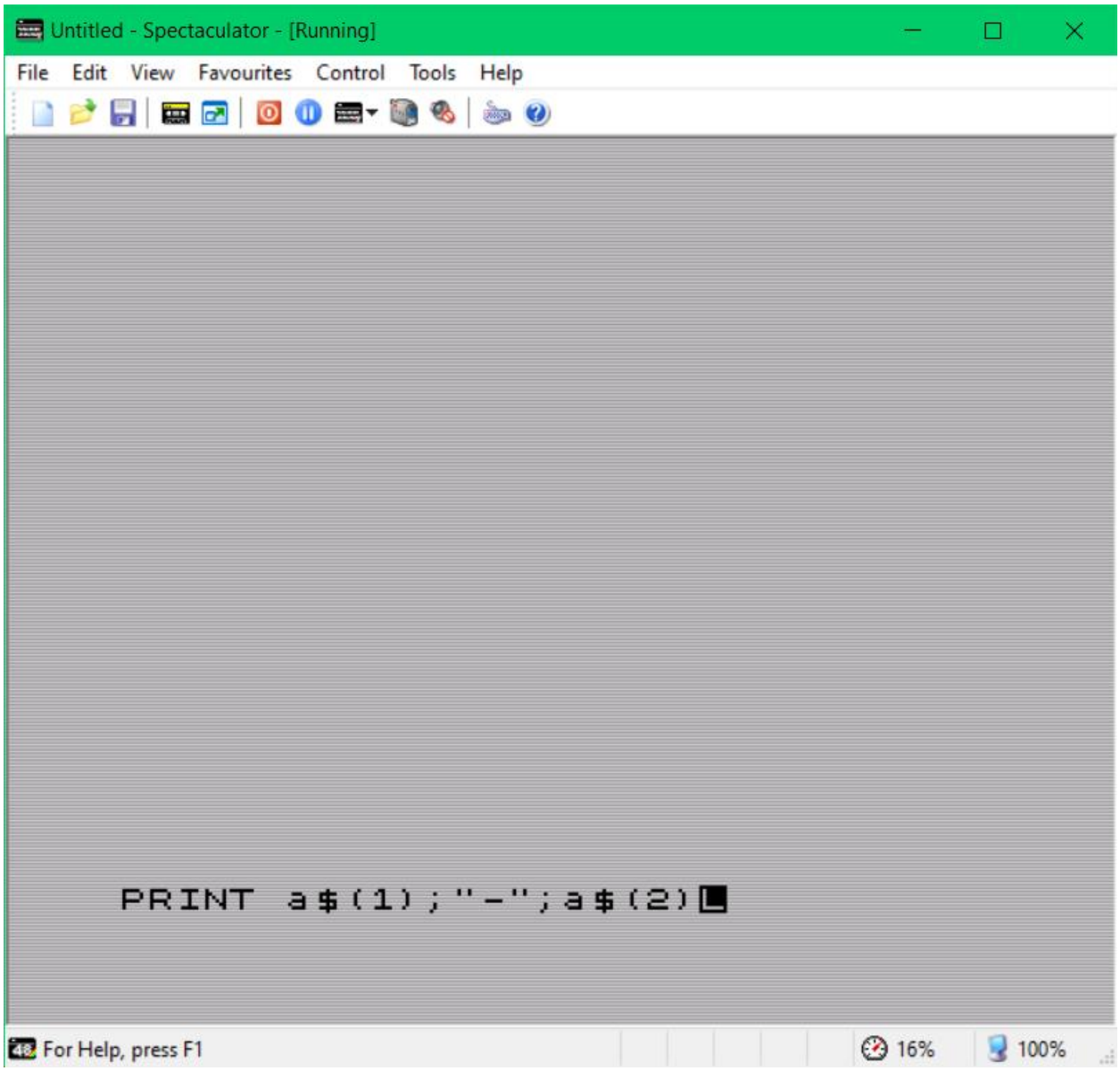


Рис. 142. Набор BASIC строки натуральным способом для проверки искусственного символьного массива.

На экране появилось….

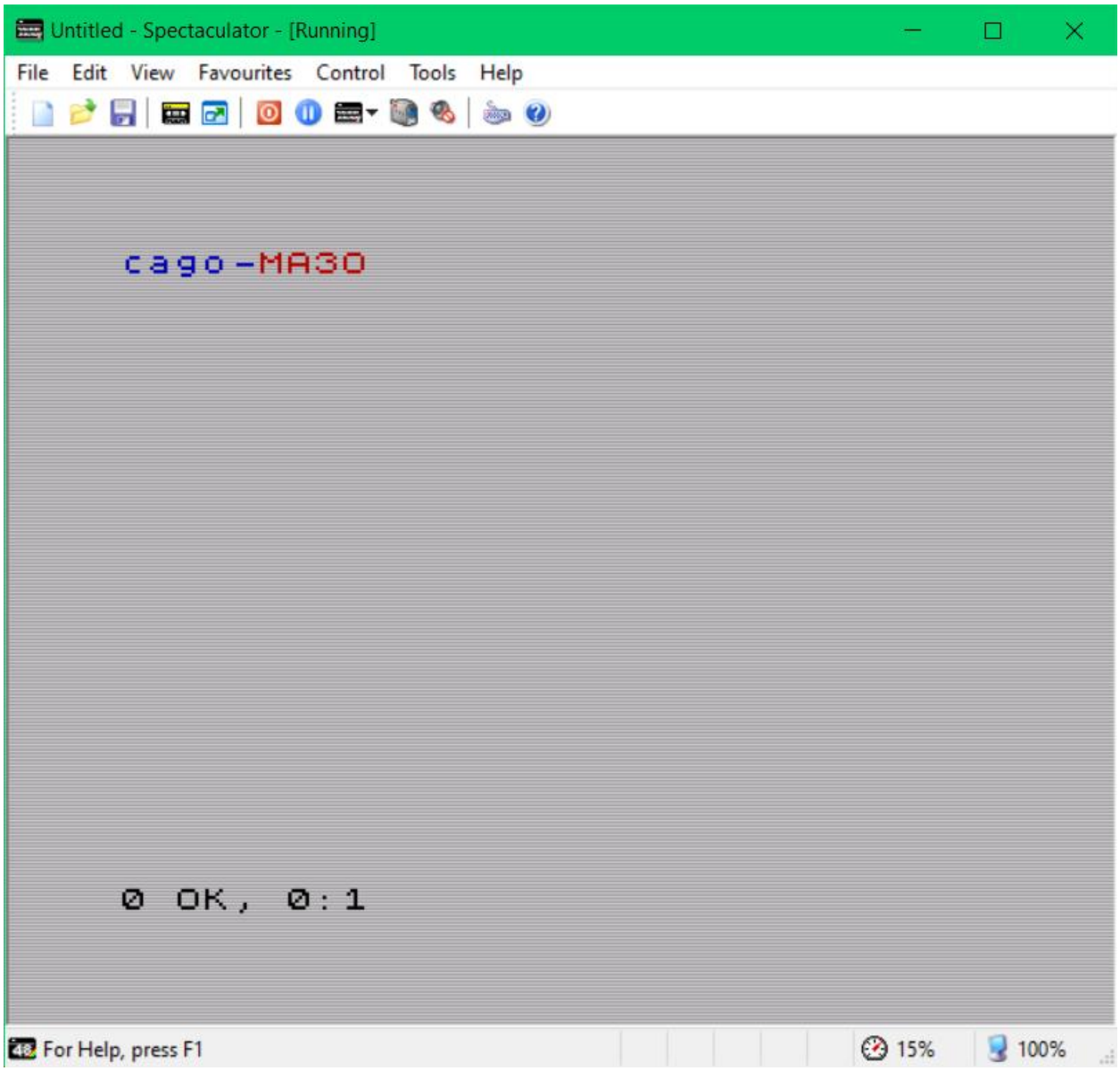


Рис. 143. Вывод синтезированного символьного массива на экран.

…до боли знакомое и щемящее душу слово с заключительной прогулки по Петербургу. Неожиданно? Хотя от меня всякого можно ожидать.

…В понедельник 19 сентября 2022 года за окном стояла приятная и пасмурная погода. Первый рабочий день после выходных. На улице никого. Красота, да и только, поэтому вместе с женой и собакой подорвались с самого утра и решили погулять по центру Петербурга. И вот идём по Садовой со своей огромной 1000 граммовой псиной, которая сидит на плече. Около «Апрашки» кипит культурная жизнь. Любовь, обожание, отдых все 24 часа – пожалуйста! Если вы вдруг хотите стать участником игры «Форт Баярд» и поискать сокровища пиратов – без проблем. А для ценителей традиционной культуры и серьёзных отношений тут всегда специальное предложение…

Рис. 144. Нелегальная трафаретная реклама на колонне ТК «Апраксин Двор». 19 сентября 2022 года.

Возвращаюсь из ~~царства БДСМ~~ воспоминаний о прошлой жизни к программе. Ничего не стирая, наберите такую строку:

```
    PRINT a$(1,3 TO 6): PRINT a$(1,1 TO
2);a$(2,3 TO 6)
```

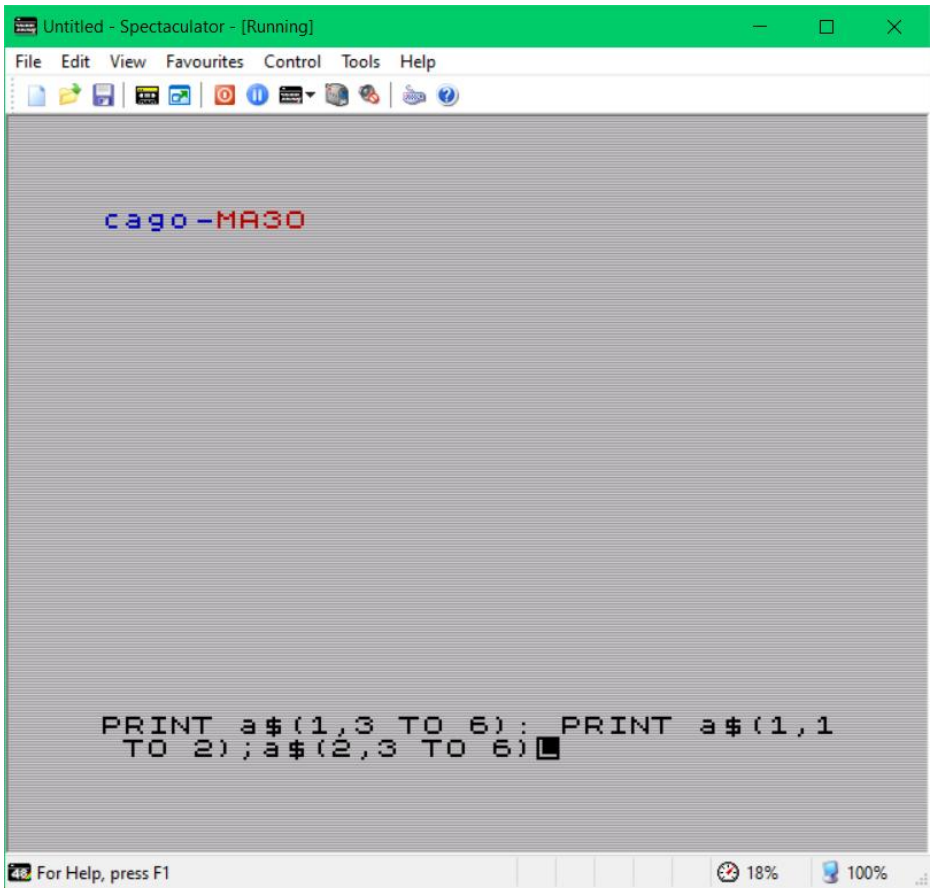


Рис. 145. Более глубокая проверка синтезированного символьного массива.

Введите:

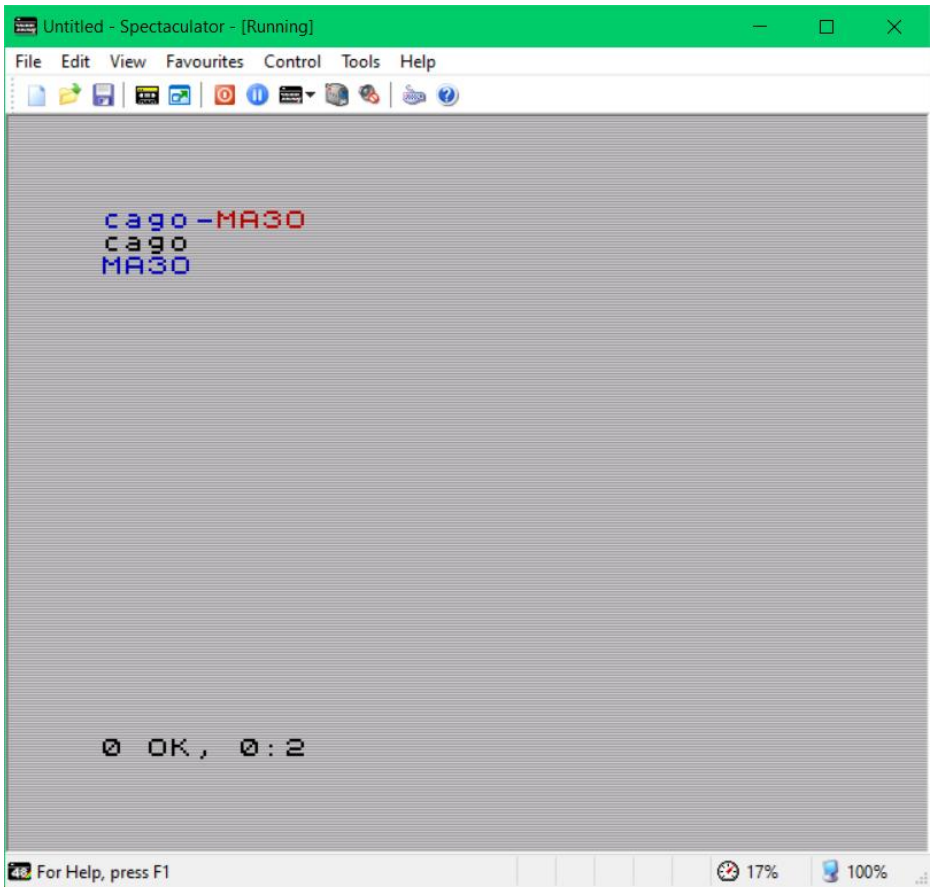


Рис. 146. Результат вывода текста и атрибутов из фрагментов символьного массива.

Обратите внимание, как выборочно можно извлекать из элементов массива не только текст, но и управляющие цветовые символы. Мало того, кусочки можно слеплять друг с другом.

Остался числовой массив. Его структура похожая и даже немного проще символьного. К коду буквы добавляется число «128». При задании массива, каждому числу выделяется по 5 ячеек, чтобы данные сразу размещать в распакованном формате.

Предлагаю воссоздать работу следующей BASIC строки:

```
DIM a (1,3): LET a(1,1)=58: LET a(1,2)=235: LET a(1,3)=202
```

Для этого наберите следующую программу:

```
Debugger
Dec
Go To 23641
23641 ← 23779 23779
23749 ← 23781 23781 23781
23755 ← 129 20 0 2 1 0 3 0
23763 ← 0 0 58 0 0 0 0 235 0 0 0 0 202 0 0 128
23779 ← 13 128
Trace
```

Выполните программу, и после выхода в *Spectaculator* натуральным способом наберите такие строки:

```
PRINT a(1,1);"-";a(1,2);"-";a(1,3)
```

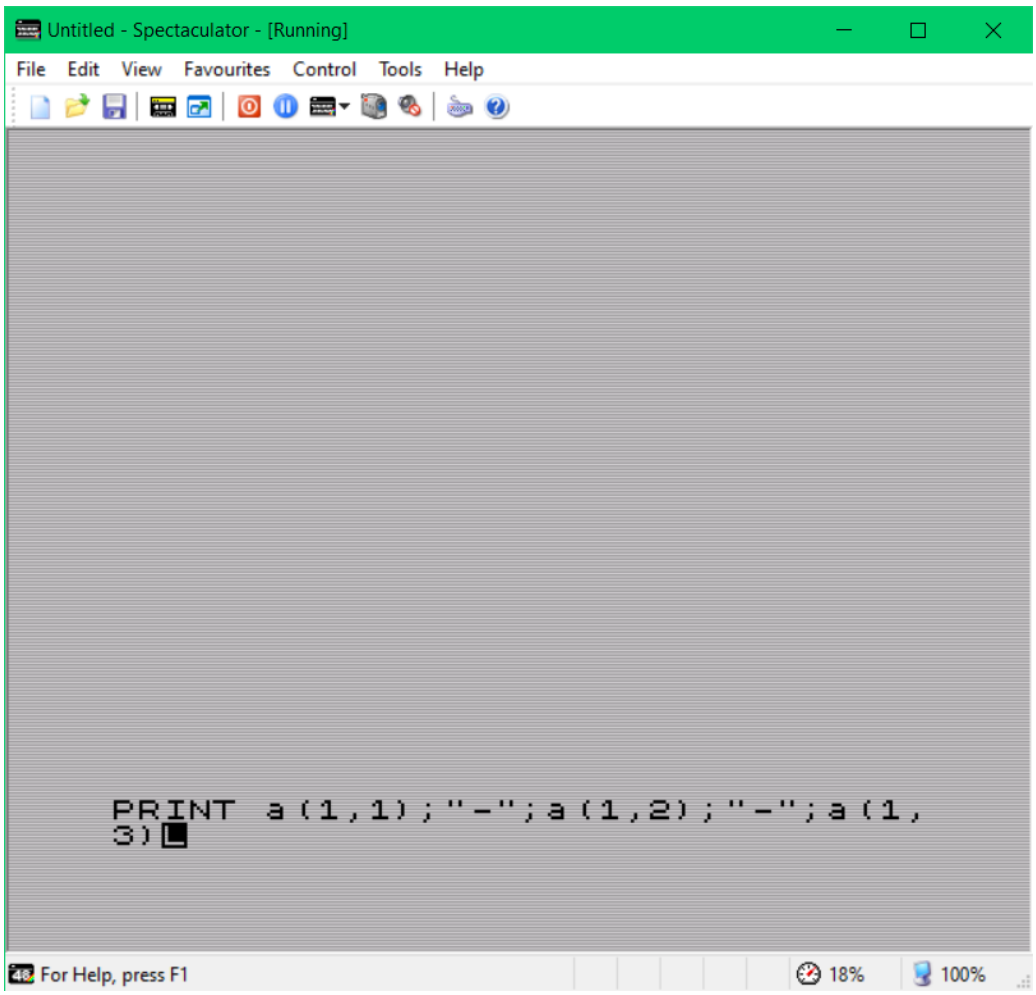


Рис. 147. Набор BASIC строки натуральным способом для проверки искусственного числового массива.

Вводите строку:

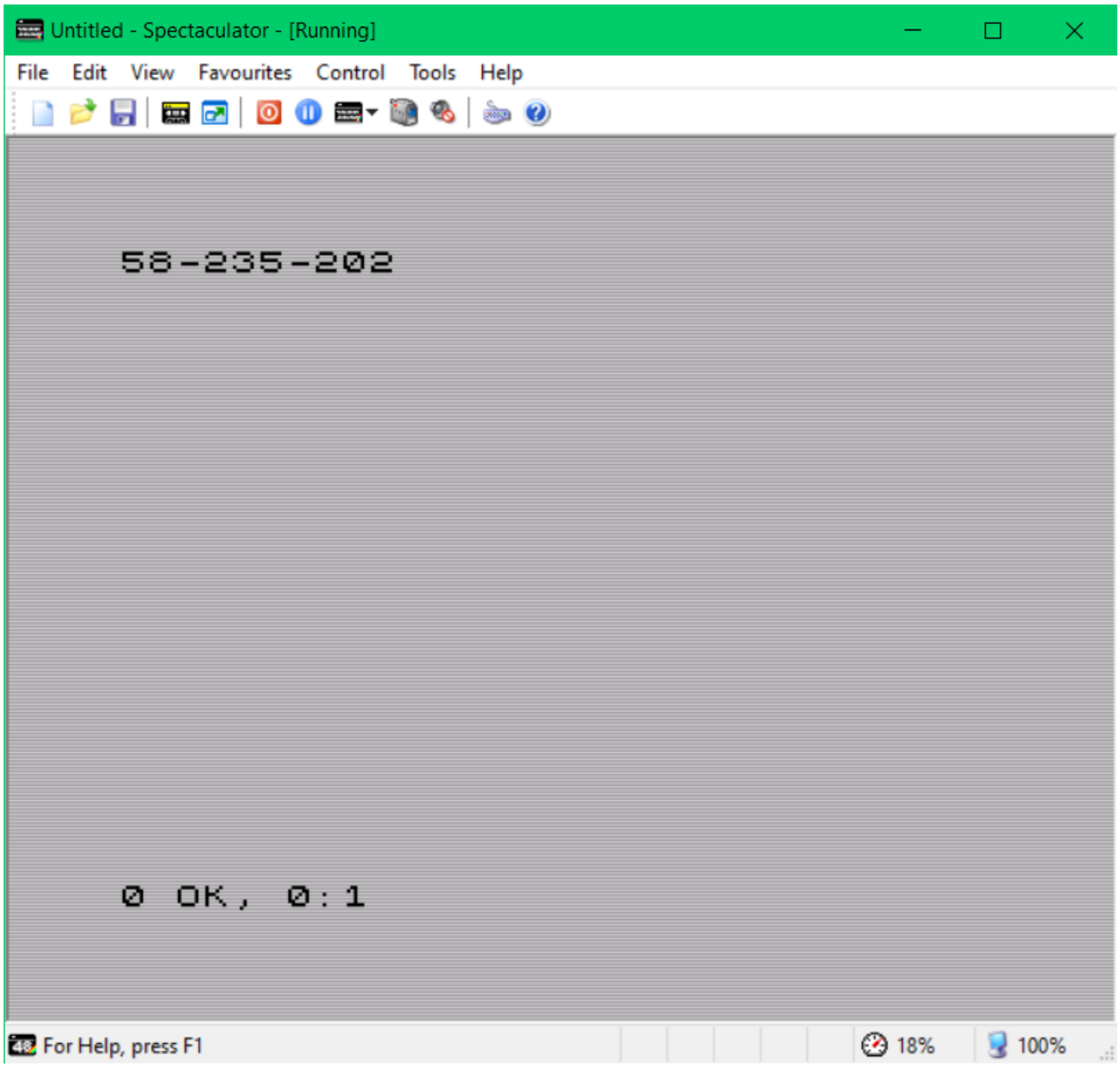


Рис. 148. Результат вывода значений из числового массива.

Как и ожидалось, через тире вывелось три каких-то числа из массива. Каких-то? Не Юрец, у тебя случайных чисел и текстов не бывает! Телефон отдела сбыта обдолбэ-самоучителей?

Ладно, уговорили, не случайное. Переведите каждое отдельное число в HEX-формат. Итак, 58=3A. 235=EB, ну а 202 это CA. А теперь поставьте их рядом:

3AEBCA

А что значит «3AEBCA», А это значит, что не только у Юрца всё Заебца, но и у вас всё «3AEB0C» (Заебок) с освоением массивов. На этом торжественном моменте предлагаю закончить главу.

Глава 13
Знакомство с номерными BASIC-строками

Краткое содержание: номера строк, формат BASIC строки, механизм записи строк

Помимо одноразовых выполняемых строк, существуют многоразовые строки с номерами, которые при вводе не выполняются, а остаются на экране. Такие строки можно запускать без потерь любой командой и сколько угодно раз. Они размещаются в области, на которую указывает набор ячеек `PROG` по адресу `23635`.

Прежде, чем приступить написанию строк BASIC программы, предлагаю ненадолго вернуться к ранним главам и вспомнить, что происходит с набранным текстом.

Сырая символьная строка вводится в область, указанную переменной `E_LINE (23641)`, через буфер редактора. В анализаторе она преобразуется в исполняемую BASIC строку и направляется в сортировочный центр. Вызывается программа определения номера строки `E-LINE-NO (6651)`.

В случае, если первым стоит исполняемый оператор, в «BC» возвращается пустота, то есть «0». Но есть нюанс, а скорее недоработка создателей прошивки: кроме пустоты, тот же самый ноль в «BC» возвращается и при считывании символов «0» перед оператором. Таким образом, 0 в данном случае означает два варианта: строка не имеет номера или строка имеет коды символов числа «0» перед командой. Если перед оператором имеется хоть одна цифра, отличная от 0 (09, 087, 0003, 0020, 0205 и т. д), то в «BC» возвращается итоговый номер, скомпилированный из сырых числовых символов. Стоящие впереди незначащие нули при этом отбрасываются:

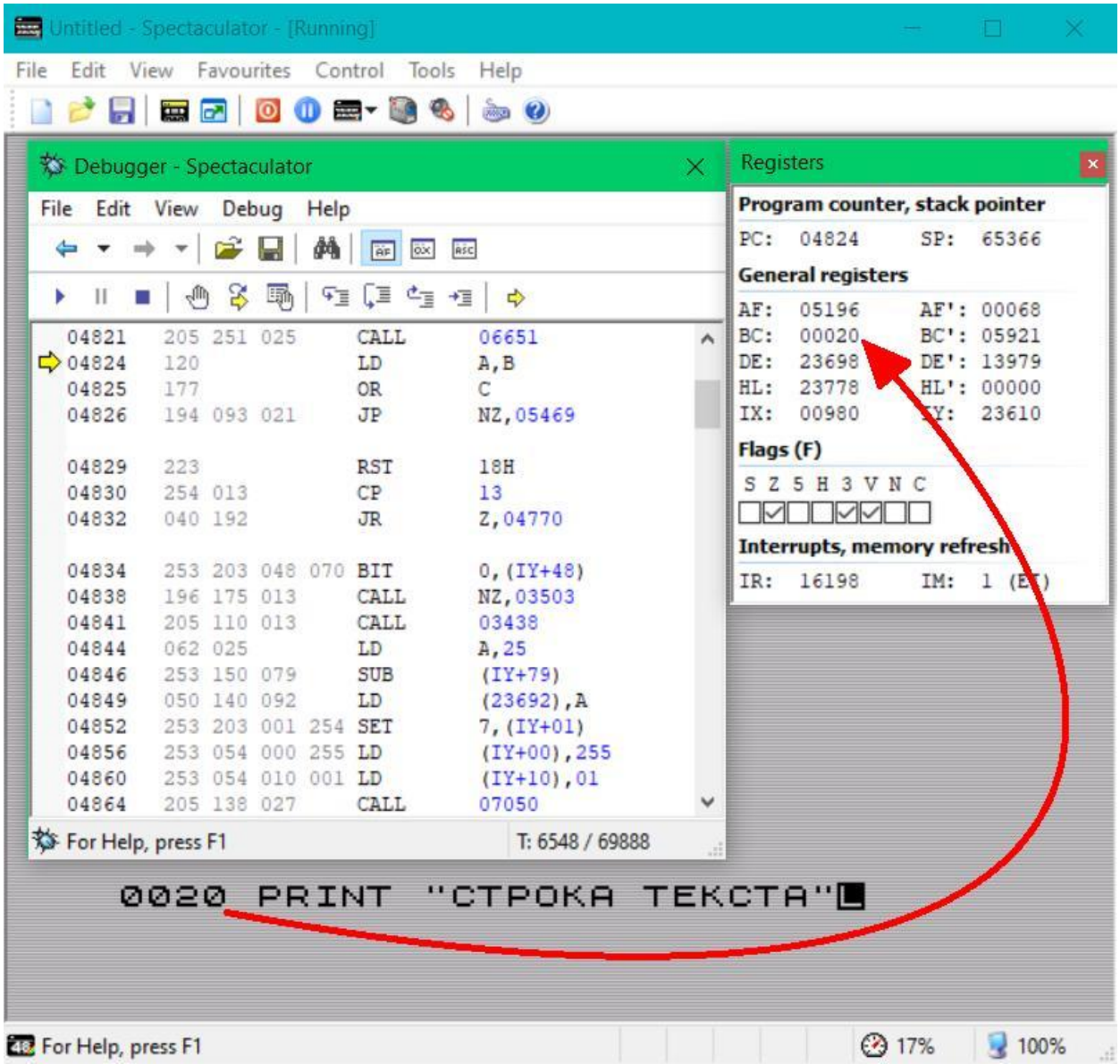


Рис. 149. Определение номера строки и фильтрация на значение «0».

Следом в 4824 стоит примитивный разделитель для отсева на категории (B OR C). Если значение строки «0», то по убеждению разработчиков, нумерация отсутствует и начинается выполнение содержимого. Если вернулось число от «1» и выше, то это строка с номером, и приготовлена для записи в память. Нули разберу чуть позже, а сейчас хочу обратить внимание на строку с любым номером отличным от «0».

Покинув сортировочный центр `MAIN-3 (4815)`, номерная BASIC строка попадает в центр проверки и учёта номеров `MAIN-ADD (5469)`. По прибытию номер строки из «BC» переписывается в переменную `E_PPC (23625)`. Из внутреннего курсора `CH_ADD (23645)`, транзитом через «HL», в «DE» записывается адрес первого символа в BASIC строке, идущего за сырым номером программы.

Поскольку программная строка не рассчитана на выполнение, то из таблицы запоминается всего один вариант сообщения об ошибке: «`G No room for line`». Оно активируется в случае, если для очередной запоминаемой BASIC строки не хватит места в отведённых границах памяти.

В «HL» берётся значение `WORKSP (23649)`, которое к этому моменту указывает за область набранной строки. Вычитая из «DE», получается длина текущей строки вместе с клавишей `ENTER`, но без учёта номера, и запоминается на будущее. Скопировав из «BC» в «HL» номер текущей строки, подпрограммой `LINE_ADDR (6510)` отправляется запрос на проверку его индивидуальности.

На выходе в «HL» вместо номера возвращается адрес начала BASIC-строки. Если строка с таким номером уже существует, то к адресу прилагается включённая Z ☑ галочка «Z» и нужно готовиться к замещению строк. А если строк с таким номером не нашлось, то дальнейшие действия пропускаются, и начинается подготовка строки к записи в память.

Начало старой строки с одинаковым номером есть, и нужно найти её конец. Для этого запускается подпрограмма `NEXT ONE (6584)`, которая в «DE» возвращает адрес начала следующей, за концом повторившейся строки.

Таким образом, в «HL» получается начало, а в «DE» конец строки. С этими вводными данными следом запускается подпрограмма `RECLAIM-2 (6632)`, которая удаляет старую строку в области `PROG` [23635], подтаскивая нижние на освободившееся место. Область `PROG` уменьшается, и её новое окончание корректируется в системной переменной `VARS (23627)`:

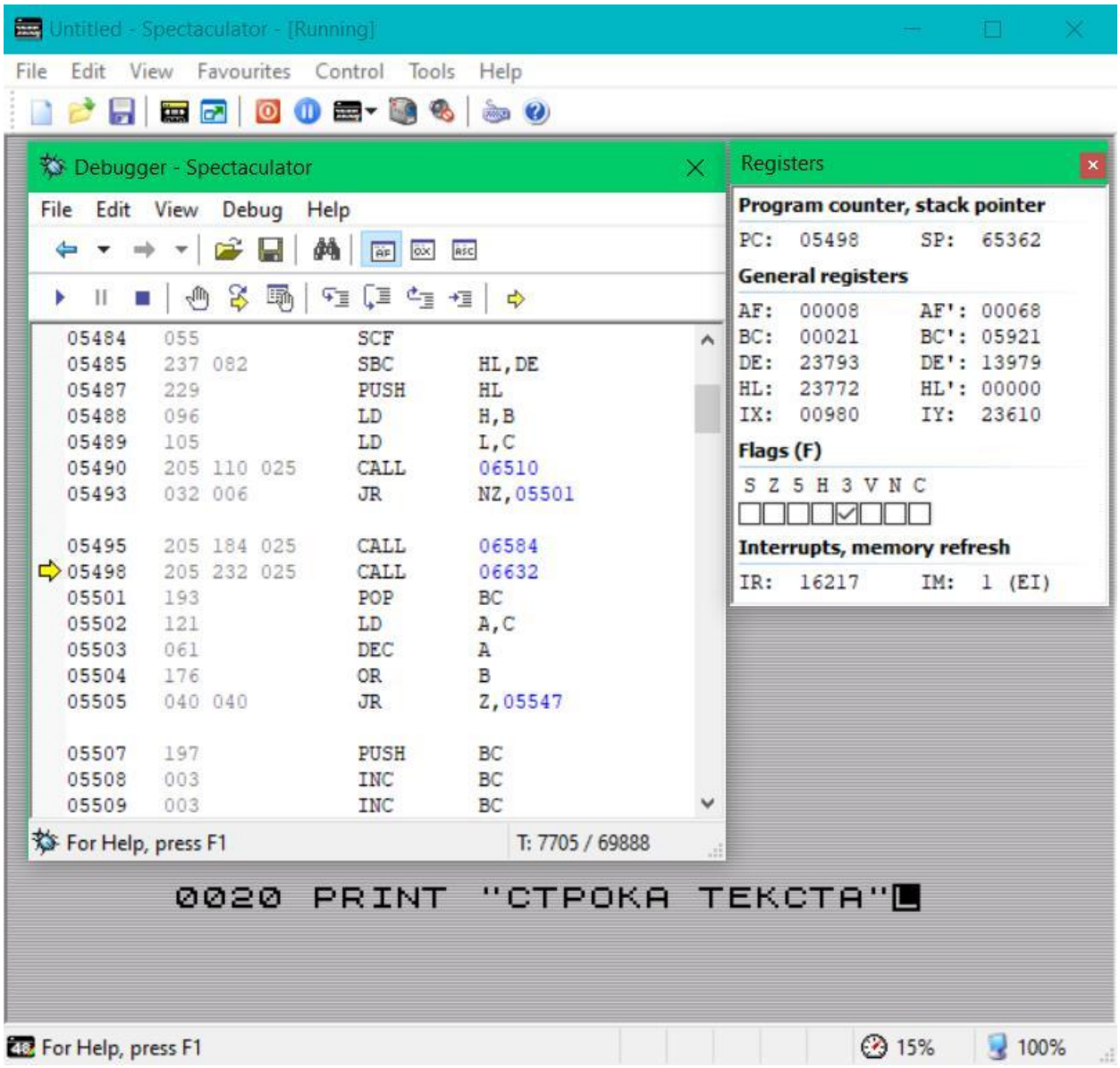


Рис. 150. Процесс корректировки места для ввода новой строки.

С адреса 5501 начинается добавление новой строки в программу. Вспоминается длина текущей строки без полуфабрикатного номера. Если она оказалась однобайтная, значит, в ней присутствовал только ENTER. Обнаружив строку-пустышку произойдёт возврат в самое начало главного цикла `MAIN EXECUTION (4770)` с последующим обновлением экрана авторапечаткой строками программы.

Если строка содержит длину более байта, значит, там имеется содержимое и нужно текущую строку поместить в память, но преобразовав её длину и номер в двухбайтный формат (не путать с 5-ти байтным).

В таком случае запоминается чистая длина строки и увеличивается на 4 байта с расчетом будущей длины с номером строки. Начинаются подготовительные работы по преобразованию.

Перед выделением памяти сохраняется переменная `PROG (23635)`, так как после работы, нижеследующая подпрограмма испортит значение этого важного указателя и не вернёт назад. Подпрограмма `MAKE ROOM (5717)` выделяет область памяти под новую строку.

Учитывая будущий порядковый номер вводимой строки, она вклинивается между строк и раздвигает имеющийся массив, но не затирает старые данные. После работы программы возвращается прежнее значение `PROG (23635)`.

Место появилось и теперь можно начинать перетаскивать туда новую строку из E_LINE (23641). В «BC» вспоминается длина строки без полуфабрикатного номера в символьном виде. В «DE» с прошлых операций уцелело начало следующей строки после вставляемой. Вычитанием единички получается адрес последней ячейки области памяти, куда нужно перетащить строку. В «HL» загружается WORKSP (23649). Вычитая маркер и первый байт, сформировывается адрес ENTER последнего байта вводимой строки из E_LINE (23641).

Снизу вверх из буфера редактора, в заготовленную дырку области PROG (23645) копируется свежая строка без полуфабрикатного номера. Сверху остаётся только 4 резервных байта для будущего номера и длины строки в двухбайтном формате.

Из переменной E_PPC (23625) узнаётся номер текущей строки программы, на которую указывает внешний курсор >. В самом начале программы туда засылался номер текущей обрабатываемой строки. В «BC» вспоминается длина новой строки. По полученным данным (длина/номер строки и адрес последнего байта) снизу вверх сформировывается длина и номер строки в конечном формате (последовательно записываются старший и младший байт длины строки, затем младший и старший байт номера строки). Для чего формат номера строки сформировался в зеркальном формате, по отношению к длине, детально разберу в следующих практических главах.

Новая или замещаемая строка записалась в память, но еще не отобразилась на экране. Осталось вернуться в самое начало главного цикла MAIN EXECUTION (4770):

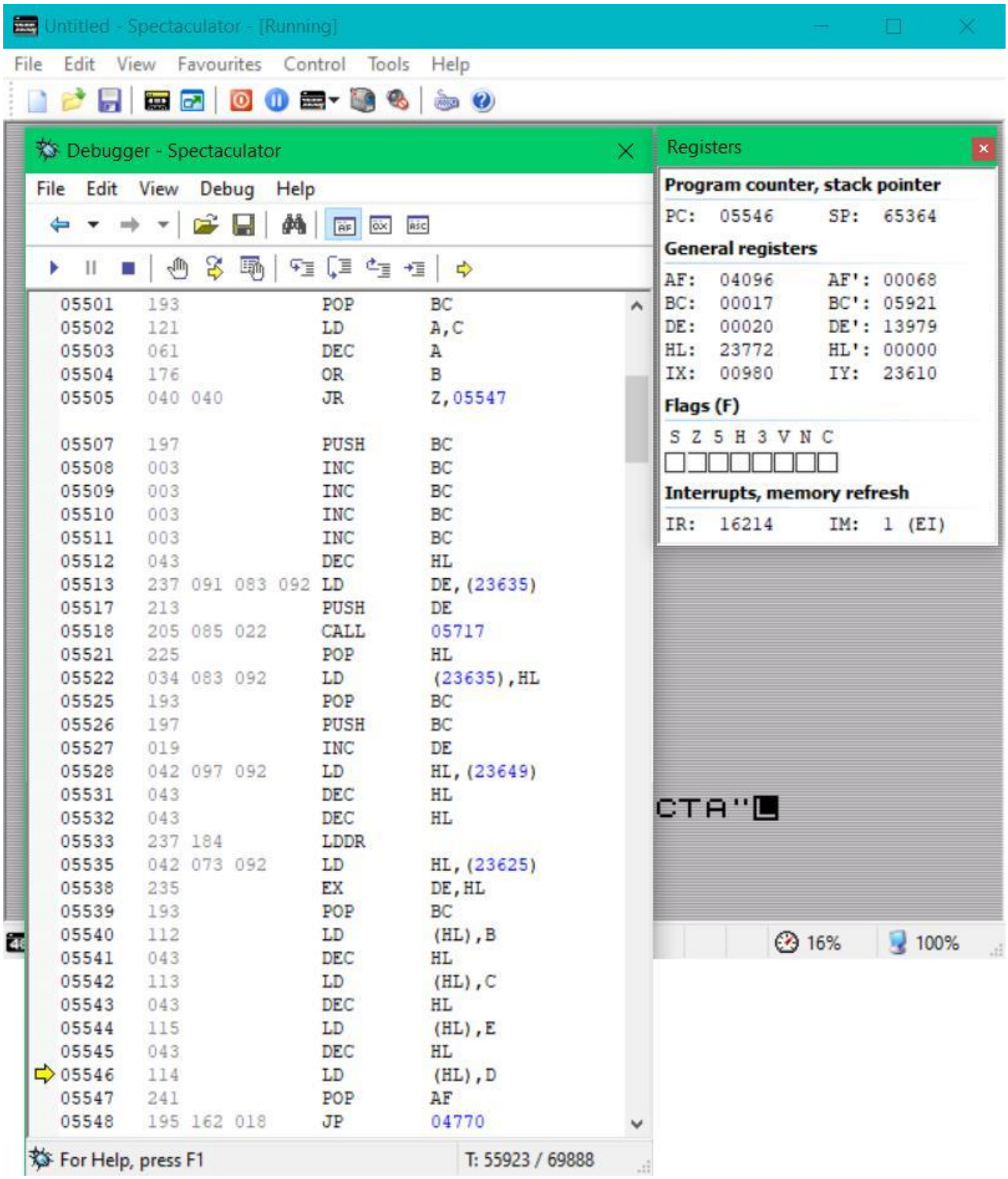


Рис. 151. Последние шаги перед возвратом в основной цикл и обновлением экрана.

В главной точке основной программы отрезаются две нижние строки под буфер `E_LINE (23641)` и выполняется очистка экрана с обновлением вывода строк программы. Настраивается канал на вывод текста в нижние строки экрана и снова вызывается огромная комплексная подпрограмма ввода и редактирования BASIC строки `EDITOR` по адресу 3884:

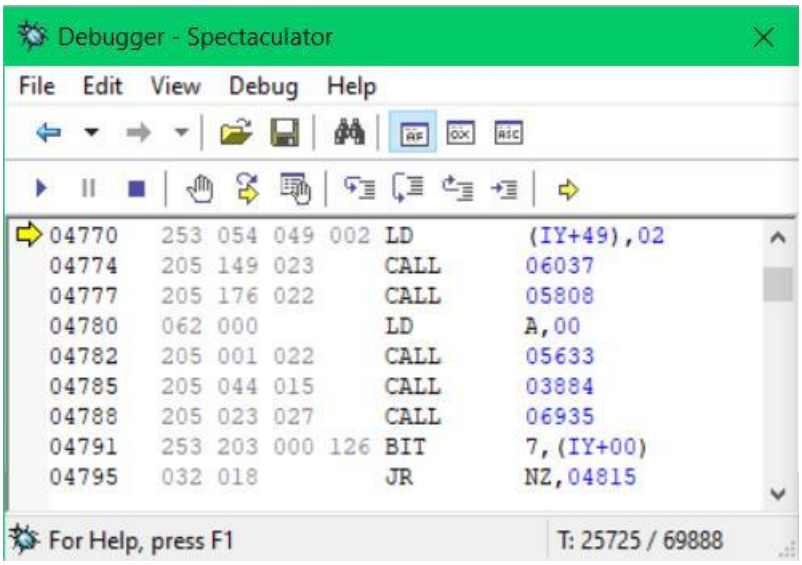


Рис. 152. Первый шаг в основной программе MAIN EXECUTION после возврата.

Полный круг работы главного BASIC цикла завершился.
Подведу итоги. Рассматриваемая в качестве примера строка:

```
0020 PRINT "СТРОКА ТЕКСТА"
```

потеряет два первых нуля и расположится в BASIC области так:

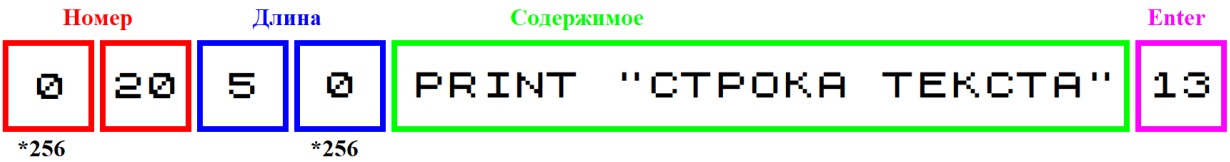


Рис. 153. Схема расположения BASIC строки в области PROG.

В заключение, как говорилось еще в одной культовой (как по опечаткам, так и по материалу) книге:

8.12 ЗАКЛЮЧЕНИЕ.

Теперь читатель готов к написанию программ в машинных кодах. Подпрограммы в машинных кодах могут использоваться в программе на бэйсике. Они вызываются в программе по команде "USR". Вся программа может состоять из нескольких подпрограмм, и, в конце концов, программа бэйсик будет состоять только из одной строки: 10 RANDOMIZE USR..... , так как никаких возвратов не будет сделано до тех пор, пока программа машинного кода не будет выполнена.

Рис. 154. Заключение из книги «Машинные коды».

Немного не так «Теперь читатель готов к написанию, а скорее искусственному синтезу, своей первой BASIC строки».

Глава 14
Редактирование нулевой строки или спаси тонущий пароход из Невы

Краткое содержание: строка 0, корректировка редактирования и ввода

Кратко ознакомившись с механизмом создания и структурой BASIC строк, предлагаю опробовать синтез полноценной номерной строки. На этот раз начну сразу, но как обычно с «н0000ля», а точнее с «н00000левой» строки.

Поскольку ввод строки будет производиться внешним методом, минуя проверку синтаксиса и преобразования, выставлять номер строки и её длину придётся самостоятельно. Еще к перекройке указателей `E_LINE (23641)`, `WORKSP (23649)`, `STKBOT (23651)` и `STKEND (23653)` добавится `VARS (23627)`. В него нужно внести адрес маркера «128», который будет стоять сразу за кодом `ENTER` (13), перед областью буфера вводимой строки.

Наберите и введите следующую программу на алгоритме «God Mode»:

```
Debugger
Dec
Go To 23560
23560 ← 13
23611 ← 32
23627 ← 23788
23641 ← 23789
23649 ← 23791 23791 23791
23755 ← 0 0 29 0 245
23760 ←""B HEBE Tepnum KPAX METEOP
23786 ← 34 13 128 13 128
Trace
```

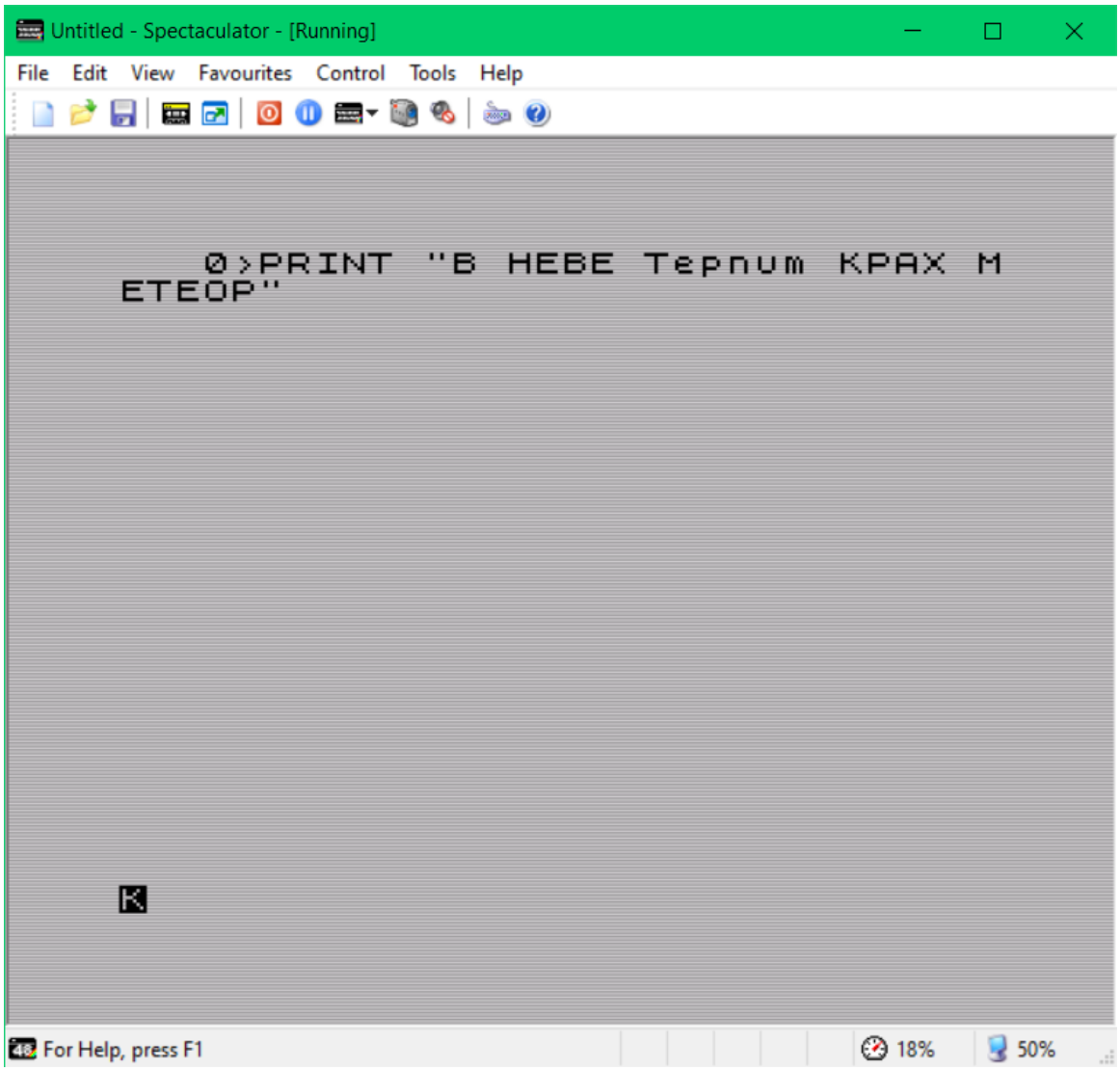


Рис. 155. Spectaculator. Искусственно синтезированная BASIC строка с номером 0.

– А сейчас посмотрите налево – монотонно тарабанит экскурсовод заученные фразы.

Туристы со скучающими лицами лениво поворачивают головы и смотрят в иллюминаторы. За бортом речного теплохода бодро проплывали дома по чётной стороне Дворцовой набережной.

– Зимний Дворец был построе...
– БДЫЩЩЩ! БУМ! БАХ!

С диким скрежетом судно подбросило вверх, а затем уронило набок. Над распластавшимися по всему салону людьми со скоростью реактивного самолёта летела женщина-экскурсовод.

Навалив на опору Дворцового моста, «Метеор» получил пробоину и начал своё медленное «титаническое» погружение в серые Невские воды. Неторопливо вращаясь вокруг своей оси, он быстро уплывал по течению Большой Невки в сторону причалов Балтийского завода...

На экране появилась полноценная искусственно сгенерированная BASIC строка с текстом. Поскольку она имела номер 0, внешний курсор ▝ автоматом встал на созданную строку. Натуральным способом введите команду RUN, чтобы проверить работоспособность программы:

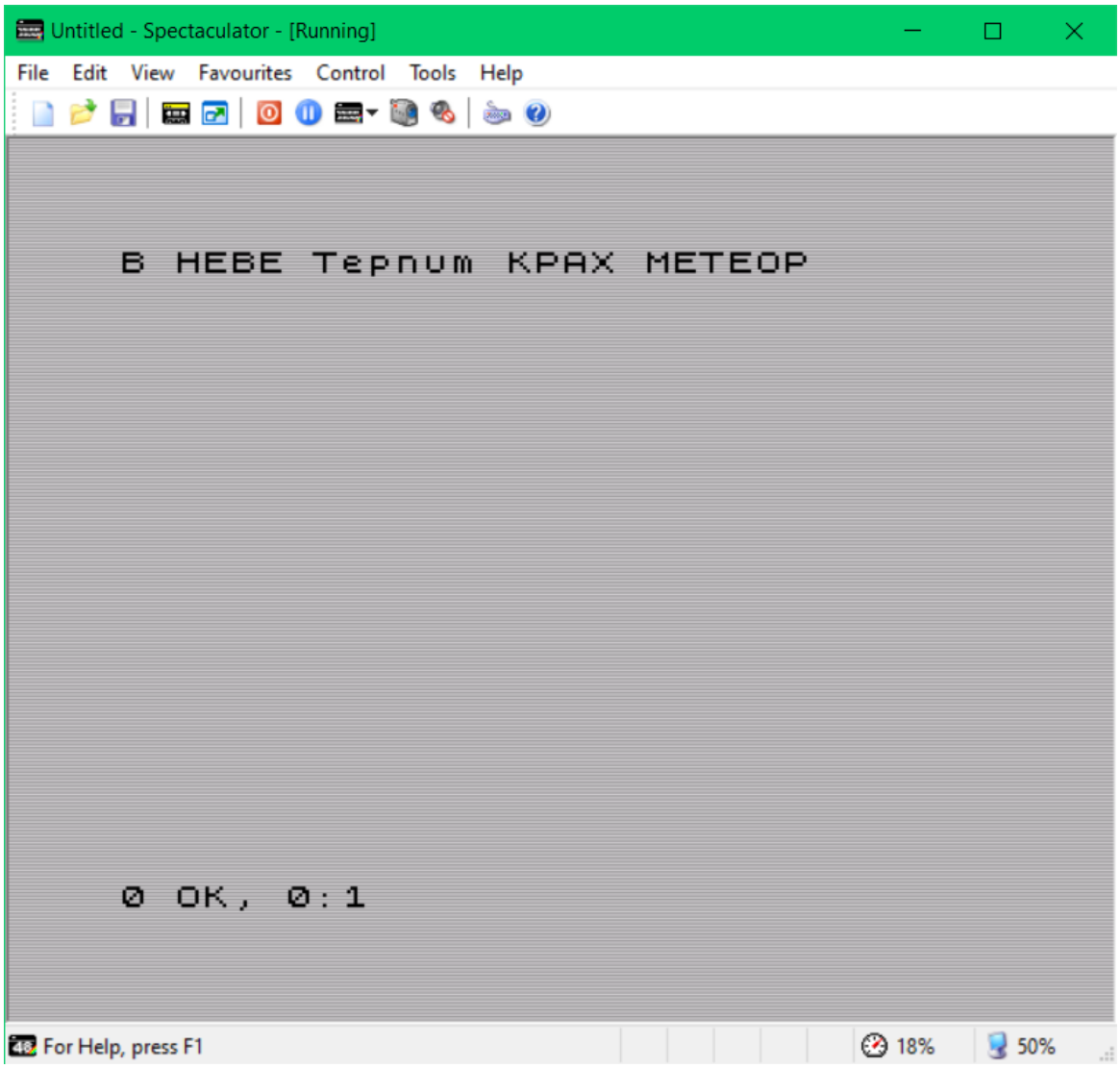
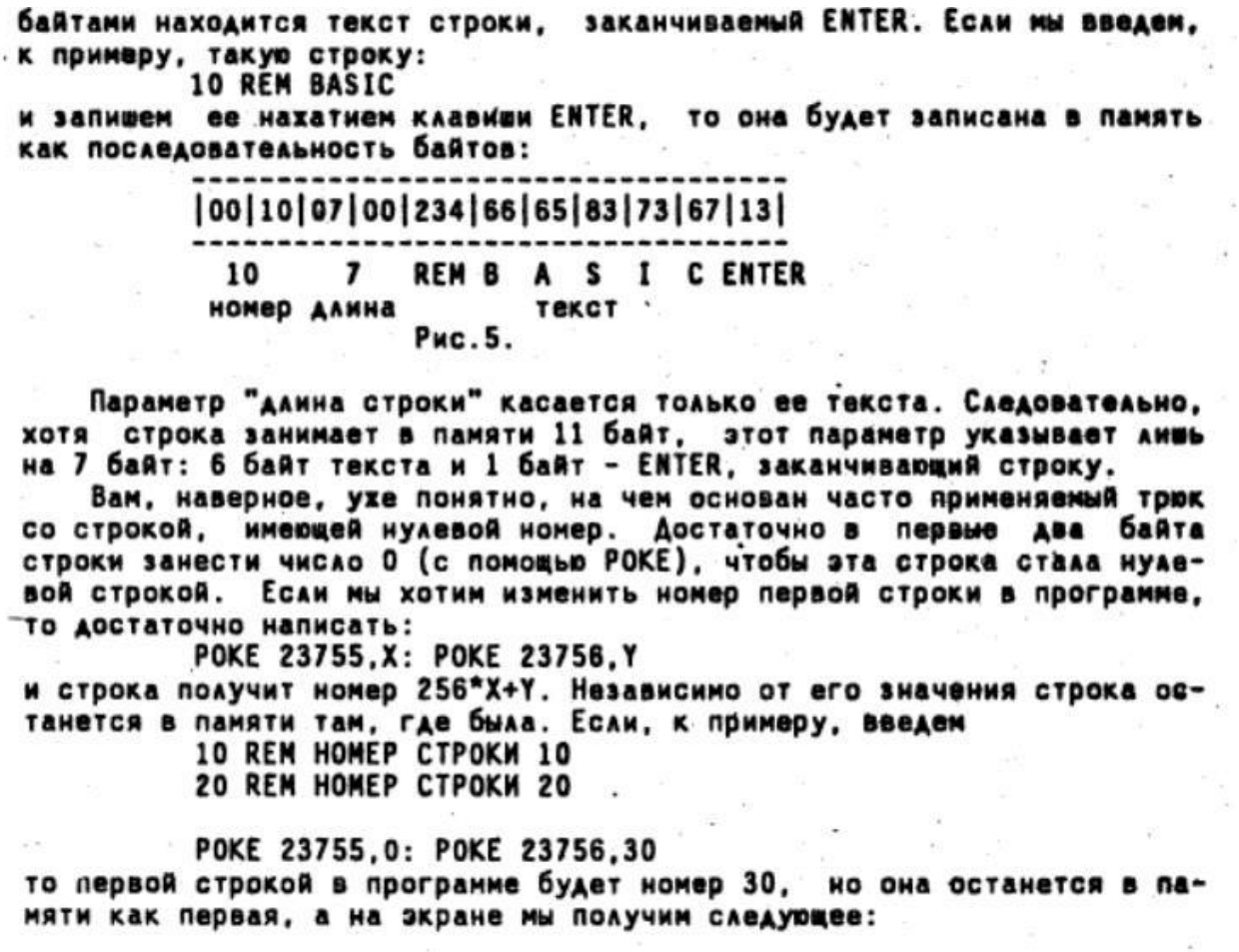


Рис. 156. Spectaculator. Проверка работоспособности синтезированной BASIC строки командой RUN.

Итак, прямой ввод строк программы в режиме «God Mode» даёт огромные преимущества. С первым вы только что познакомились. Из примера становится понятно, что BASIC строки могут иметь порядковый номер от 0 до 65535.

А теперь нужно бежать на помощь и спасать туристов с теплохода. Но как? Строка номер «0» известна издавна по BASIC загрузчикам из игр и обросла мифами и легендами. Это видимая выполняемая строка, которая не создаётся, не берётся на редактирование и не удаляется. Она просто существует. Забавно, правда? Нужно глянуть, что думали и знали о ней в древности наши предки.

В многочисленных однотипных брошюрах «*Тайники ZX-Spectrum*» (*растиражированный некачественный перевод с польского первоисточника*) даётся совет, как самую верхнюю BASIC строку превратить в 0 или задать любой другой номер:



байтами находится текст строки, заканчиваемый ENTER. Если мы введем, к примеру, такую строку:

```
10 REM BASIC
```

и запишем ее нажатием клавиши ENTER, то она будет записана в память как последовательность байтов:

```
-----------------------------------
|00|10|07|00|234|66|65|83|73|67|13|
-----------------------------------
  10     7   REM B  A  S  I  C ENTER
 номер длина        текст
```

Рис.5.

Параметр "длина строки" касается только ее текста. Следовательно, хотя строка занимает в памяти 11 байт, этот параметр указывает лишь на 7 байт: 6 байт текста и 1 байт - ENTER, заканчивающий строку.

Вам, наверное, уже понятно, на чем основан часто применяемый трюк со строкой, имеющей нулевой номер. Достаточно в первые два байта строки занести число 0 (с помощью POKE), чтобы эта строка стала нулевой строкой. Если мы хотим изменить номер первой строки в программе, то достаточно написать:

```
POKE 23755,X: POKE 23756,Y
```

и строка получит номер 256*X+Y. Независимо от его значения строка останется в памяти там, где была. Если, к примеру, введем

```
10 REM НОМЕР СТРОКИ 10
20 REM НОМЕР СТРОКИ 20

POKE 23755,0: POKE 23756,30
```

то первой строкой в программе будет номер 30, но она останется в памяти как первая, а на экране мы получим следующее:

Рис. 157. Типовой текст из перевода польской книжки о секретах строки с номером «0».

На момент выпуска этих книжек информацию можно считать прорывной и сенсационной. Но на 2024 год это явно не то, что хотелось бы знать о нулевой строке. Гораздо любопытнее выяснить, почему строка «0» имеет такие свойства, где они заложены и как модифицировать прошивку, из которой состоит BASIC.

Снова на помощь приходит Жёлтая ➪Стрелочка. Ведь она хорошо знает свои территории. Вот вы берёте на редактирование только что созданную нулевую строку… Стоп, но ведь везде пишут, что она не берётся? На заборе тоже написано. Возьмётся, куда она денется, достаточно поставить где надо малиновую ● точку-ловушку. Ведь нужно спасать туристов с тонущего экскурсионного катера, а то унесёт ненароком в Финский залив.

Войдите в отладчик по адресу 4027 и установите малиновую ● точку любым удобным способом, как объяснялось в главе 8 (кнопкой ✋, *CTRL+SPACE*, двойным нажатием левой кнопки мыши, выбором пункта меню «*Add/Remove Breakpoint*»):

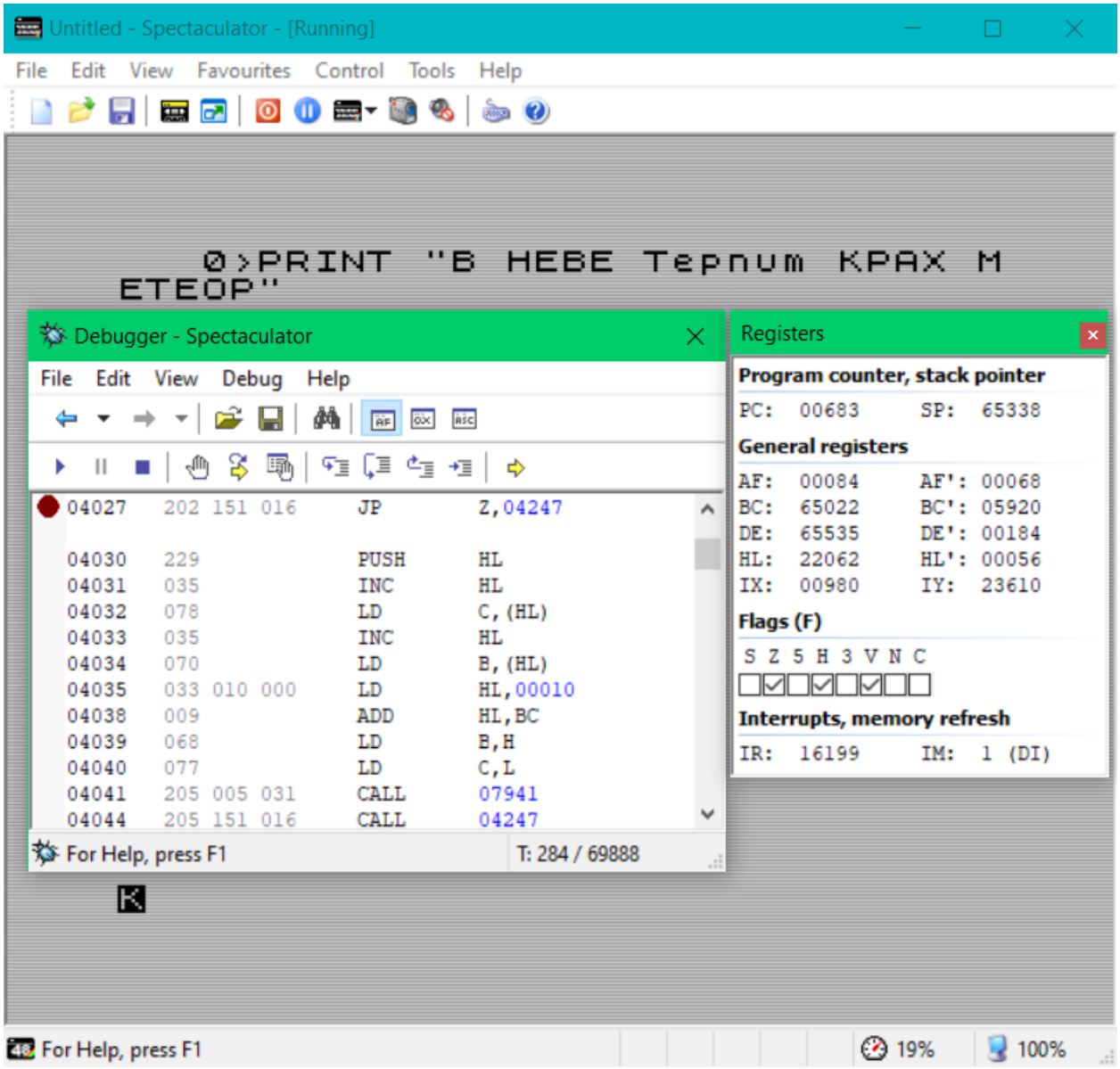


Рис. 158. Установка малиновой точки для обхода защиты редактирования нулевой строки.

Возвращайтесь в отладчик кнопкой «*Trace (F5)*». А теперь традиционным способом нажимайте на клавиатуре `CAPS SHIFT+1`, чтобы взять созданную нулевую строку:

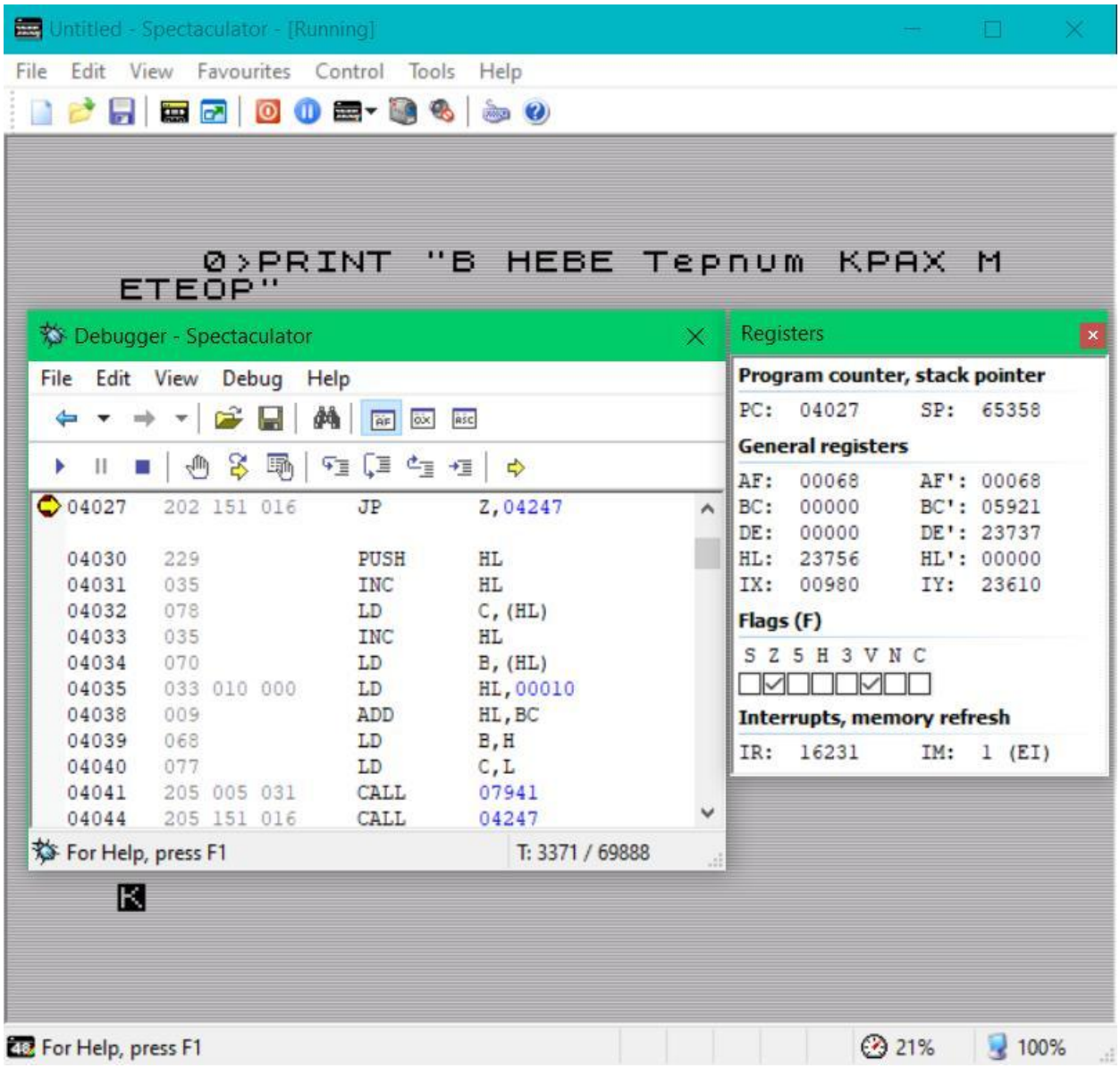


Рис. 159. Процесс остановки BASIC в процессе взятия на редактирования нулевой строки.

И вот Стрелочка попалась в выставленную ловушку. Чтобы строка «0» взялась, требуется всего лишь не дать Стрелочке перейти по условию «`JP Z, 04247`», но судя по выставленной Z ☑ галочке «Z», именно это она и собирается сделать следующим шагом.

Как это предотвратить? Просто снять Z ☐ галочку с этого квадратика под буквой Z в окошке «*Registers*». Пока просто смотрите:

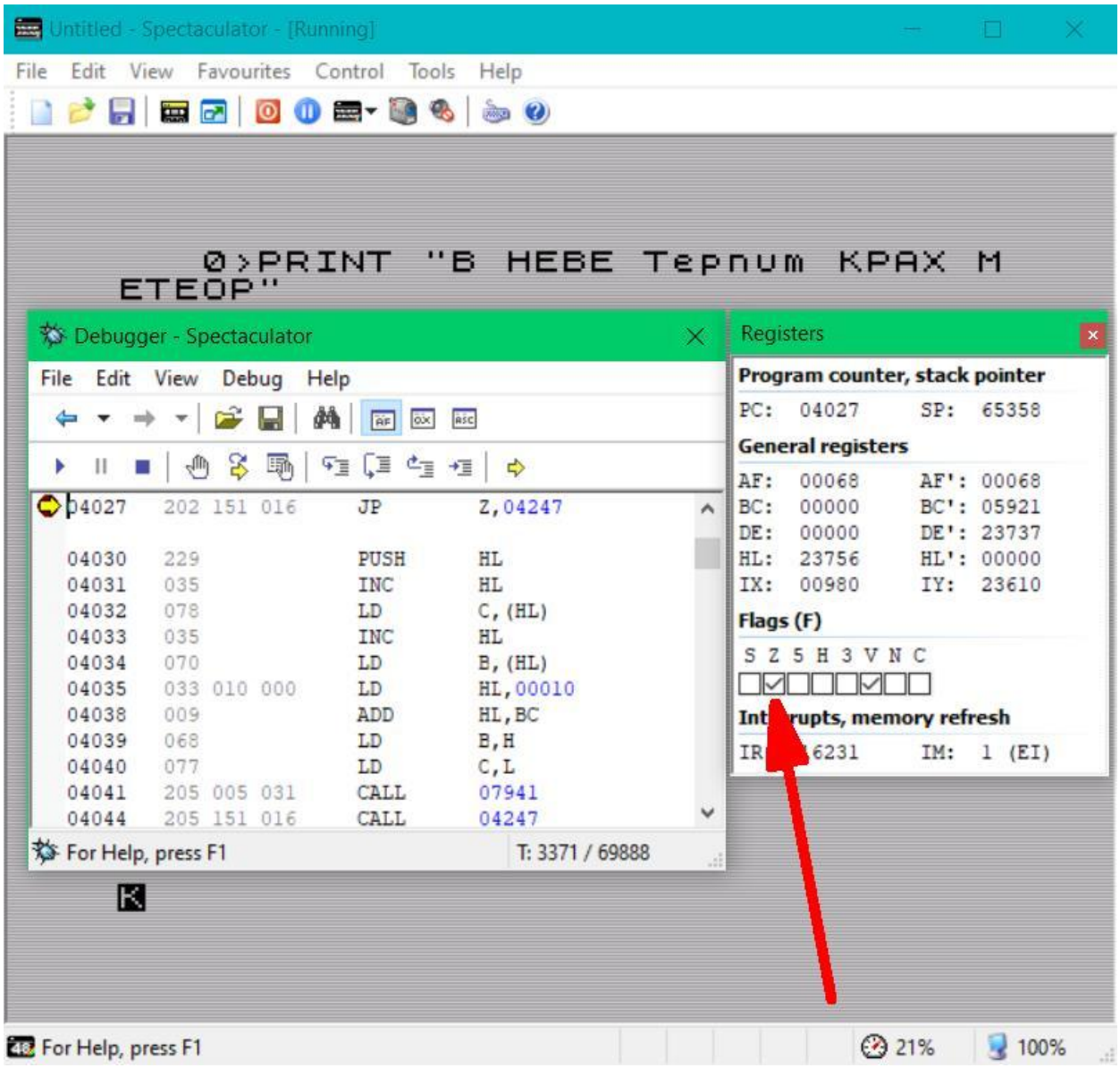


Рис. 160. Spectaculator. Галочка Z, которую нужно снять для корректировки выполнения программы.

А вот теперь наведите курсор мышки на окошко и нажмите левую кнопку. Галочка пропала и… значение «AF» покраснело. Вместо «68» оно стало 4, то есть снятая галочка «Z» из текущего значения «AF» вычитает 64. Так что аналогичного результата можно добиться простым вычитанием по принципу: AF=AF-64.

Теперь Стрелочке открыт прямой путь вниз, но прежде нужно снять капкан с приманкой. Удалите малиновую точку (повторным нажатием кнопки , *CTRL+SPACE*, двойным нажатием левой кнопки мыши, выбором пункта меню «*Add/Remove Breakpoint*»):

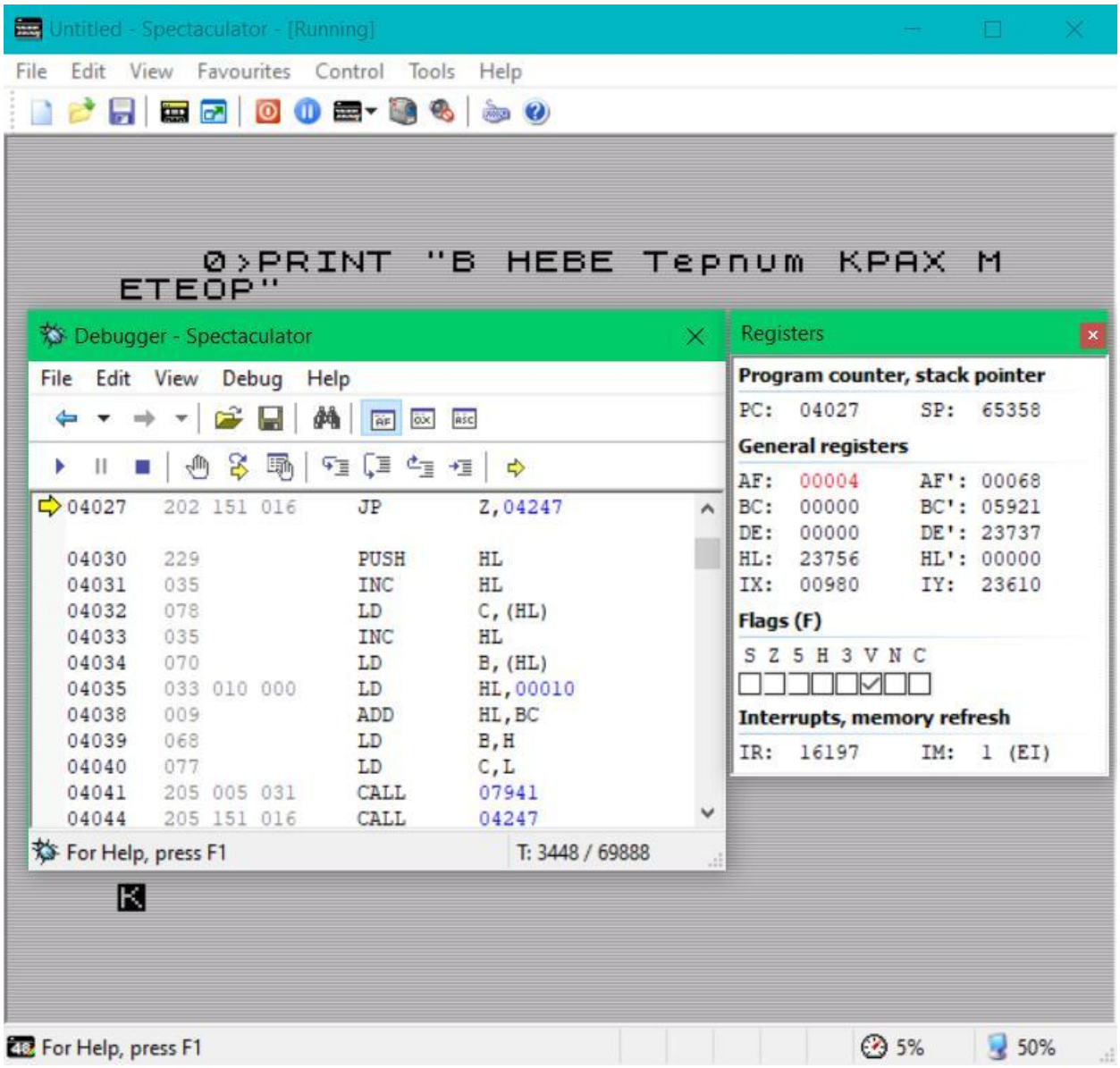


Рис. 161. Снятие галочки Z для корректировки работы программы взятия нулевой строки.

Нажимайте «*Trace (F5)*»:

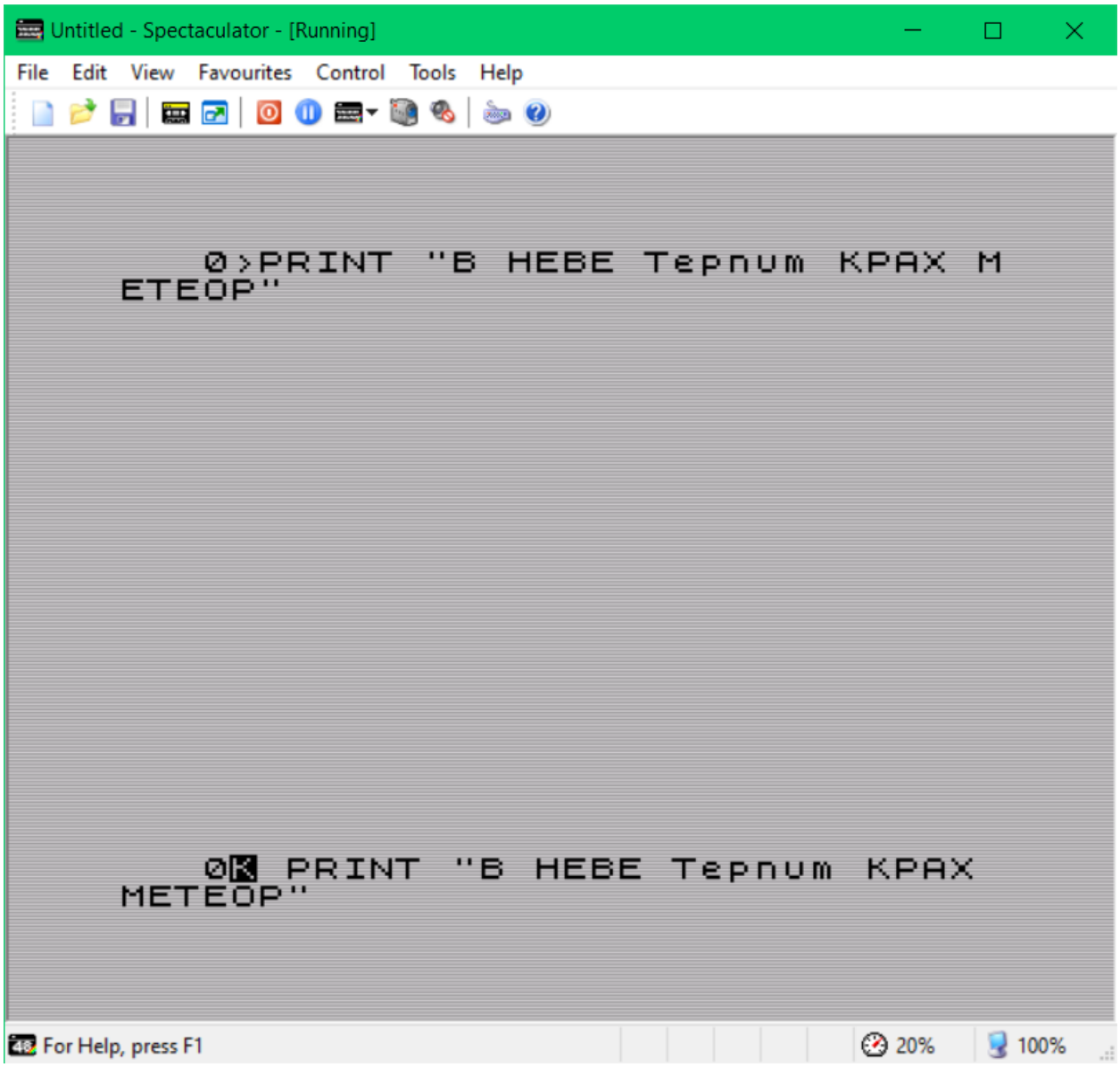


Рис. 162. Процесс взятия на редактирование нулевой строки после корректировки.

Вот так. И пусть теперь кто-нибудь возразит, что нулевую строку невозможно взять в буфер на редактирование. Из «*God Mode*» – как нефиг.

Приступаем к спасению судна. Нулевая строка разблокирована. Теперь отредактируйте стандартным способом фрагмент фразы «`Тернит КРАХ`» на «`ОТ ВЕТРА МОТАЕТ`».

Должно получиться:

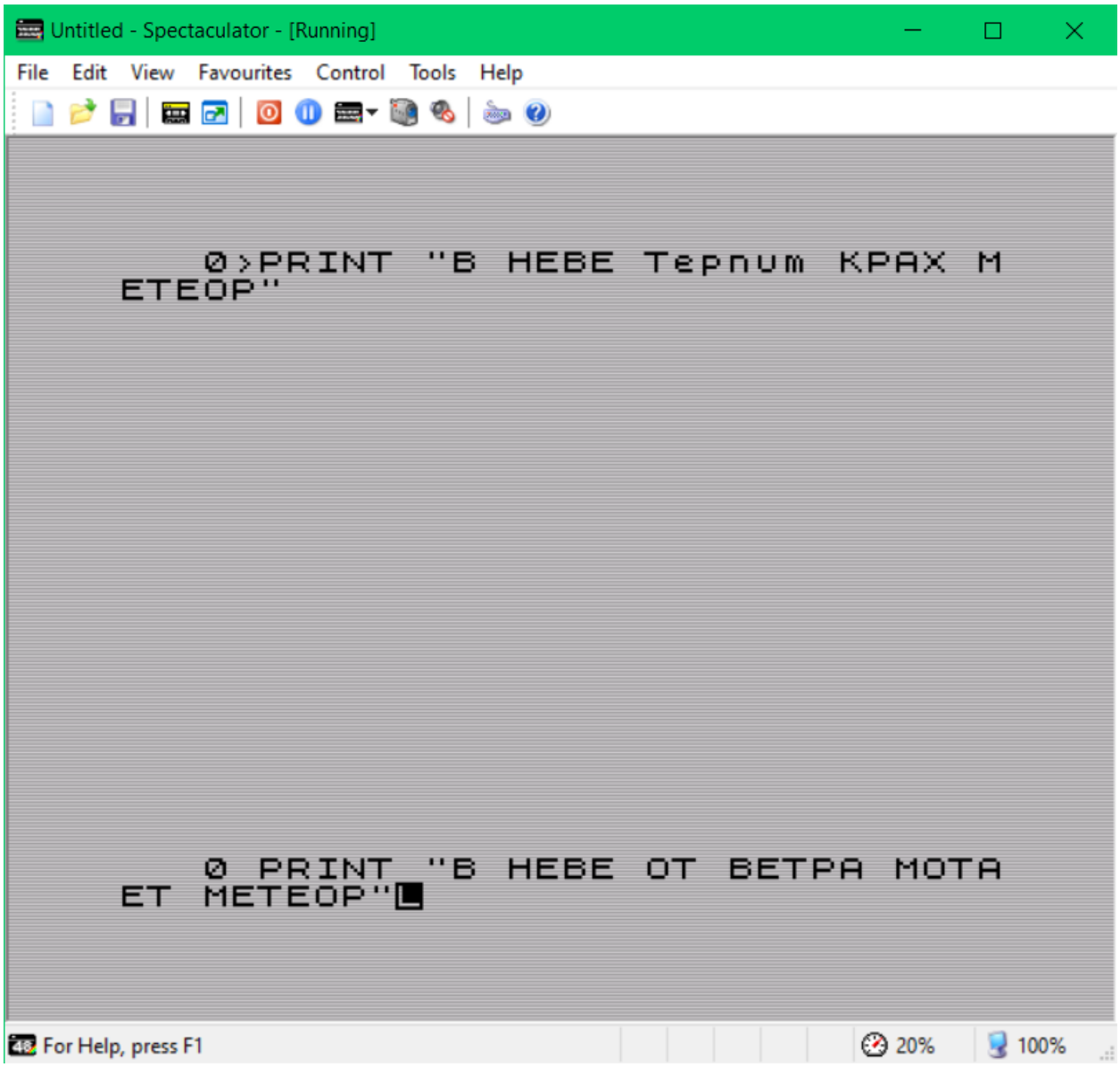


Рис. 163. Внесение изменений в нулевую строку натуральным способом.

Вводить прямым способом эту строку бесполезно. Из теории прошлой главы вы знаете, что при нажатии `ENTER` строка не обновится. Вместо этого выдастся сообщение «`C Nonsence in BASIC`» и на этом всё закончится.

Снова войдите в отладчик и теперь по адресу 4826 установите малиновую точку:

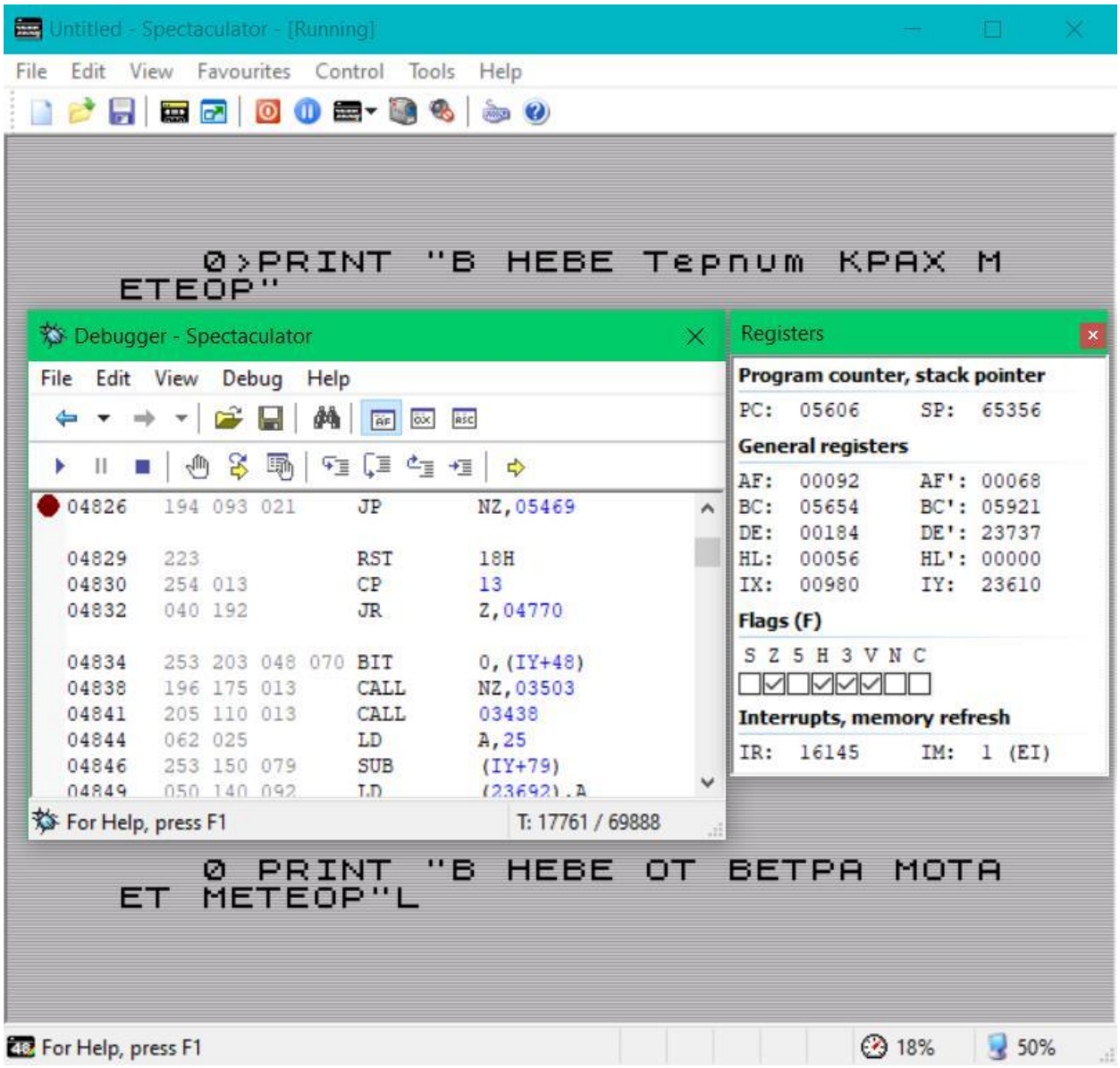


Рис. 164. Установка точки прерывания для обхода защиты ввода нулевой строки.

Нажимайте «*Trace (F5)*». По выходу в *Spectaculator*, натуральным способом нажмите ENTER для ввода строки:

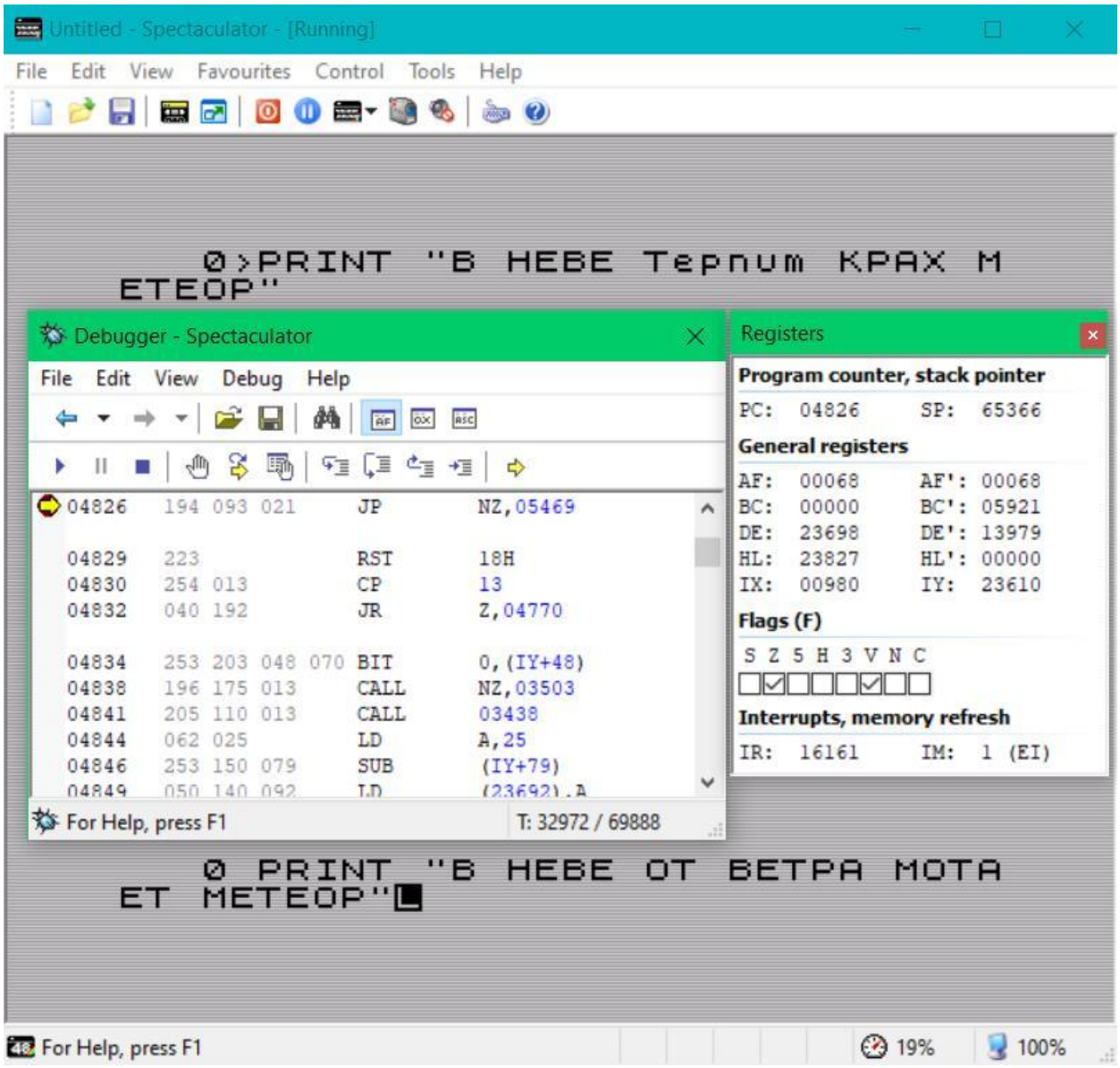


Рис. 165. Spectaculator. Остановка на точке прерывания во время ввода нулевой строки.

Экран мигнул, и Стрелочка появилась на нужном адресе. Аналогичным образом уберите точку, снимите галочку из окошечка «Z» а окне «*Registers*» или вычтите «64» из «AF» и внесите туда 68-64=4:

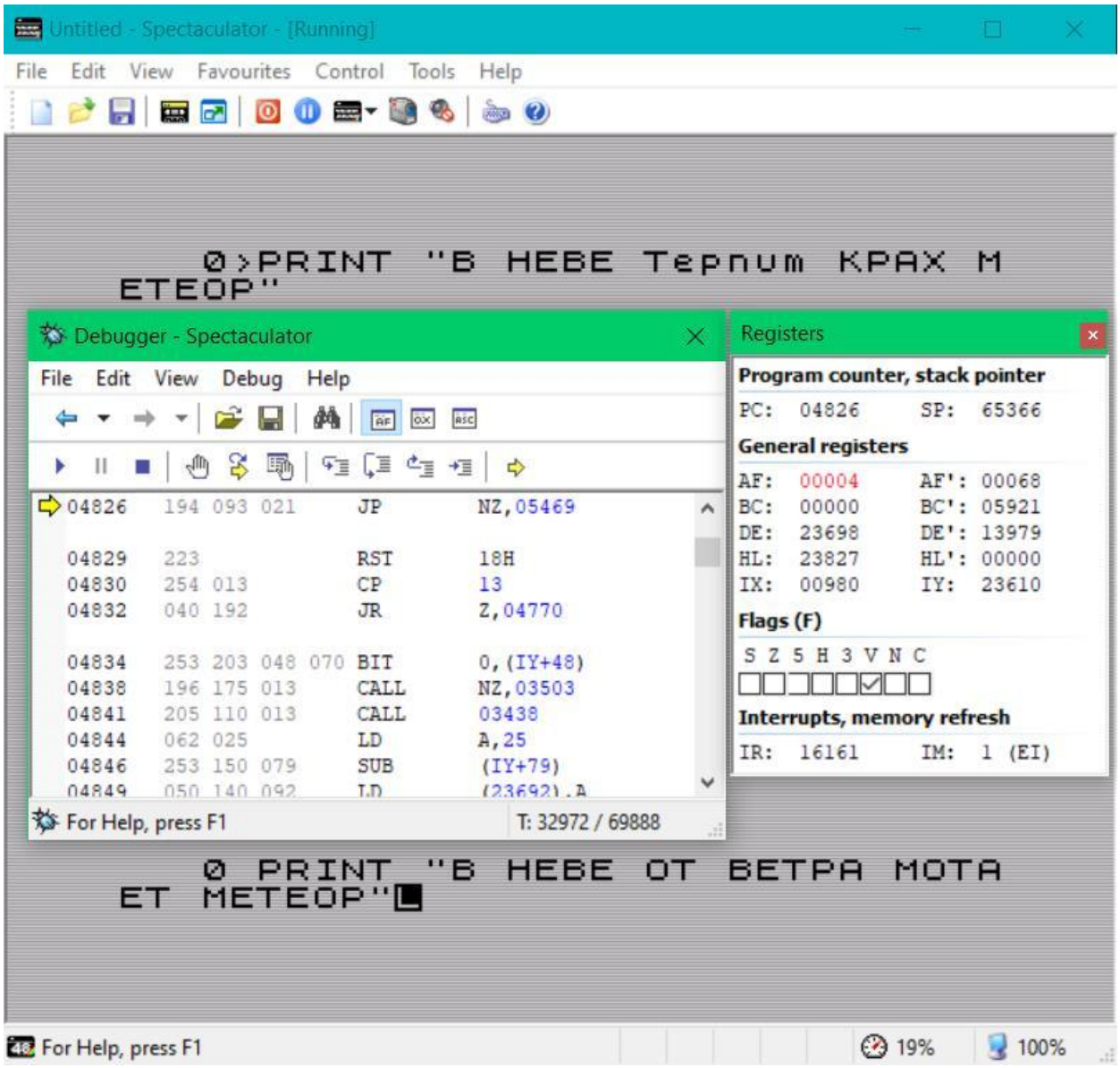


Рис. 166. Снятие галочки Z для корректировки свойств программы при вводе нулевой строки.

Нажимайте «*Trace (F5)*»:

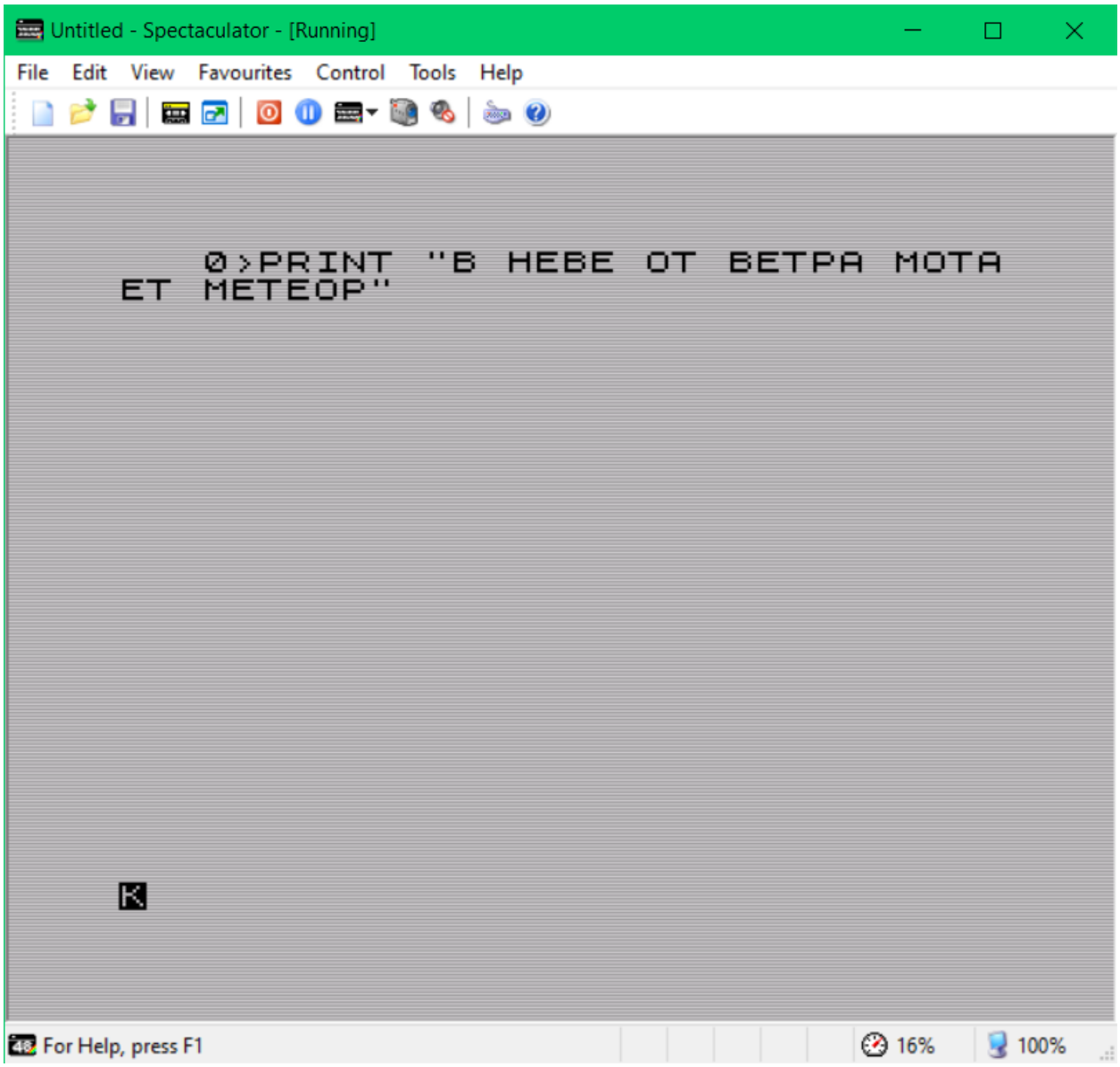


Рис. 167. Ввод отредактированной нулевой строки.

И вот отредактированная нулевая строка, как и любая другая, заместила и осталась висеть. Теплоход больше не тонет, а лишь сильно качает на волнах. Туристы чуть подмочили ширинки от страха, но больше не бултыхаются в холодной воде, а это самое главное.

Предлагаю подробнее рассмотреть процессы редактирования и ввода. Начну с редактирования:

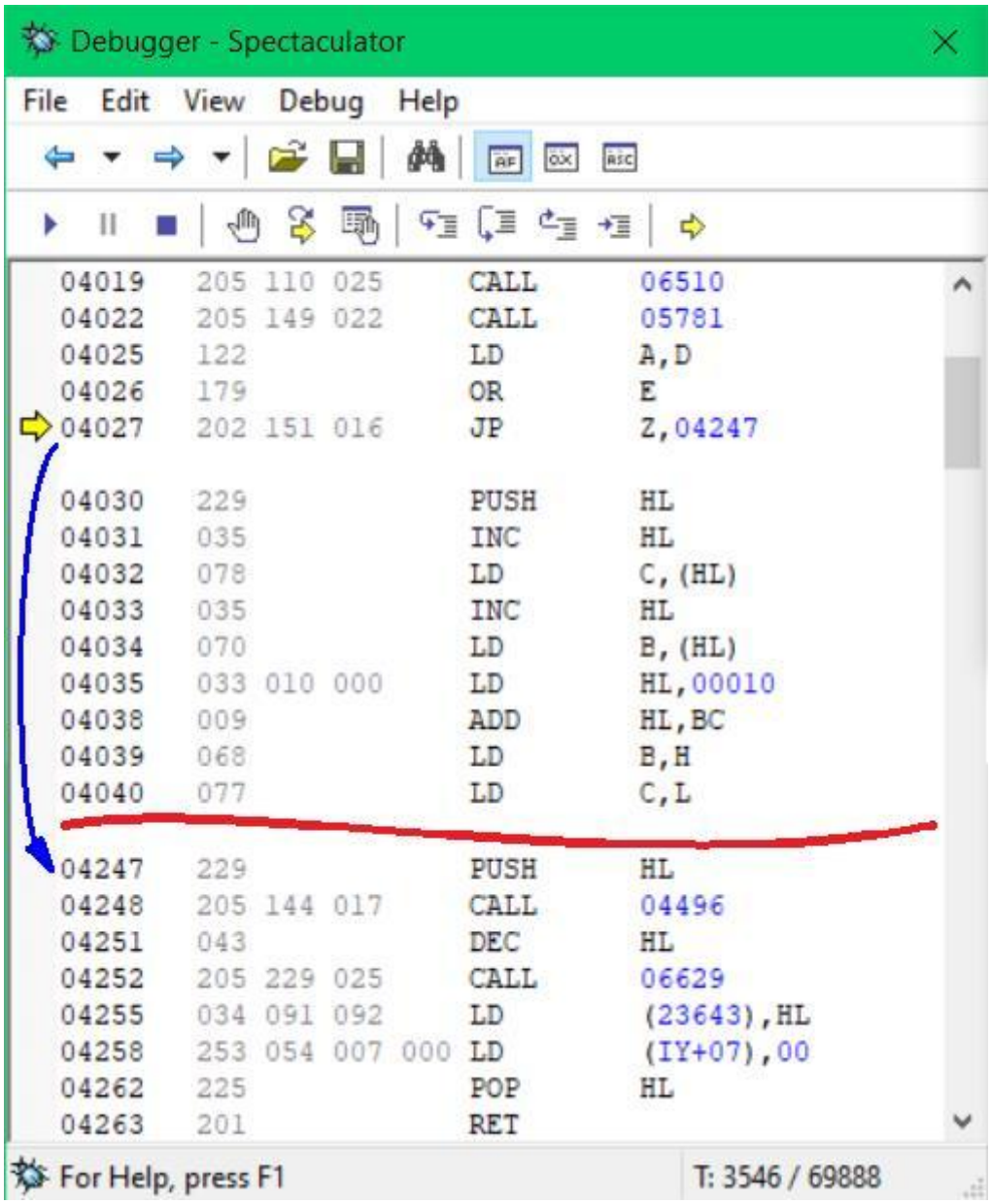


Рис. 168. Процесс взятия строки с номером «0».

А происходит там следующее. Подпрограммой `LINE ADDR (6510)` определяется адрес строки для взятия на редактирование. По выходу из неё, в «HL» и «DE» скидывается адрес начала строки. Следующая программа `LINE NO (5781)` по заданному адресу определяет номер строки и выдаёт номер в «DE». Следом в 4025 и 4026 стоит проверка с отсечкой (`D OR E`). Если выбранная строка имеет номер «0», иначе говоря, не существует (по мнению создателей, естественно), то редактирование игнорируется и просто освежается экран подпрограммой `CLEAR SP (4247)`. Ликвидировав этот переход по условию нуля (`JP Z, 4247`), строка с номером «0» появляется в буфере редактора. В том вы убедились на примере выше.

Сделано это не случайно, и не для разработчиков игр, которые в качестве мнимой «защиты» плодили нулевые строки. Всё гораздо проще. Это элементарная защита от сбоя, чтобы при холостом нажатии `CAPS SHIFT+1` в редактор не взялся кусок памяти с мусором, а просто обновился экран.

Не удаляется строка «0», а при попытке ввода выдаётся сообщение «`C Nonsence in BASIC`» по этой же причине. Выше этот механизм рассматривался.

Взгляните снова:

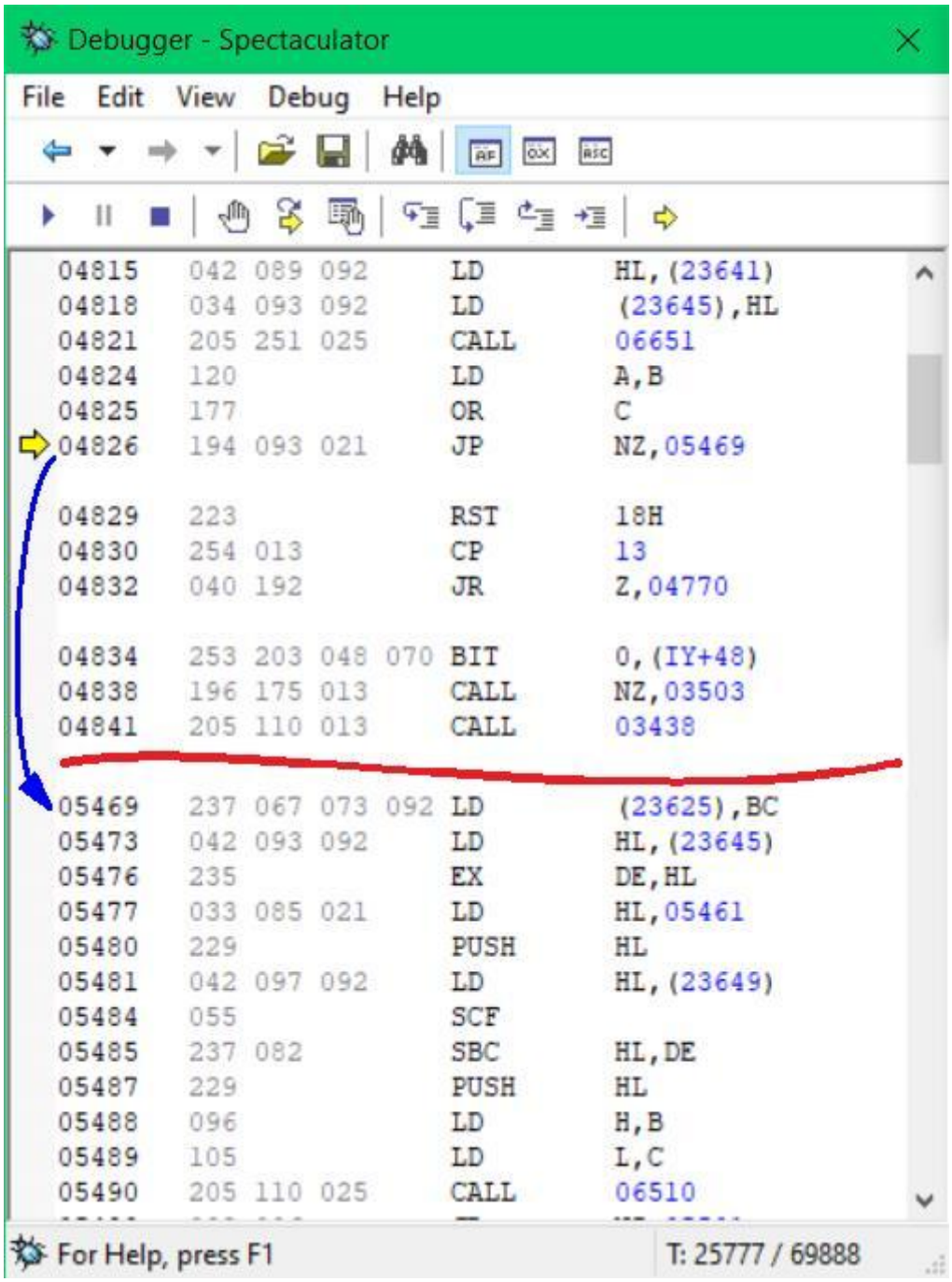


Рис. 169.

Подпрограммой `E-LINE NO (6651)` вычисляется и создаётся номер текущей строки, после чего возвращается в «BC». В ячейках 4824-4825 происходит проверка на 0 (`B OR C`). Если в «BC» имеется какой-то номер, значит, строку нужно просто положить в память, если ничего не отложилось, и остался 0, то строка без номера. Её следует выполнить и забыть.

Но ведь «0» это не только пустота, а ещё и символ 0. Приняв строку с числом перед оператором, компьютер начинает её выполнять. Считывая первый символ, он понимает, что это не команда, а число. Происходит разрыв шаблона и выдаётся сообщение «`C Nonсence in BASIC`» с выходом на поверхность в главный цикл.

При попытке ввести «0» для стирания существующей нулевой строки тоже ничего не произойдёт. Пройдя отсев по значению 0, в адресе 4829 начинается прощупывание программы на предмет операторов и тут выясняется, что за этим нулём (или 2-4 нулями) голый `ENTER`. Происходит возврат в 4770 с обновлением экрана.

В прошлом примере, сделав принудительный переход по условию «JP NZ 5469», защита проигнорировалась, и нулевая строка ввелась. Аналогичным образом можно стереть строку, если ввести чистый 0 с ENTER.

Подводя итоги, выведу алгоритм взятия строки и ввода нулевой строки. С этой главы появилась новая команда «снять галочку». Пока предлагаю её обозначить как «*Z OFF*» «*Z ON*» или «*AF ← AF-64*». Посмотрю, какой вариант приживётся в будущих главах.

Программа редактирования нулевой строки будет выглядеть так:

```
Debugger
Dec
Go To 4027
Add Breakpoint 4027
Trace
BASIC ← EDIT
Remove Breakpoint 4027
AF ← AF-64
Trace
```

Программа ввода/удаления нулевой строки:

```
Debugger
Dec
Go To 4826
Add Breakpoint 4826
Trace
BASIC ← ENTER
Remove Breakpoint 4826
AF ← AF-64
Trace
```

На этом всё. Нажмите Reset и переходите к следующей главе.

Глава 15
Небанальные свойства банальных строк или с Первым Сентября

Краткое содержание: синтез BASIC строк, нумерация, курсор, 1 сентября

Категория строк в интервале с 1 по 9999 называются «Банальными». Это классические видимые выполняемые строки с полным набором функций редактирования, ввода и удаления. И вроде бы детально их рассматривать не имеет смысла, но в режиме «*God Mode*» даже с этим типом строк можно творить некоторые интересные вещи.

Прежде чем приступить попробую обосновать сюжеты нижеследующих примеров. Я начал писать эту книгу 30 мая 2004 года в 17 часов 42 минуты и 38 секунд. Страшно подумать, но сегодня уже 30 августа. Три месяца пролетело, я работаю всё свободное время, а по задумке создано не более половины книги, а возможно и меньше. А ведь послезавтра первый день осени…. Первое сентября!

Предлагаю ввести программу, демонстрирующую «небанальные» свойства строк с самыми банальными номерами и на осеннюю тематику:

```
Debugger
Dec
Go To 23560
23560 ← 13
23611 ← 32
23627 ← 23878
23641 ← 23879
```

```
23649 ← 23881 23881 23881
23755 ← 3 232 34 0 245
23760 ←""АРТЕМ Рисует МАРКЕРОМ На парте
23791 ← 34 13 0 10 36 0 245
23798 ←""СЕМЕН Расхаживает перед Учениками
23832 ← 34 13 0 100 40 0 245
23839 ←""ВОВА На Трубе кидает жвачку В Мастера
23876 ← 34 13 128 13 128
Trace
```

Вводите:

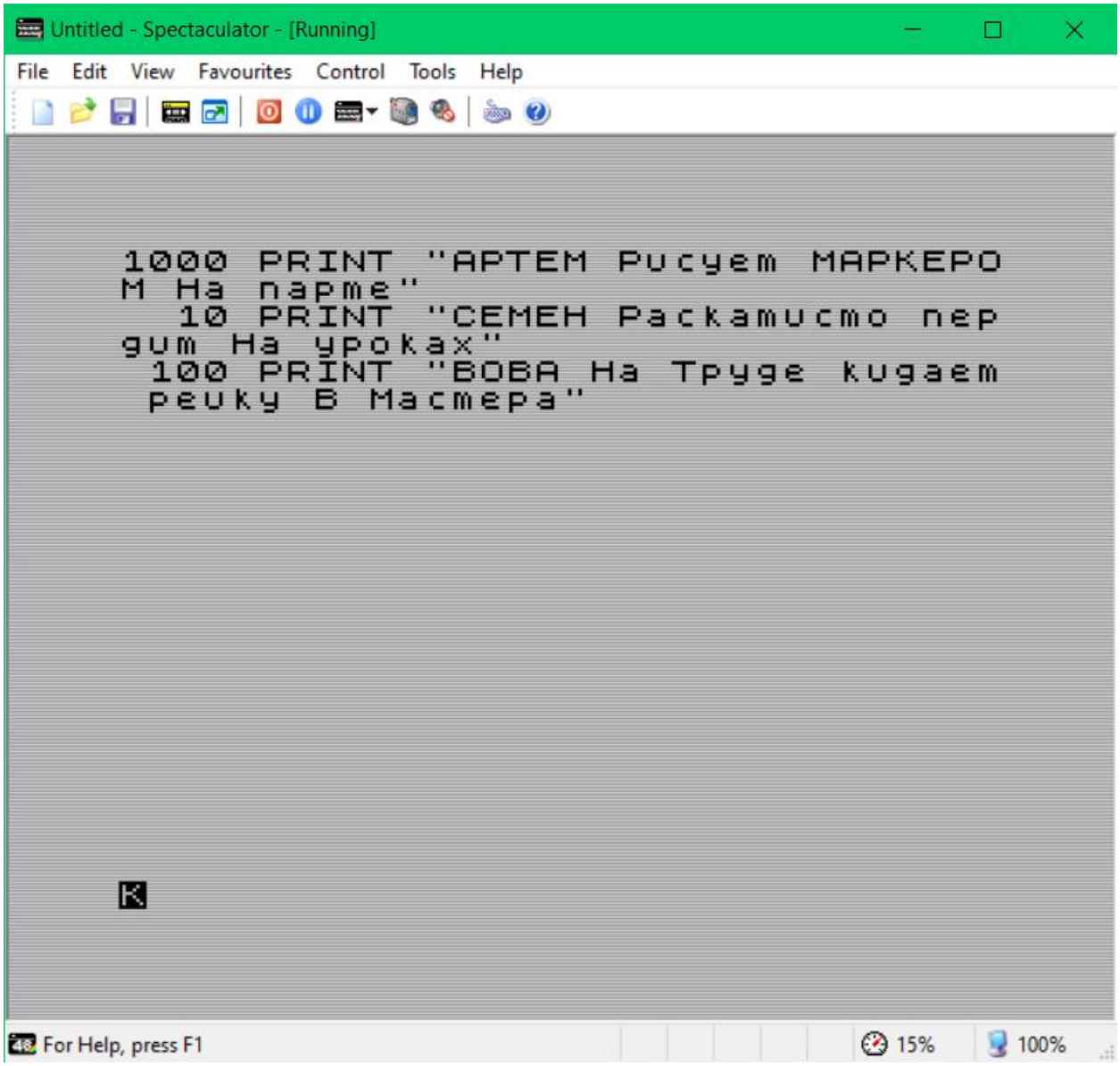


Рис. 170. Расположение BASIC строк в хаотичном порядке.

Первое сентября – день знаний, праздник детей, родителей и учителей. Начало нового учебного года. В школы и ПТУ с радостью потянулись ученики. Со школьных крылечек вновь зазвучали матюжки и конское ржание старшеклассников. В коридорах на переменах со свистом залетали отодранные плитки линолеума. В туалетах запахло папиросами с травой, на потолках замерли жжёные спички, в писсуарах появились первые кучи, а из разбитых унитазов вновь потекли полноводные реки. В эту золотую пору жизнь в учебных заведениях забила ключом…

Обратите внимание, строки встали в том порядке, в котором вы вводили, а не по номерам. 1000, 10, 100… лесенка дураков, блин. А хотя, чего это я? Какие примеры, такое и расположение. Хочется верить, , что рейка просвистела мимо, и двоечник Вовочка не зашиб бедного трудовика.

А кстати, посмотрите внимательно на номера строк. Не хватает привычного внешнего курсора `>`. Неплохо бы поставить. За его отображение отвечает указатель `E_PPC (23625)` в который достаточно вписать желаемый номер строки.

Войдите в отладчик и выполните следующие команды:

```
23625 ← 10
23611 ← 32
Trace
```

Смотрите, что получилось:

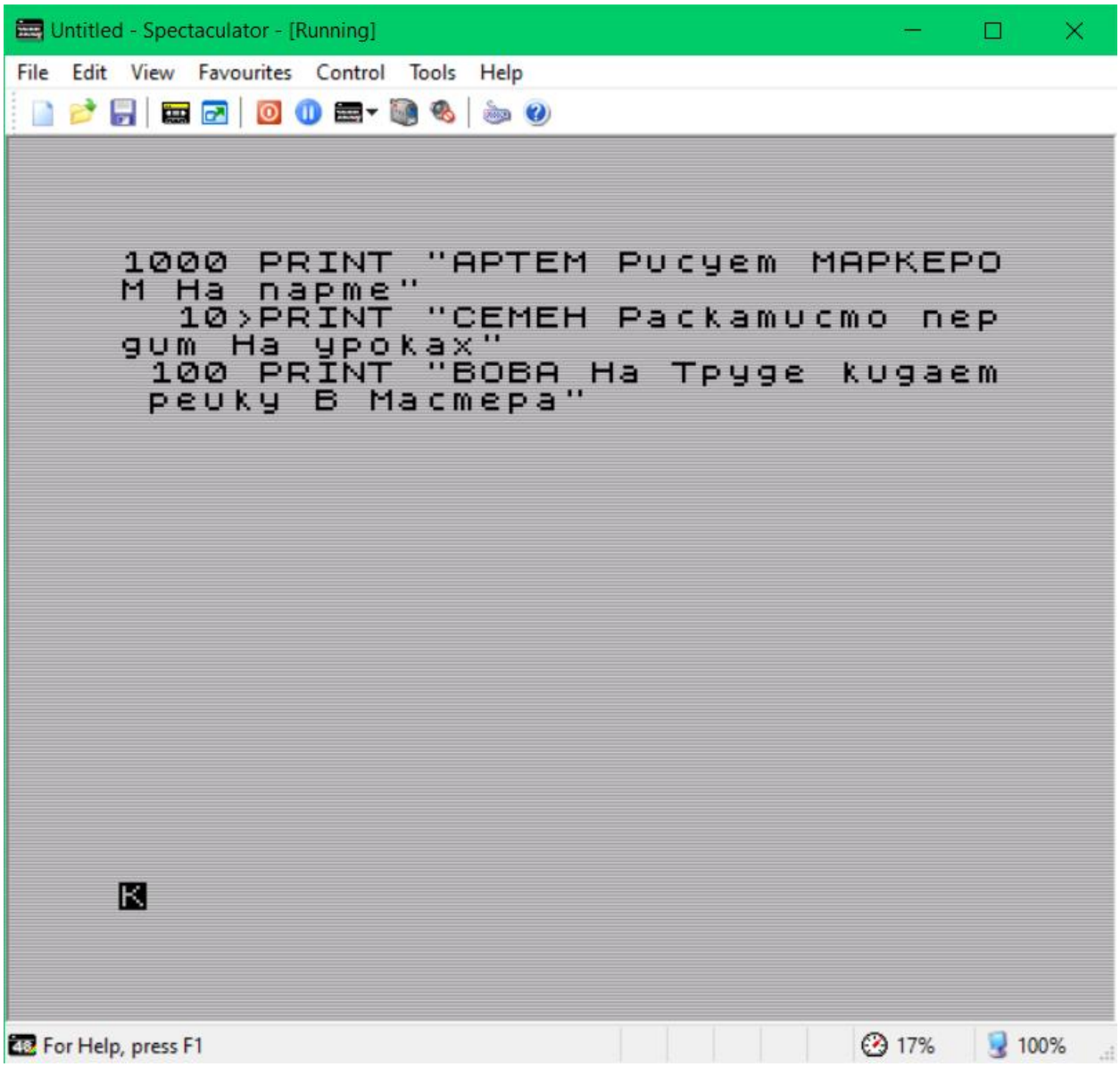


Рис. 171. Установка курсора искусственным способом на нужную строку.

Вот и курсорчик подвезли. Всё как задумывалось, на 10-ю строку. То, что она стоит по центру между 1000-й и 100-й, ну это так, издержки производства. Какой можно сделать вывод после этой главы? А вот такой:

Строкам можно задать одинаковые номера, располагать в памяти по убыванию, возрастанию или вразнобой».

Нажмите и можно двигаться дальше.

Глава 16
Символьно-числовое безумие или мифы старых книжек

Краткое содержание: «символьно-числовые строки», свойства строк 10000-16383

А почему, собственно, категория банальных строк по 9999? Да потому, что традиционным методом строка с номером 10000 и выше тупо не введётся. По нажатию ENTER в конце числа замигает знак вопроса. Предлагаю посмотреть, что именно мешает с клавиатуры ввести числа более 10000. Как выяснилось, даже ходить далеко не надо:

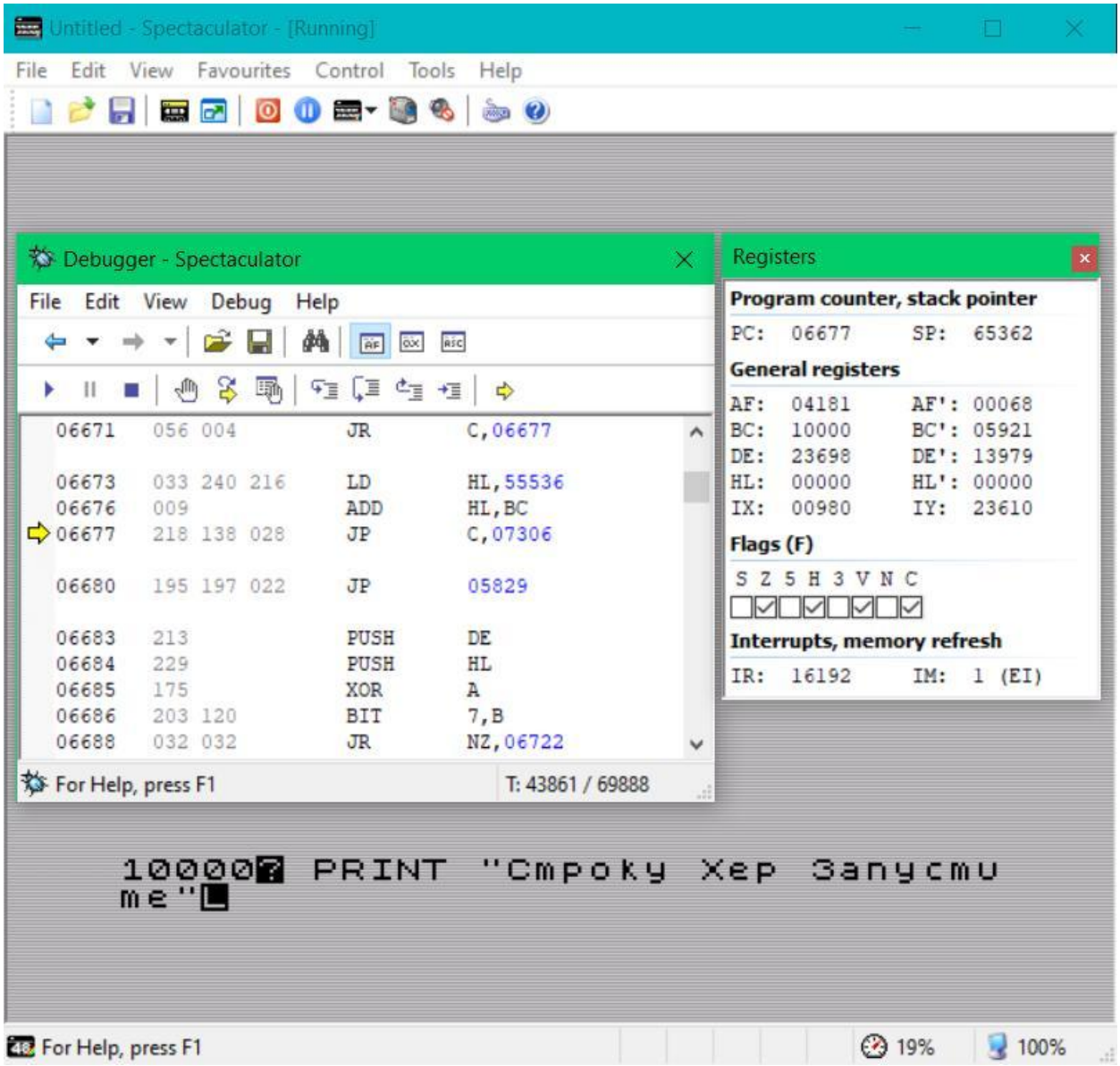


Рис. 172. Программа E-LINE NO. Процесс проверки строки на максимально допустимый номер.

В адресе 6673 задаётся значение 55536, что ровно на 9999 меньше предела исчисления. Суммируя с номером строки, получается некое значение. Если строка имела номер 10000 и выше, при сложении происходит переливание значения через предел 65535.

На основании этого включается ☑ галочка «C», выполнение прерывается и происходит возврат в редактор с мигающим знаком вопроса для удаления лишней цифры.

Фигня-война, но для начала хочется взглянуть, что думают по этому поводу в разной странной литературе:

> Известным способом защиты является также занесение значения больше 9999 в ячейки памяти, обозначающие номер строки БЕЙСИКа (первый байт - старший). Если он размещается в границах 10000-16383, а на листинге это выглядит, например, 0000 (вместо 10000) и строки невозможно вызвать на редактирование <EDIT>, если же превышает 16384 - дальнейшая часть программы считается не существующей.

Рис. 173. Вырезка из книги «Тайники ZX-Spectrum».

О как! Ну раз так, то обязательно нужно попробовать создать строку 10000. И вот так незаметно я подобрался к следующему виду строк, которые имеют нумерацию с 10000 по 16383. В народе называют… впрочем, название скажу чуть позже.

Откройте *Spectaculator* и натуральным способом наберите следующую строку, но не вводите:

```
10000 PRINT "В кустах у дороги стоит КАМАЗ"
```

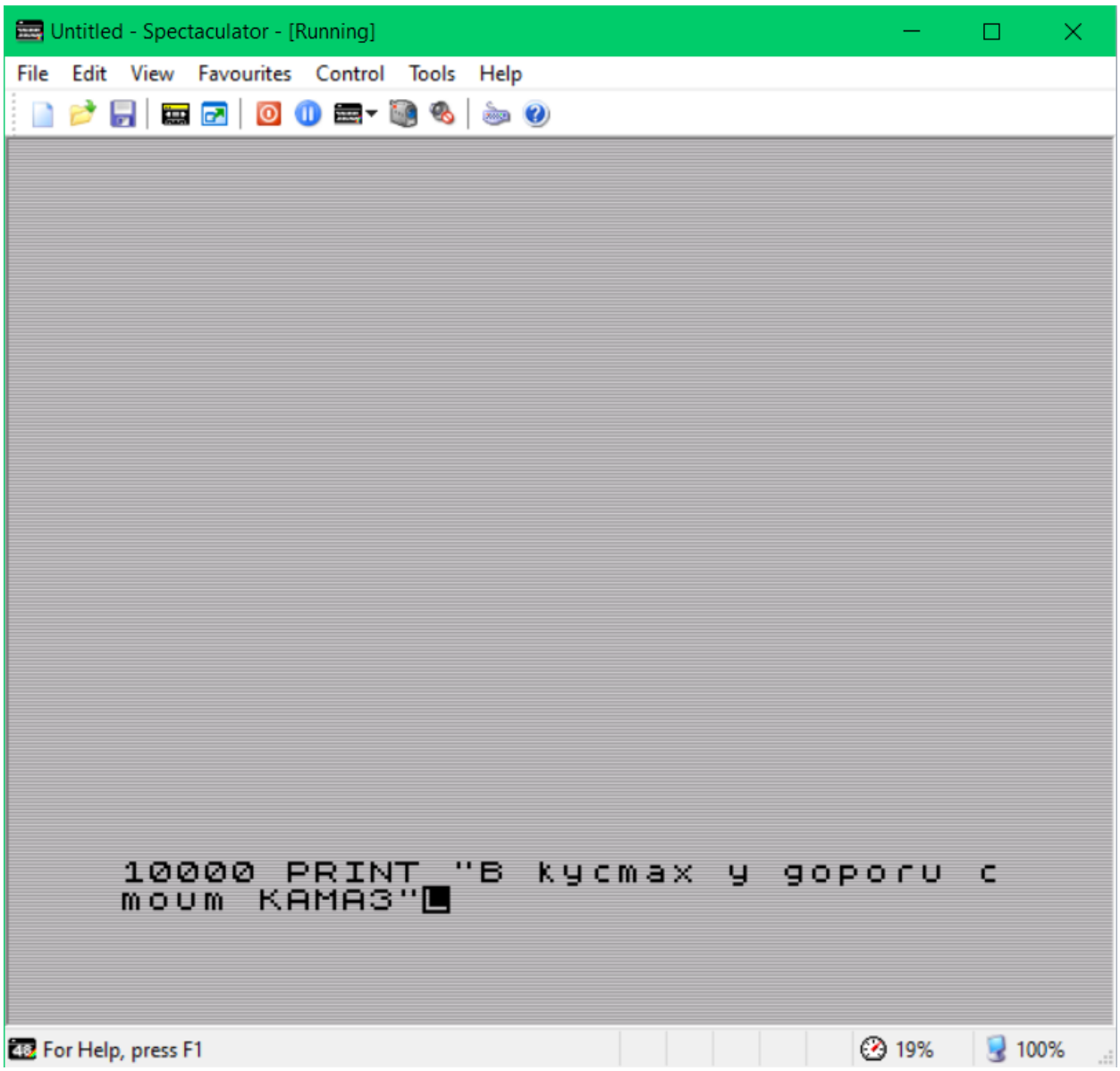


Рис. 174. Набор BASIC строки натуральным способом для очередного эксперимента.

А вот и герой данной фразы. Скромно приютился в кустах на дороге возле заброшенной котельной посёлка Елизаветино Волосовского района Ленинградской области:

Рис. 175. КАМАЗ в ПГТ Елизаветино у котельной. Фото 15 сентября 1997 года.

Обойти ограничение можно по аналогии с редактированием нулевой строки. Для этого достаточно проигнорировать команду `JP C, 7306` по адресу 6677, для чего снять с «C» галочку C ☐ или вычесть из значения «AF» единицу (*так как C это бит №0*).

Откройте «*Debugger*» в 6677 и поставьте малиновую точку на этот адрес:

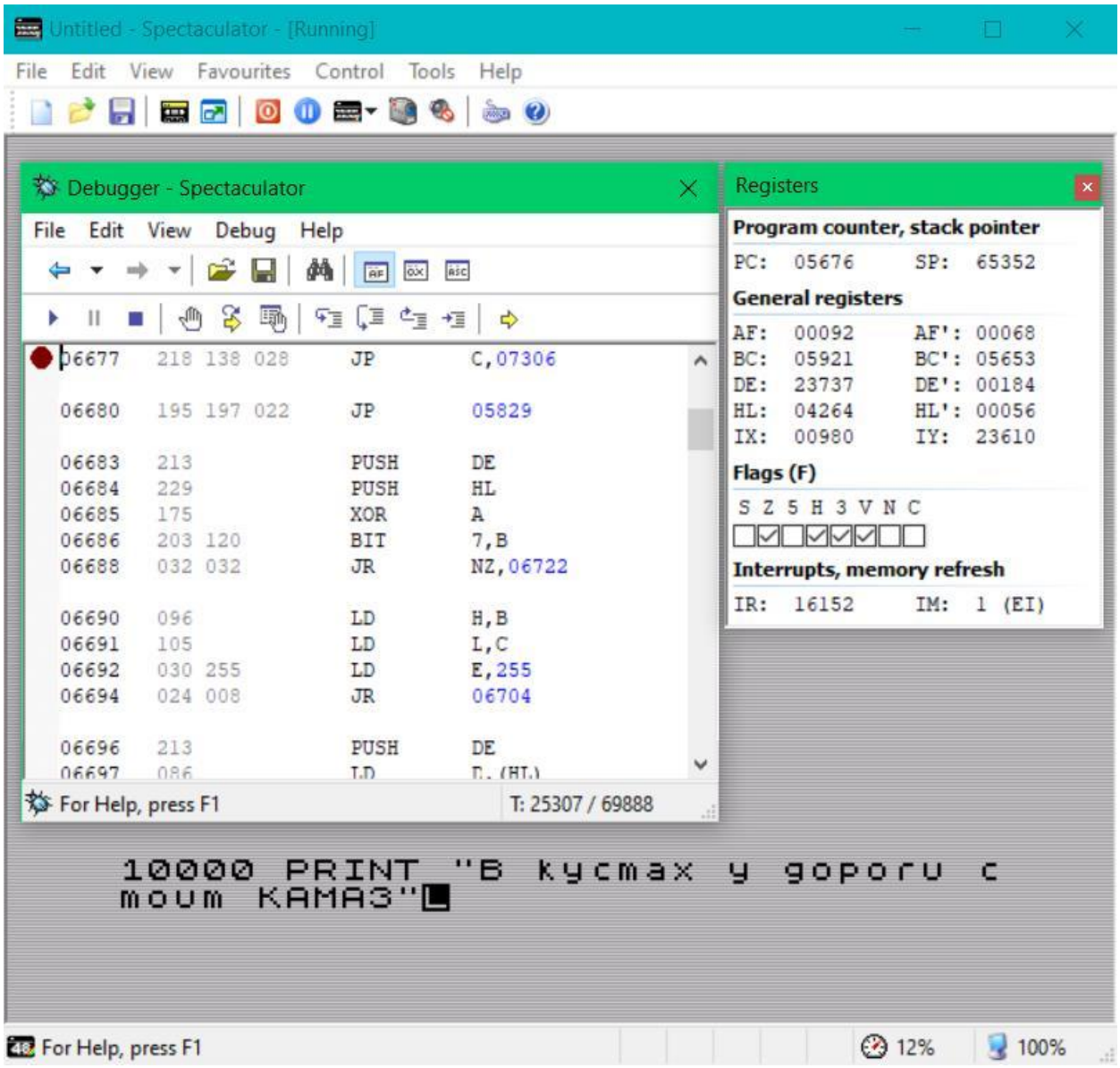


Рис. 176. Установка малиновой точки для подготовки ввода строки 10000

Нажмите «*Trace (F5)*». Следом введите строку натуральным ENTER'ом. Отладчик открылся на нужном адресе. По аналогии с нулевой строкой уберите малиновую ● точку. Снимите с «C» галочку левой кнопкой мыши. Значение «AF» с 4181 поменяется на 4180:

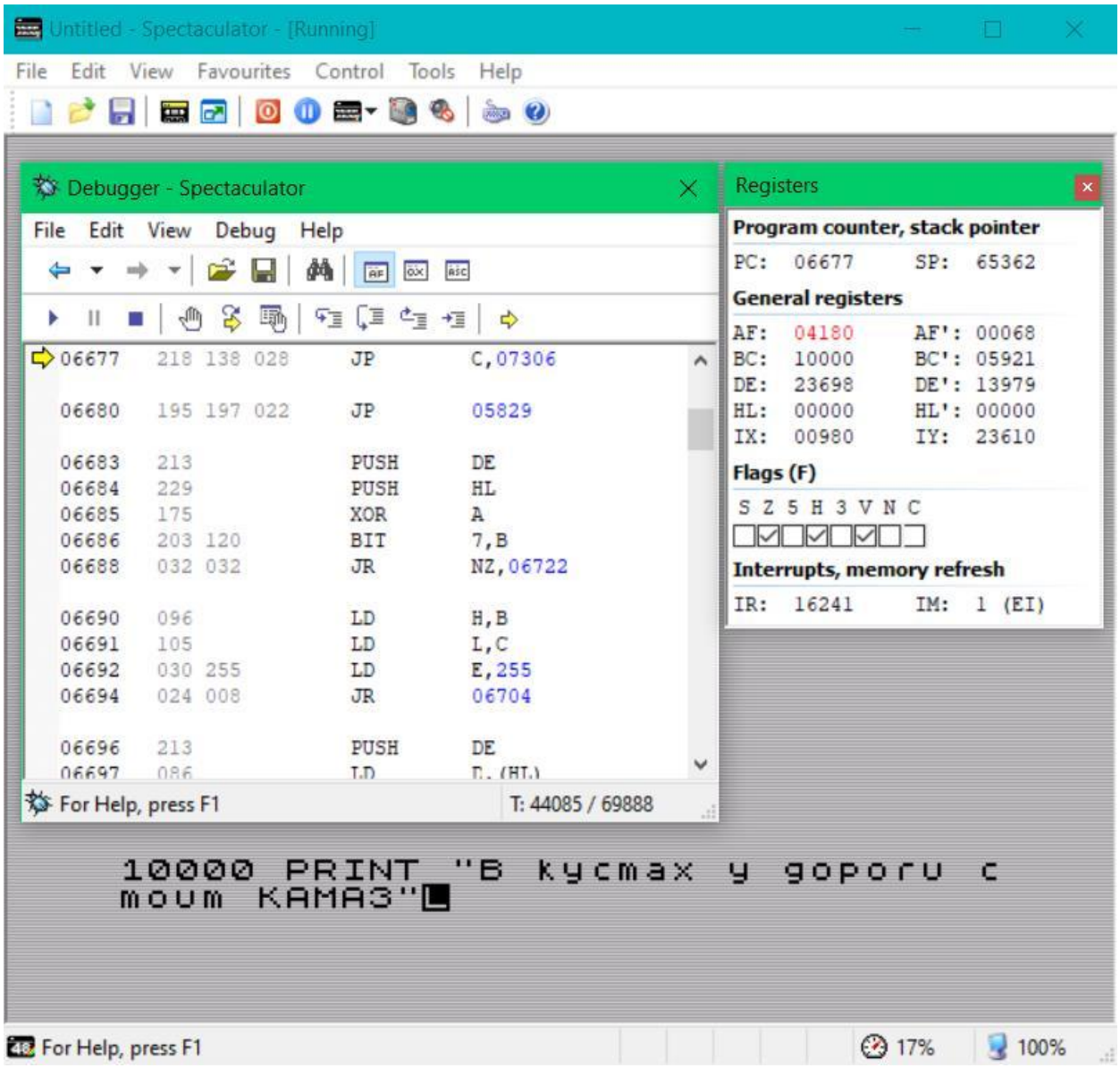


Рис. 177. Корректировка свойств программы ввода строки 10000. Снятие галочки «C».

Нажимайте «*Trace (F5)*» и:

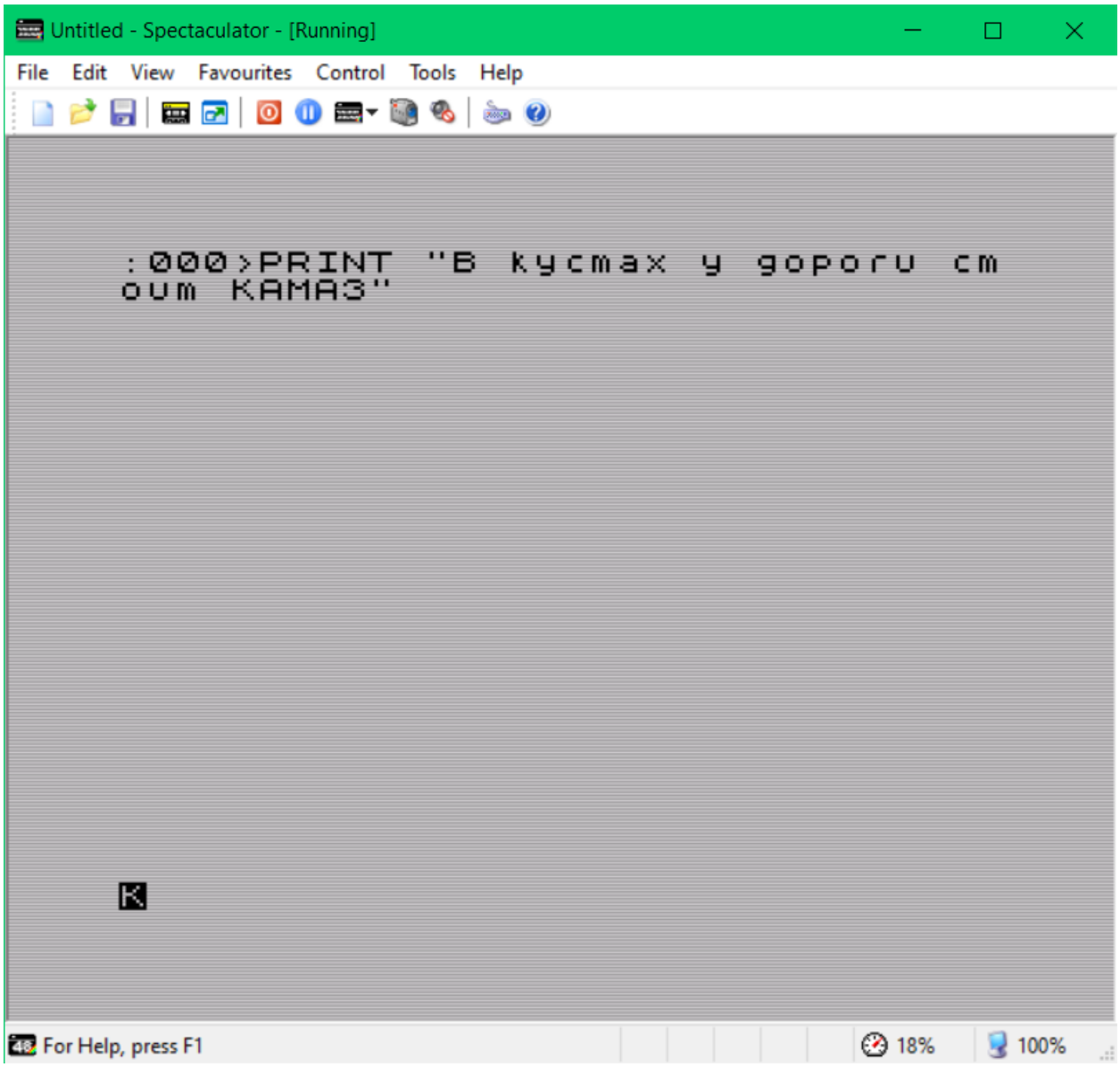


Рис. 178. Отображение на экране BASIC строки с номером 10000.

Ну и где тут анонсированный «0000»? Вообще-то строка стала с номером «:000». Алгоритмически вышестоящая программа выглядит так:

```
Debugger
Dec
Go To 6677
Add Breakpoint 6677
Trace
BASIC ← 10000 PRINT "B kycmax y goporu cmoum KAMA3"ENTER
Remove Breakpoint 6677
AF ← AF-1
Trace
```

Добавьте еще одну строку к существующей:

```
14567 PRINT "OKA ugem TAPAHOM HA KPA3"
```

На этот раз можно искусственно синтезировать по алгоритму «God Mode»:

```
Debugger
Dec
```

```
Go To 23560
23560 ← 13
23611 ← 32
23627 ← 23824
23641 ← 23825
23649 ← 23827 23827 23827
23792 ← 56 231 28 0 245
23797 ←""OKA ugem TAPAHOM HA KPA3
23822 ← 34 13 128 13 128
Trace
```

Вводите:

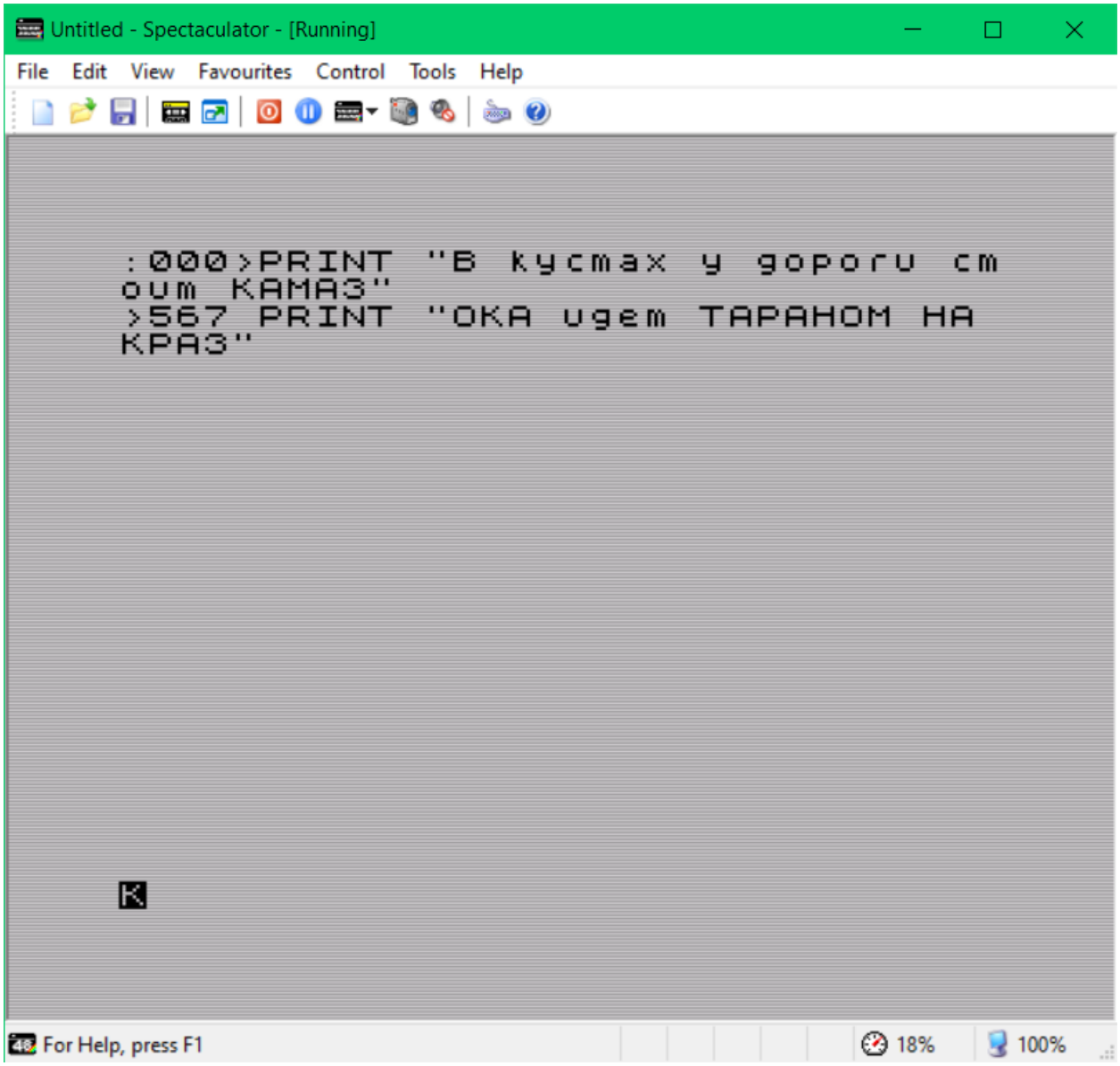


Рис. 179. Добавление строки 14567 к BASIC программе методом синтеза.

Теперь из 14567 в реальности получилось число «>567». Уже вырисовывается закономерность, поэтому возвращаюсь к недосказанной фразе из начала главы. Такие строки в народе называются…. Обманул я. Никак они никем не назывались… до последнего момента, пока за дело не взялся я. Теперь этому типу строк дано определение: «символьно-числовые». Прежде, чем приступить к принципу кодирования и отображения символьно-числовых строк, нужно проверить и второе книжное утверждение:

Известным способом защиты является также занесение значения больше 9999 в ячейки памяти, обозначающие номер строки БЕЙСИКа (первый байт - старший). Если он размещается в границах 10000-16383, а на листинге это выглядит, например, 0000 (вместо 10000) и строки невозможно вызвать на редактирование <EDIT>, если же превышает 16384 - дальнейшая часть программы считается не существующей.

Рис. 180. Вырезка из книги «Тайники ZX-Spectrum».

Попробуйте взять строку « : 000» на редактирование, а потом ввести:

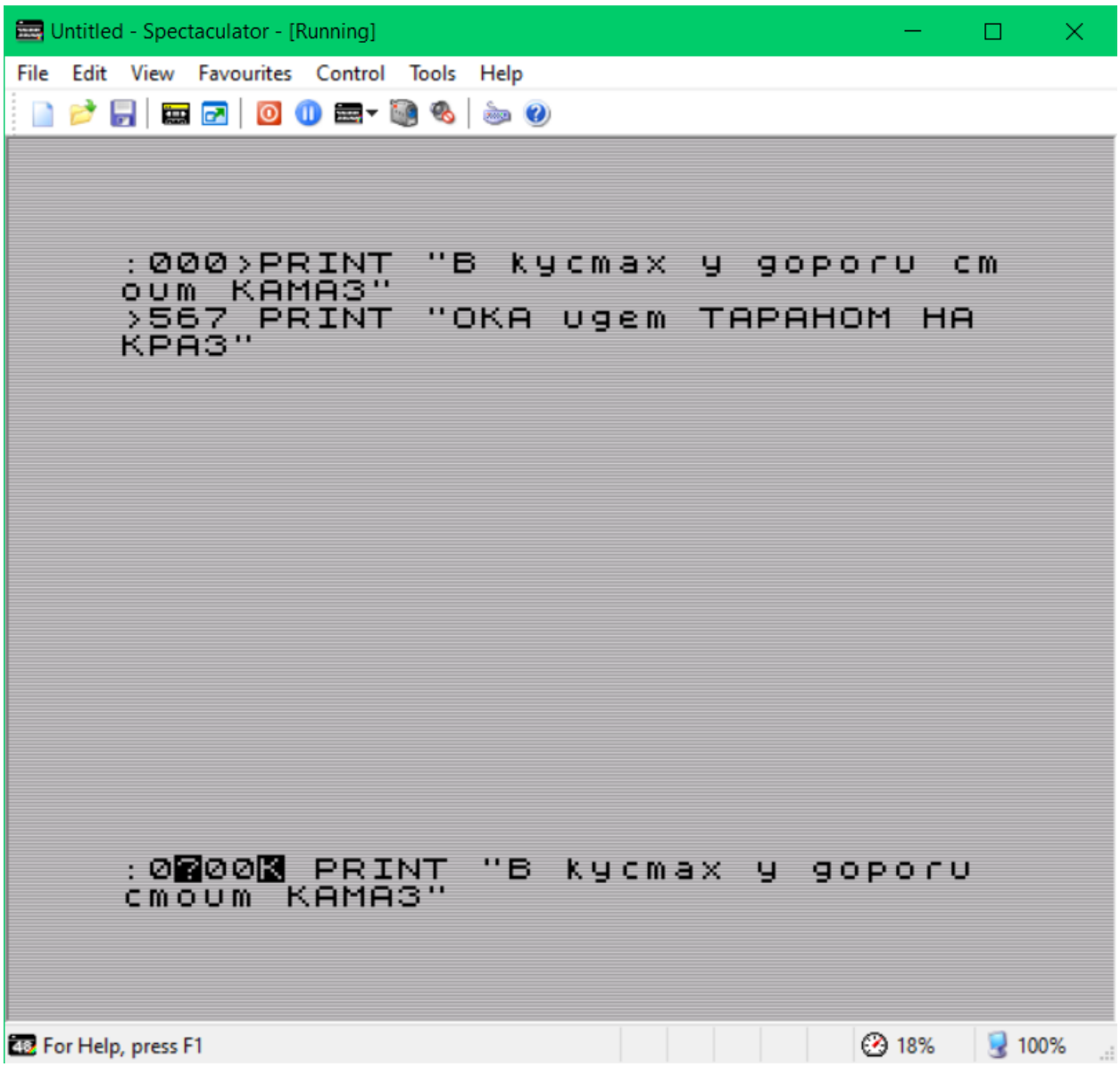


Рис. 181. Попытка обратно ввести символьно-числовую строку 10000 после взятия на редактирование.

И где тут не взять на редактирование??? Строка прекрасно взялась. Снова обманули. Другое дело, что из-за инородной символьной составляющей её не ввести обратно из буфера редактора. Чтобы избавиться от строки, придётся стереть как минимум номер, но можно и всю.

Теперь нужно её запустить. Обращение к строке в виде «GO TO : 000» не выйдет, что наглядно показал предыдущий пример. Такой номер тупо не пропускает анализатор и возвращает в редактор. Но ничто не мешает обратиться по реальному номеру. Для проверки наберите натуральным способом на клавиатуре GO TO 10000:

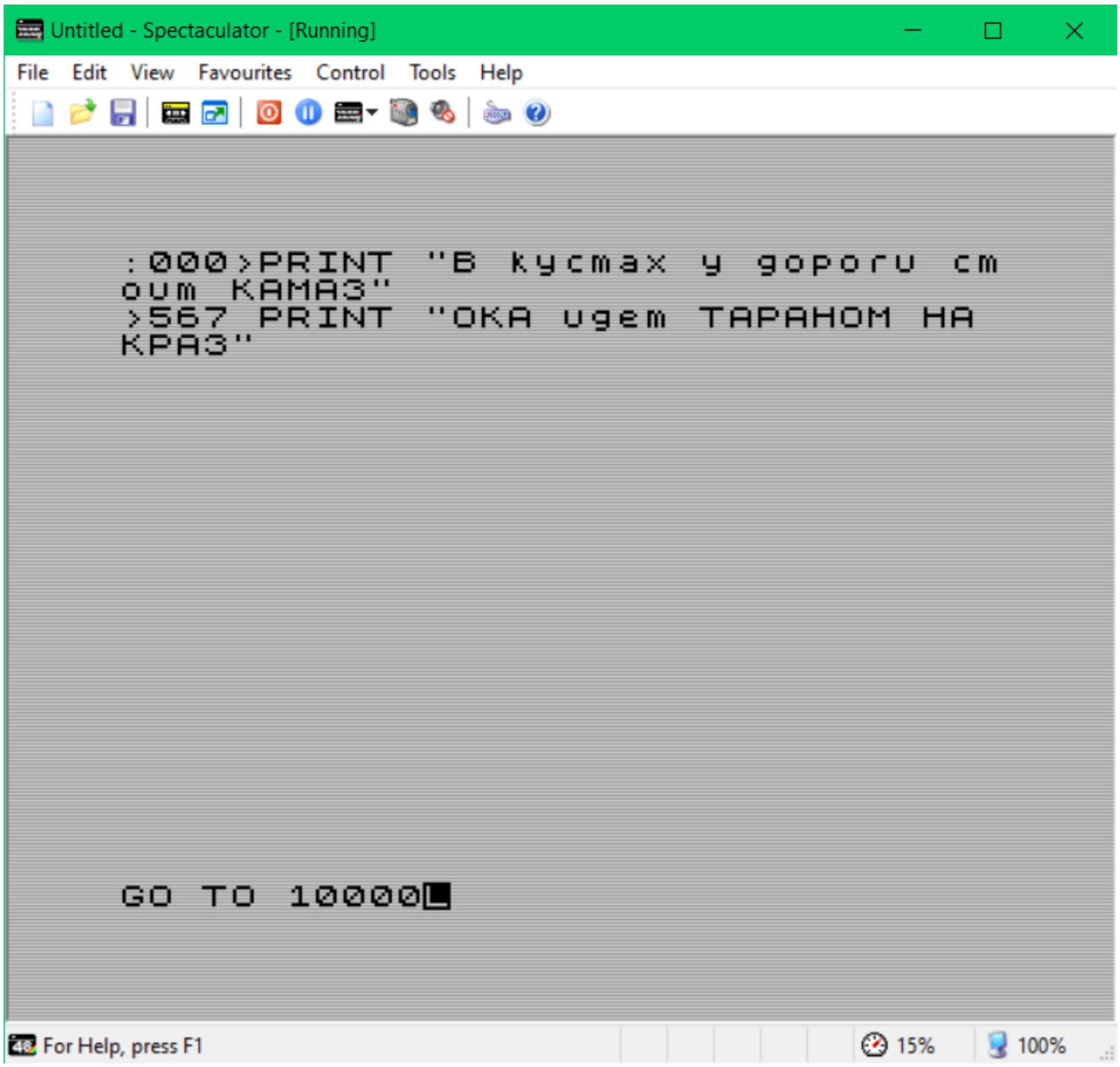


Рис. 182. Набор команды GO TO 10000 натуральным способом для запуска одноимённой строки.

А теперь введите:

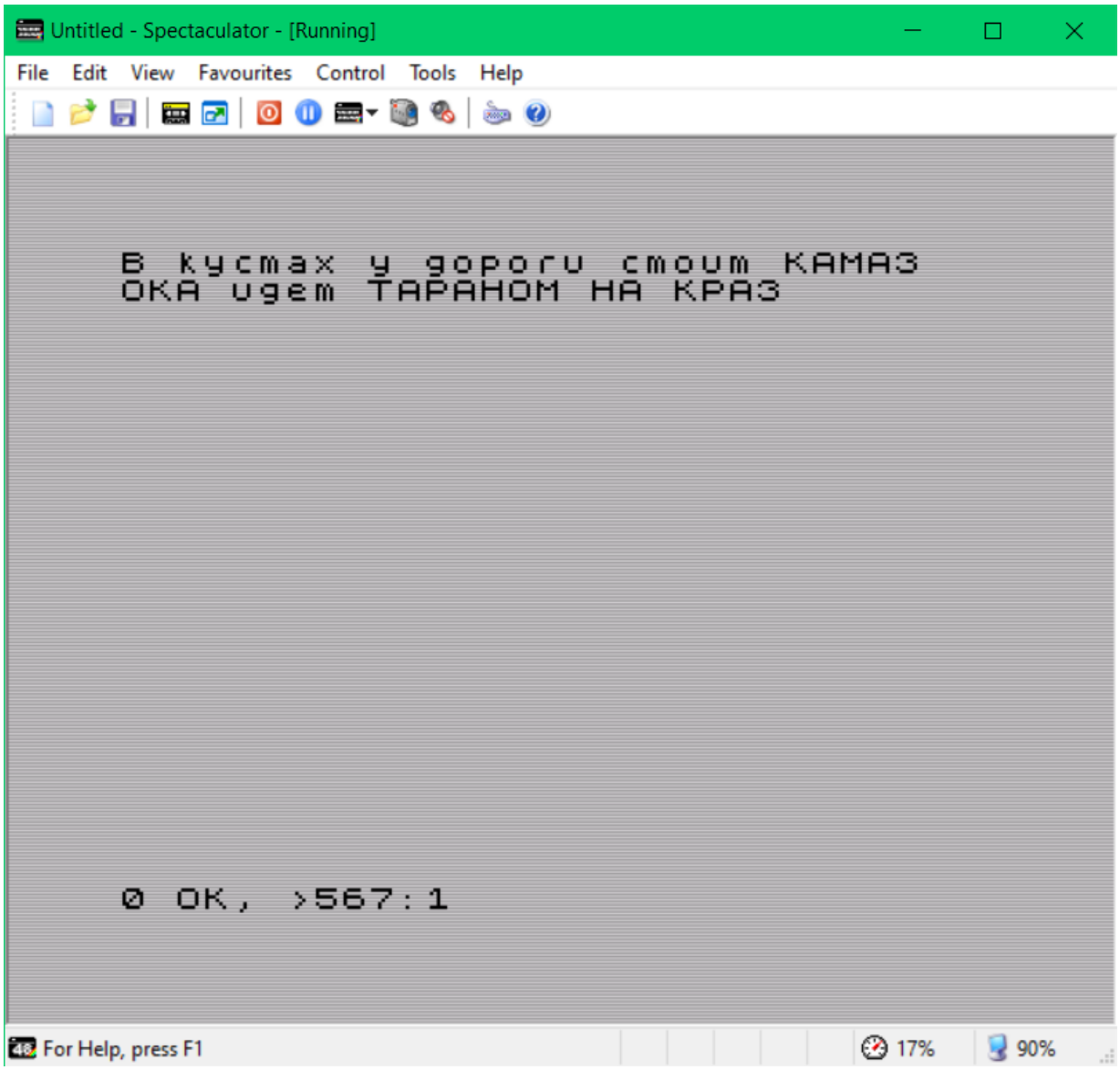


Рис. 183. Успешное выполнение «символьно-числовых» строк по команде GO TO 10000.

Программа выполнилась с сообщением «`0 OK, >567:1`». Если вы попытаетесь набрать `RUN`, то обе строки также спокойно запустятся.

А вот теперь предлагаю разбираться с теорией формата «символьно-числовых» строки, и заглянуть в одну книжку уровнем повыше:

```
ПОДПРОГРАММА 'REPORT AND LINE NUMBER PRINTING' ('Печать сообщений и номеров строк')
Точка входа OUT-NUM-1 приведет к тому, что число в регистровой паре BC будет
напечатано. Однако, любое значение свыше '9.999' не будет печататься правильно.
Точка входа OUT-NUM-2 приводит к тому, что напечатается число
```

```
непрямо адресованное регистровой паре HL. На этот раз появится необходимое количество
начальных пробелов. Числа будут правильно печататься до величины '9999'.

1A1B OUT-NUM-1    PUSH  DE              Записать через под-
                  PUSH  HL              программу других регистров.
                  XOR   A               Очистить регистр A.
                  BIT   7,B             Переход вперед, чтобы
                                        напечатать 0 вместо
```

Рис. 184. Фрагмент из книги «Полное описание ПЗУ ZX-Spectrum».

Ну как бы я и так вижу, что оно печатается неправильно, но мне нужен механизм и закономерность. Короче… ну вы поняли. Опять лезть и выяснять с помощью Стрелочки:

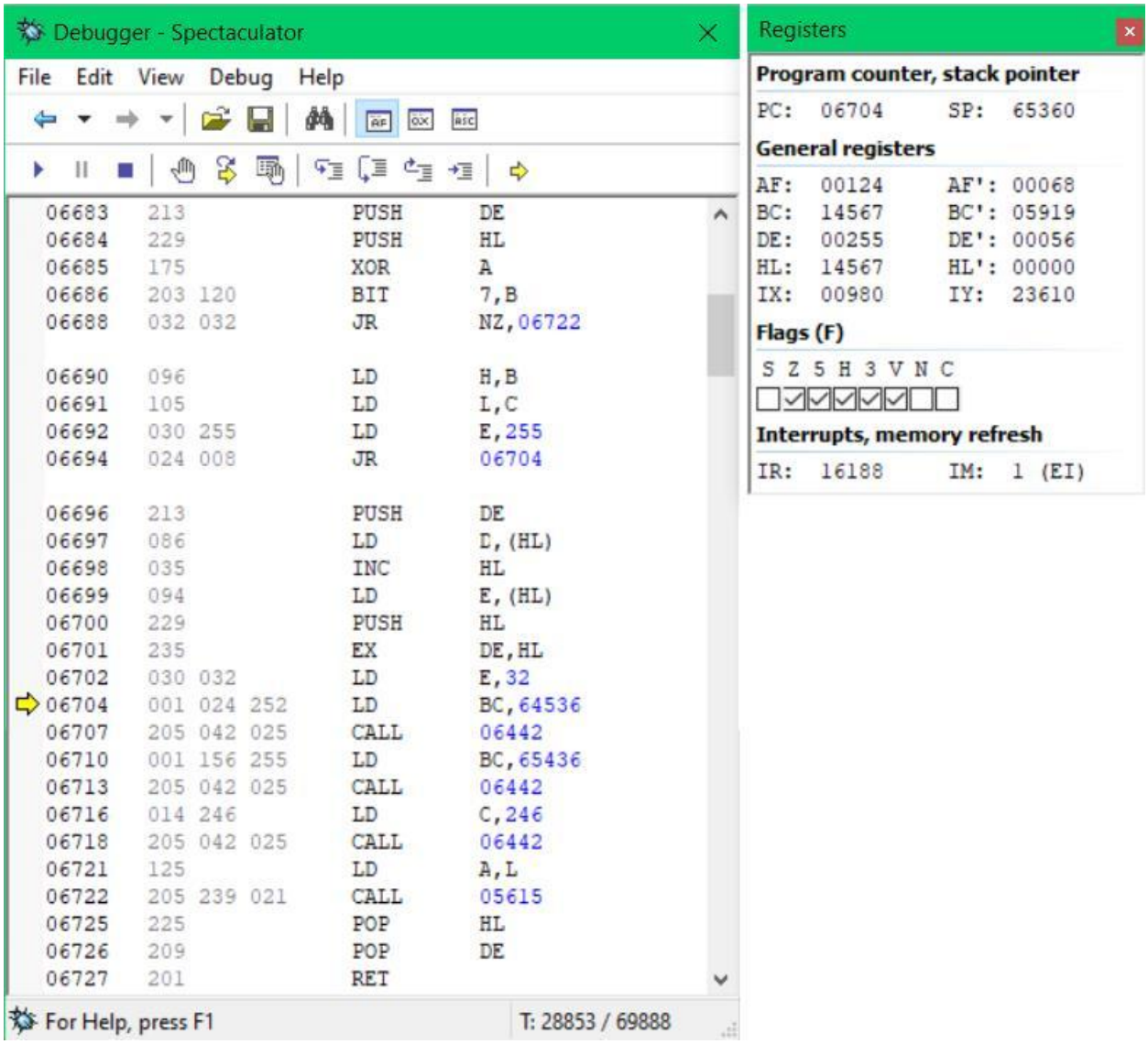


Рис. 185. Программа OUT NUM-3. Процесс кромсания и вывода номера 14567 на экран.

Ну вот и всё встаёт на свои места. Из вырезки книги видно, что за упаковку/распаковку чисел по цепочке цифр и печать символов отвечает пакет программ `PRINTING CHARACTERS IN A BASIC LINE` по адресу 6437. За несколько проходов он способен корректно выводить на экран даже числа, свыше 65535. Это понятно.

Однако для вывода 4-х значных номеров строк BASIC программы берётся усечённый комплект с точкой входа в подпрограмму `OUT NUM-2` по адресу 6696. Во время захода, в «HL» содержится адрес начала BASIC строки, который перед печатью преобразуется в номер.

С адреса 6702 начинается непосредственная распаковка числа на последовательность символов для вывода на экран. Обратите внимание, 5-го знака числа не предусмотрено, поэтому у значений свыше 9999, вместе с 4-м числом забирается верхний остаток. На примере выше, идёт процесс разрезания числа 14567 и посимвольного вывода на экран. В шаге первом по адресу 6704 отрезается верхушка «14». За следующий проход заберётся число «5», следом «6» и, наконец, «7».

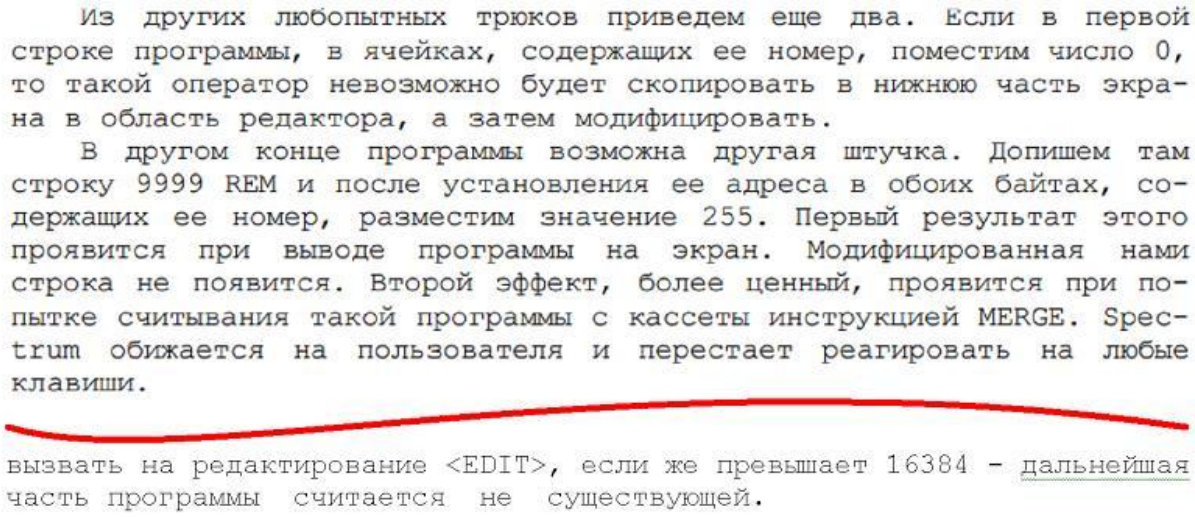
Цифры 14 не существует и защиты от вывода тоже, поэтому для отображения захватываются коды символов ASCII из таблицы, которые расположены дальше 9. Таким образом, для 10 это двоеточие, 11 – точка с запятой, 12 – знак <, 13 – равно, 14 это будет >

и так далее до максимального возможного числа 65535, которое будет выглядеть как «q535». Следовательно, число 14567 высветится на экран как «>567». Вот и весь секрет.

Глава 17
Невидимые выполняемые строки (16384-32767)

Краткое содержание невидимые строки, запуск, снятие невидимости

Вот очередь дошла и до самых загадочных строк, которые начинаются с номера 16384. На первый взгляд они и, правда, загадочные, но только при поверхностном знакомстве. Предлагаю посмотреть, как об этих строках думали люди в старину, когда трава была зеленее, серп прилагался к молоту ☭, а посмотреть на работу ZX-Spectrum'а снаружи можно было только в научно-фантастической литературе через видеотелевизор:



Из других любопытных трюков приведем еще два. Если в первой строке программы, в ячейках, содержащих ее номер, поместим число 0, то такой оператор невозможно будет скопировать в нижнюю часть экрана в область редактора, а затем модифицировать.

В другом конце программы возможна другая штучка. Допишем там строку 9999 REM и после установления ее адреса в обоих байтах, содержащих ее номер, разместим значение 255. Первый результат этого проявится при выводе программы на экран. Модифицированная нами строка не появится. Второй эффект, более ценный, проявится при попытке считывания такой программы с кассеты инструкцией MERGE. Spectrum обижается на пользователя и перестает реагировать на любые клавиши.

вызвать на редактирование <EDIT>, если же превышает 16384 - дальнейшая часть программы считается не существующей.

Рис. 186. Вырезки из книг «Тайники ZX-Spectrum» разных изданий.

Для разминки предлагаю синтезировать следующую строку:

```
17408 PRINT "Вuхрь-6М"
```

Для чего нужно набрать и ввести:

```
Debugger
Dec
Go To 23560
23560 ← 13
23611 ← 32
23627 ← 23771
23641 ← 23772
23649 ← 23774 23774 23774
23755 ← 68 0 12 0 245 34 66 117 120 112 98 45
23767 ← 54 77 34 13 128 13 128
Trace
```

После ввода алгоритма ничего не произошло, а в нижнем углу привычно замигал курсор. Учитывая обрывки информации из разнообразных вариаций «*Тайников ZX-Spectrum*», чего-то подобного стоило ожидать.

Для прояснения ситуации откройте отладчик на адресе 23755:

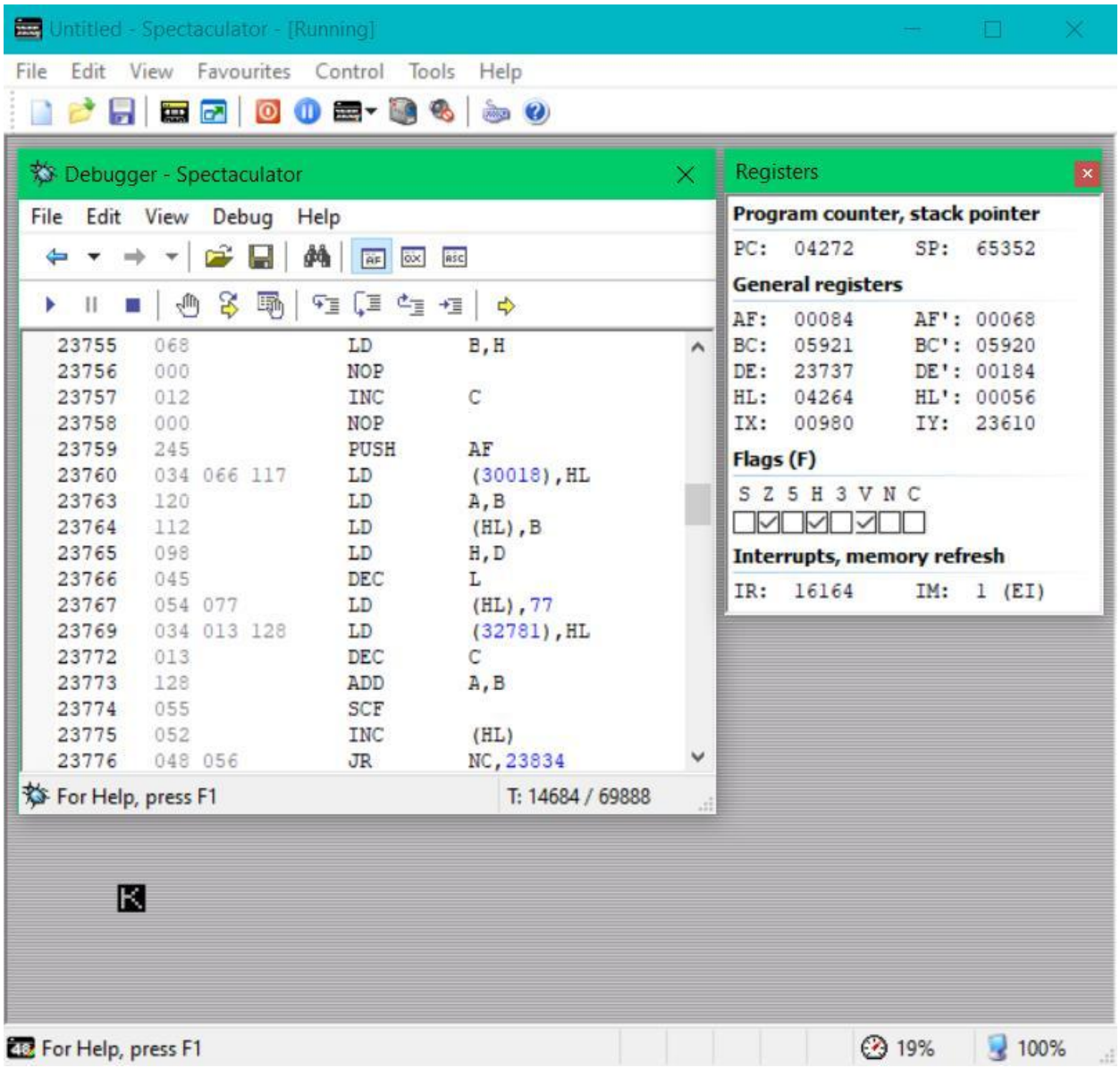


Рис. 187. Расположение в памяти неотображаемой строки с номером 17408.

Введенная строка пока на месте, просто не отображается. Выходите из отладчика и натуральным способом, а затем наберите и введите такую строку:

```
GO TO 17407
```

Снова никакого эффекта. А теперь наберите `GOTO 17408`:

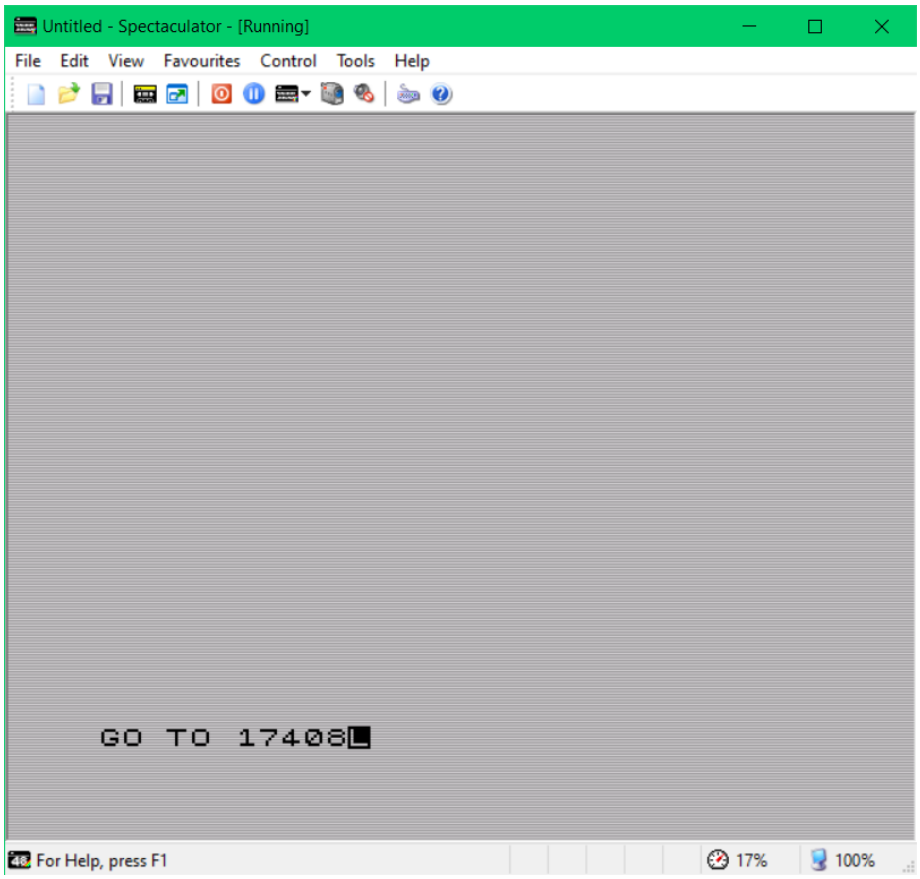


Рис. 188. Набор команды GO TO 17408 натуральным способом.

Вводите.

О Чудо! Строки нет, но работа от выполнения появилась:

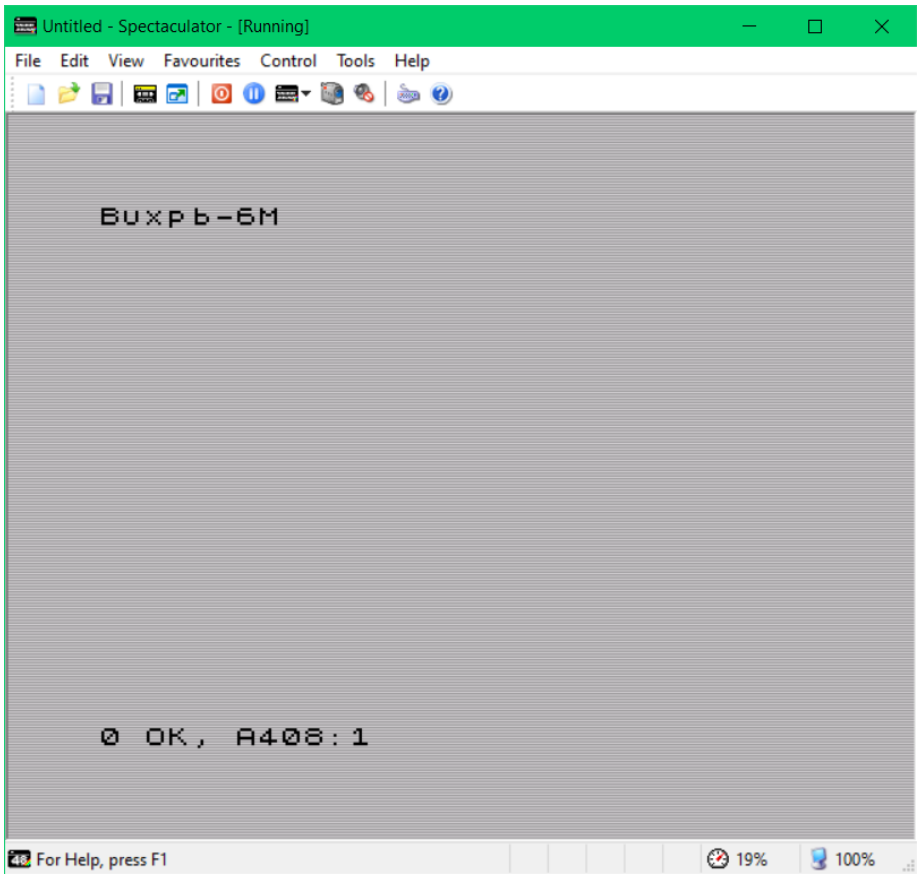


Рис. 189. Отображение на экране выполнившейся неотображаемой строки 17408.

Так что считать программу со строками свыше 16383 несуществующей, не совсем корректно. Она же выполнилась и даже с сообщением «0 OK, A408:1». То есть строка существует. Она просто невидимая, но её номер, по-прежнему, отображается «символьно-числовым» способом. Пусть такие строки и называются «невидимыми», а поскольку они еще и выполняются, то «невидимые выполняемые».

Нетрудно догадаться, что «невидимые» строки являются разновидностью «символьно-числовых» строк высшей нумерации. Несмотря на то, что они запускаются, у невидимых строк имеется множество отличий. Из опыта убедились, что запустить невидимую «символьно-числовую» строку можно лишь по прямому обращению, когда известен её точный номер.

И все-таки, почему они невидимые? Нужно разбираться и искать причину. Для этого достаточно просто вспомнить формат переменных, которые рассматривались несколькими главами ранее.

Напомню, на чистом компьютере, переменная VARS (23627) указывает в тот же самый адрес, что и PROG (23635). Эти области дружественные и не изолированы друг от друга маркерами. Граница BASIC строк и начала области переменных условная и визуально проходит по ENTER'у последней введённой строки. Глухой границей они разделены только с областью ввода строки, на которую указывает E_LINE (23641).

Ну и другой не менее важный вопрос: а что это такое?

А вот именно сейчас это и выяснится, но вначале позволю еще немного интриги. Добавьте к концу невидимой строки переменную:

```
LET d$=''▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒''
```

Для чего наберите и введите следующую алгоритмическую программу:

```
Debugger
Dec
Go To 23641
23641 ← 23787 23787
23649 ← 23789 23789 23789
23771 ← 68 12 0 75 112 121 109 111 32 67 111 99
23783 ← 101 109 33 128 13 128
Trace
```

Не очищая экран, наберите строку PRINT d$:

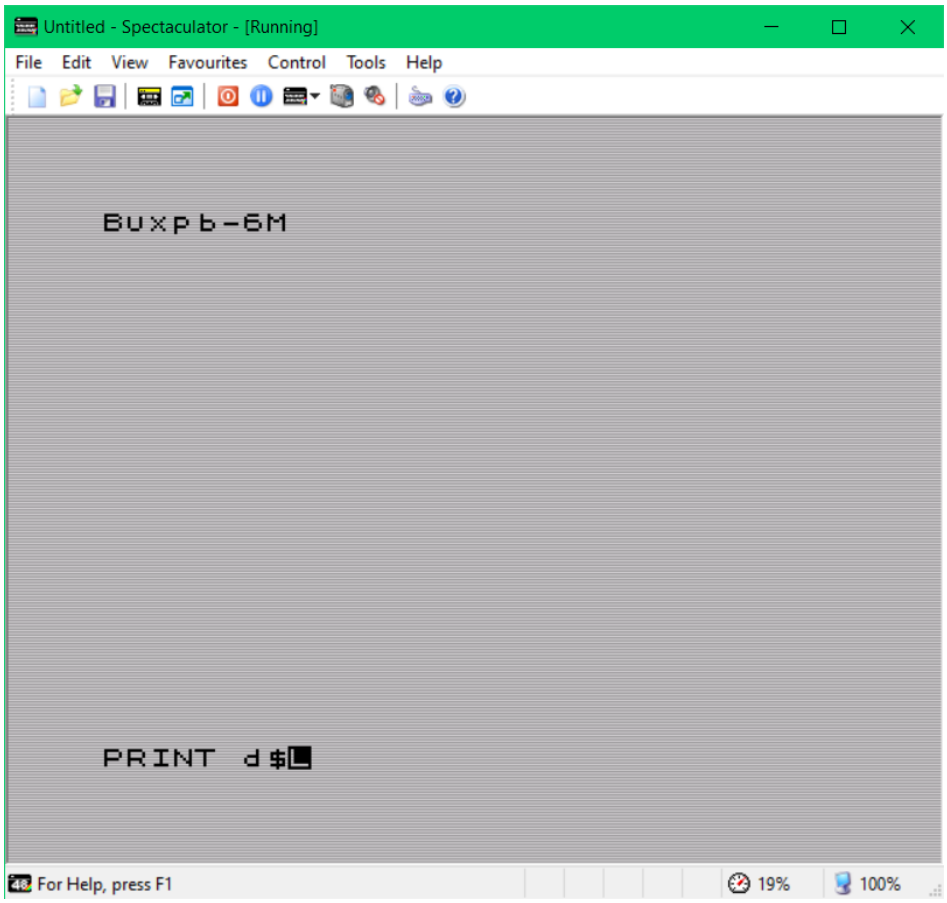


Рис. 190. Ввод команды для вывода синтезированной символьной переменной.

Введите:

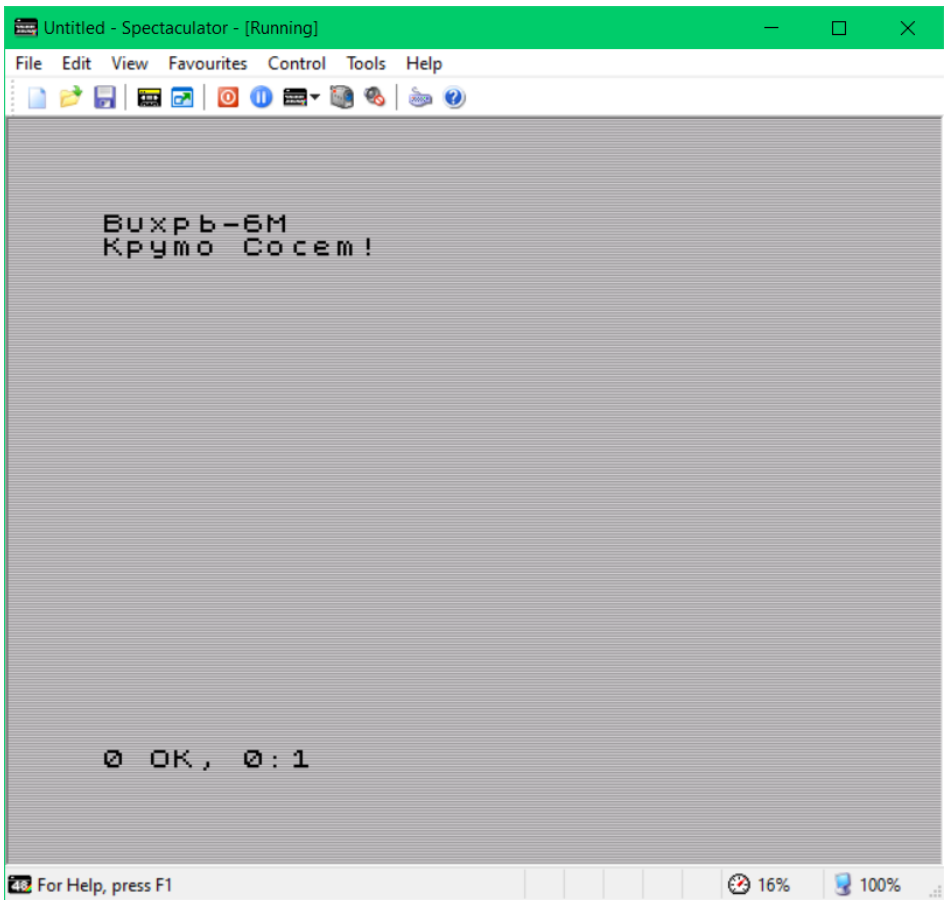


Рис. 191. Вывод окончания фразы из искусственно синтезированной символьной переменной.

Круто сосёт пыль, причём за сущие копейки! Как в рекламе. Ну а ниже сам виновник торжества:

Рис. 192. Пылесосы ВИХРЬ-6М (Модель ЭП-6М, тип ПН-600). Фрагменты фото из интернета.

Знакомьтесь: легендарный пылесос «ВИХРЬ-6М» Ленинградского машиностроительного объединения «Спутник» (Волковский пр. 6). Модернизированные модели ЭП-6М синего и тёмно-бирюзового цвета, изготавливались по ТУ27-09-1254-76.

Ностальгические звуки невозможно забыть даже спустя 40 лет. Генеральная уборка квартиры начиналась с самого утра буднего дня, когда большая часть населения квартиры уходила на работу. После подготовительных работ и протирки высотных поверхностей, бабушка доставала такой вот пылесос и начиналась пропылесошивание паркета, матрасов и дивана. Наслаждаясь мелодичным гулом, я смотрел в черное выходное отверстие, на синие искорки, летящие от щеток электродвигателя…

А сейчас просто откройте отладчик и сравните невидимую строку с переменной:

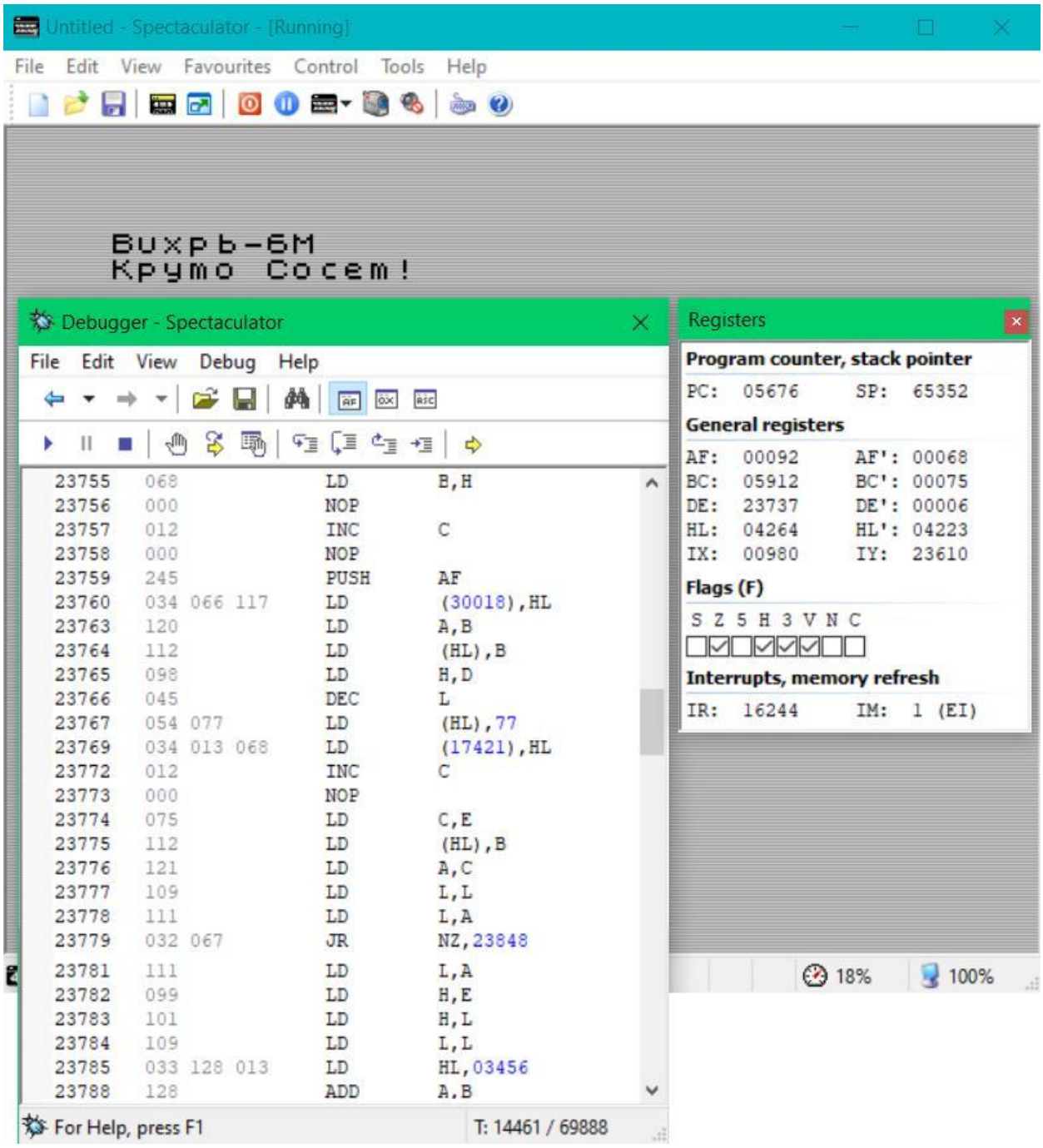


Рис. 193. Расположение невидимой строки и переменной в памяти.

Правда, много общего у этих блоков данных?

А теперь попробую поставить их друг на друга, разбив на структуру:

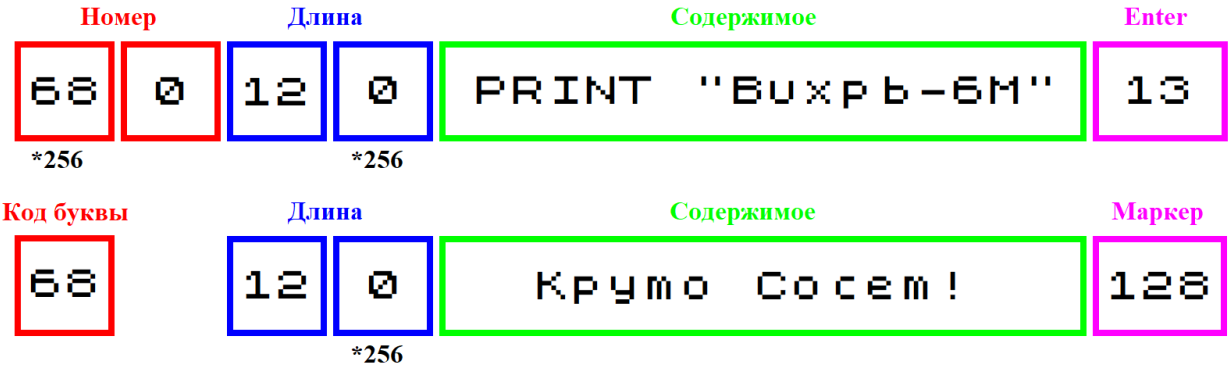


Рис. 194. Сравнение структуры невидимой строки и однобуквенной символьной переменной.

Вот и ответ сразу на два вопроса.

1) Даже не прибегая к помощи «Стрелочки» наглядно видно, что большие значения, засунутые в первый байт номера строки, являются маркером и признаком переменной, а значит, такую строку просто не нужно выводить на экран.

2) Именно для удобства идентификации младший и старший байт номера строки поменяны местами.

Предлагаю посмотреть процесс обновления на примере текущей программы. Для любого действия, связанного с обновлением экрана, требуется перерисовка всего пространства с BASIC строками. За печать строк отвечает комплекс подпрограмм `PRINT A WHOLE BASIC LINE` по адресу 6229, о чём неоднократно упоминалось ранее.

А вот теперь нажимайте `ENTER` и понаблюдайте, какие процессы начнутся в «Волшебной Стране»:

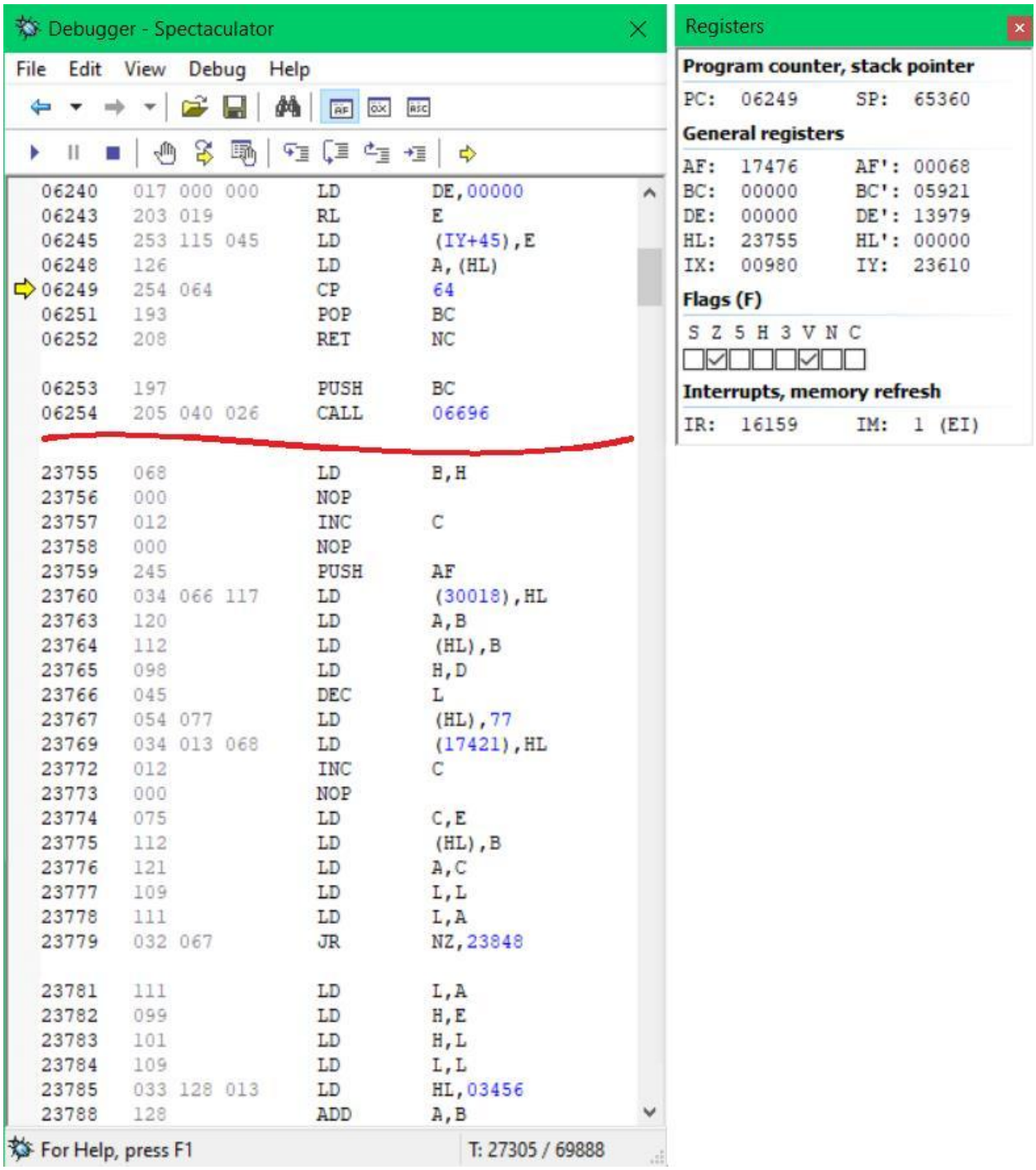


Рис. 195. Окончание вывода на экран BASIB строк.

Идёт подготовка к процессу вывода строк на экран. В «HL» уже лежит адрес строки, которую нужно вывести. Невидимая строка стоит первой и единственной, её адрес будет 23755 и совпадать с началом BASIC области. Как видно из схемы на прошлом рисунке, первым фрагментом номера строки стоит старший байт (*256). И вот Стрелочка считывает его и смотрит. Если значение больше 64, значит, по задумке разработчиков, это элемент переменной BASIC или маркер конца области. А переменные всегда стоят в конце программы, да и строка не может иметь номер свыше 9999, то есть значения 39 (39*256+15=9999). Следовательно, BASIC программа кончилась и можно выходить из программы вывода.

Больше печатать нечего, поэтому в 6252 командой `RET NC` происходит выход из программы по C☐ *отсутствию галочки «C»*. Мало того, следом стоящие строки с любыми меньшими номерами тоже не выведутся на экран, потому как процесс оборвётся на первой обнаруженной строке с головным числом, большим 63. Вот так строки с номерами от 64*256=16384 становятся невидимыми.

Нетрудно догадаться, что поймав Стрелочку на шарик перед адресом 6252, достаточно установить C☑ галочку «C» (или прибавить к текущему значению «AF» единицу) и произойдёт вывод следом стоящей невидимой строки. После вывода строки Стрелочка снова окажется тут. Ну а если повторить манипуляции снова, на экран выведется и переменная с областью вводимой строки.

Ну что, всё по аналогии с предыдущими экспериментами. Попробуйте вывести эту невидимую строку. Для этого выполните следующий алгоритм:

```
Debugger
Dec
Go To 6252
Add Breakpoint 6252
23560 ← 13
23611 ← 32
Trace
AF ← AF+1
Trace
Remove Breakpoint 6252
AF ← AF+1
Trace
```

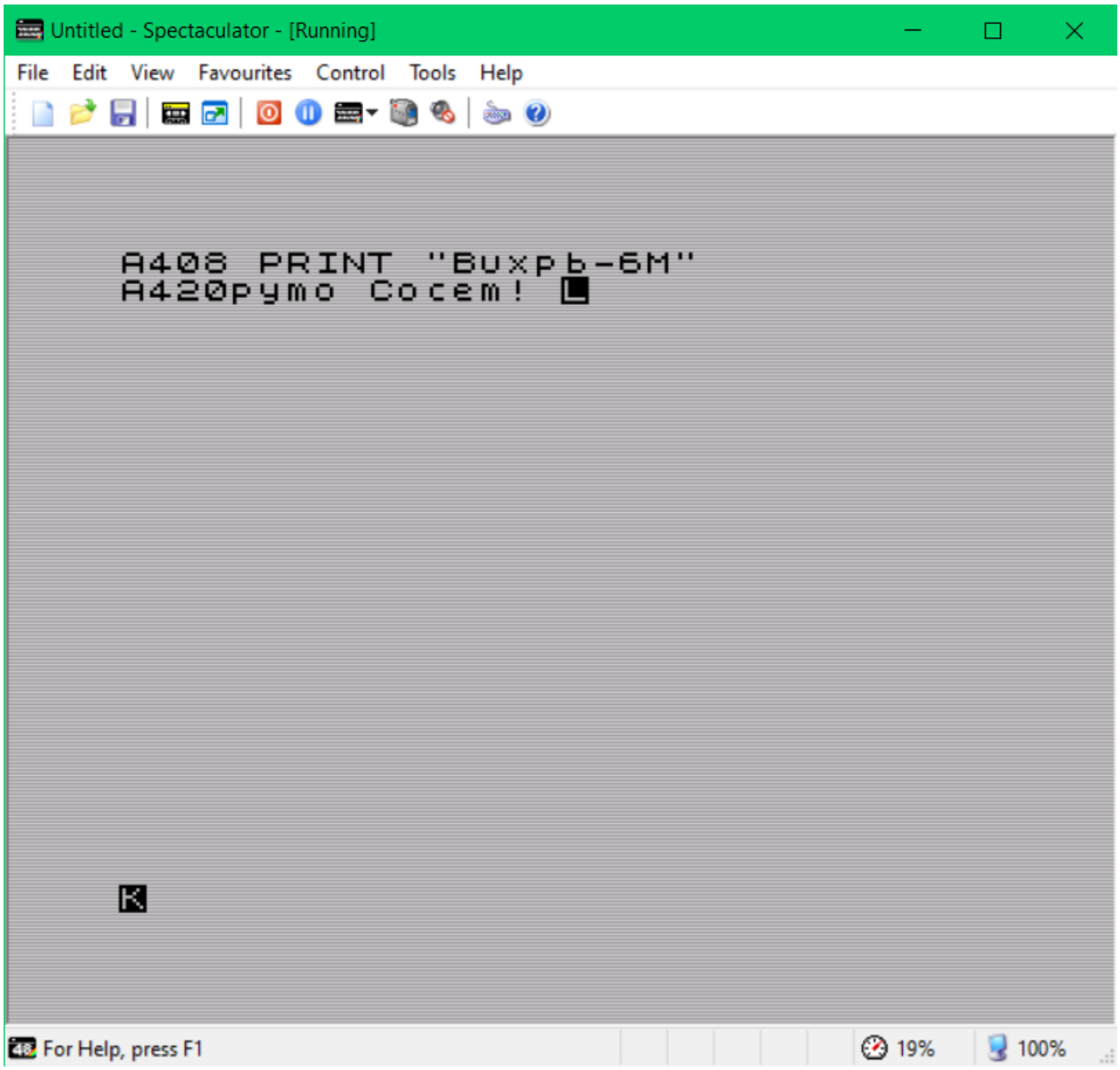


Рис. 196. Вывод на экран невидимой строки с переменной и курсором.

Красота! На экране появилась заветная невидимая строка, содержимое переменной d$ (естественно без первого символа), которая стояла следом и… буфер строки с курсором.

Снова нажмите ENTER:

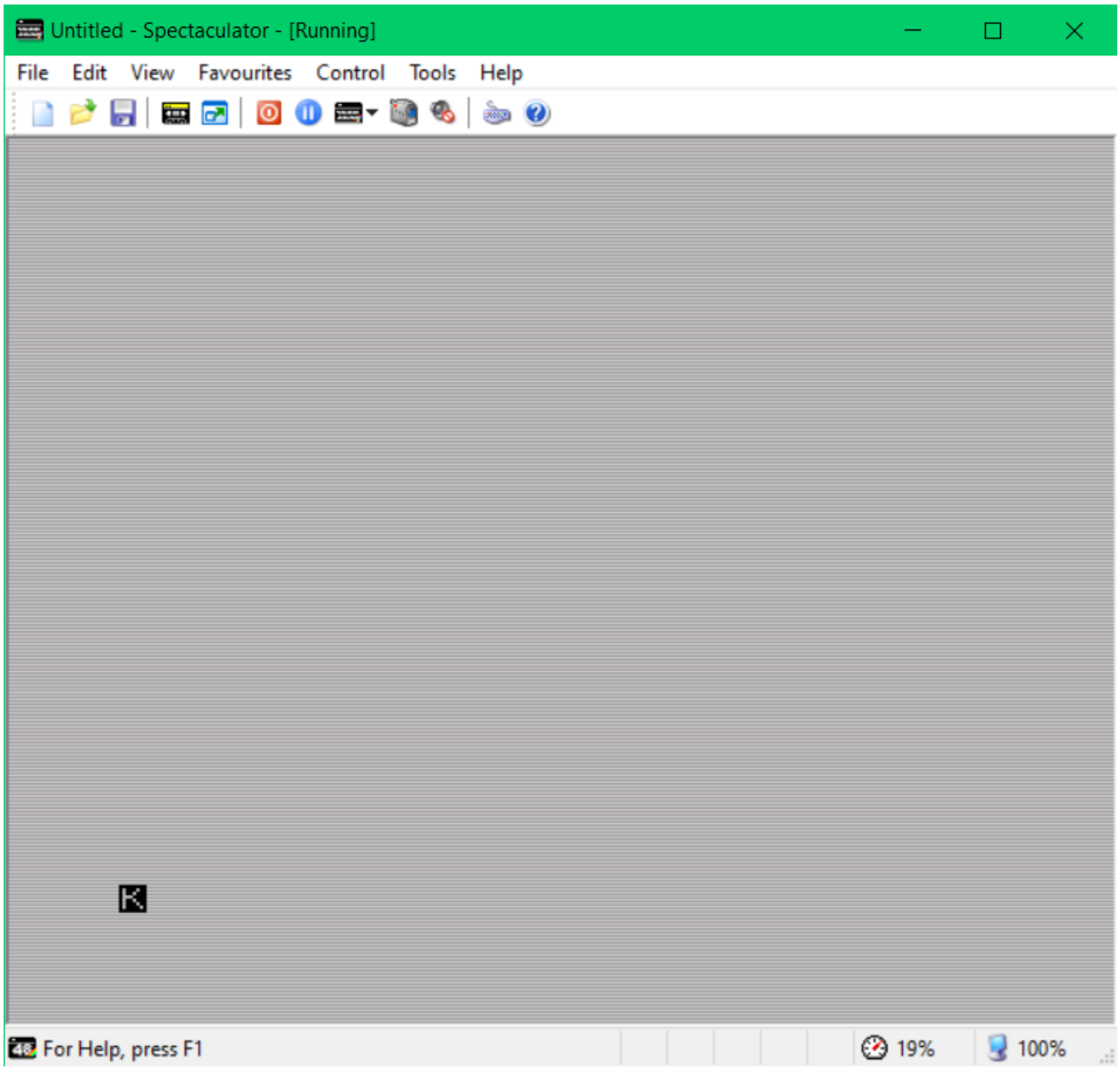


Рис. 197. Исчезновение невидимых строк с переменной после нажатия ENTER.

И волшебство исчезло. Строка 17408 с символьной переменной снова стали невидимыми. Таким способом можно вывести на экран строку с любым номером, да и вообще любой мусор. Нажмите Reset и можно приступать к самой жестяной категории строк.

Глава 18
Невидимыме невыполняемые строки (32768-65535)

Краткое содержание: невидимые строки, обход ограничений, трудности запуска

Со строкой 17408 примерно понятно, а дальше? А что будет, если создать строку 40000 или вообще 65535, как советовали в книжках «Тайники Спектрума», а потом попробовать запустить?

Предлагаю синтезировать следующую BASIC программу на электрическую тему:

```
40000 PRINT "В РОЗЕТКЕ uckpum КОНТАКТ"
65535 PRINT "НА НАВЕСЕ Bucum РН-100"
```

На языке алгоритма наберите и введите программу:

```
Debugger
Dec
Go To 23560
23560 ← 13
23611 ← 32
23627 ← 23817
23641 ← 23818
23649 ← 23820 23820 23820
23755 ← 156 64 28 0 245
23760 ←""B РОЗЕТКЕ uckpum КОНТАКТ
23785 ← 34 13 255 255 26 0 245
23792 ←""НА НАВЕСЕ Bucum РН-100
23815 ← 34 13 128 13 128
Trace
```

А теперь натуральным способом наберите `GO TO 40000`:

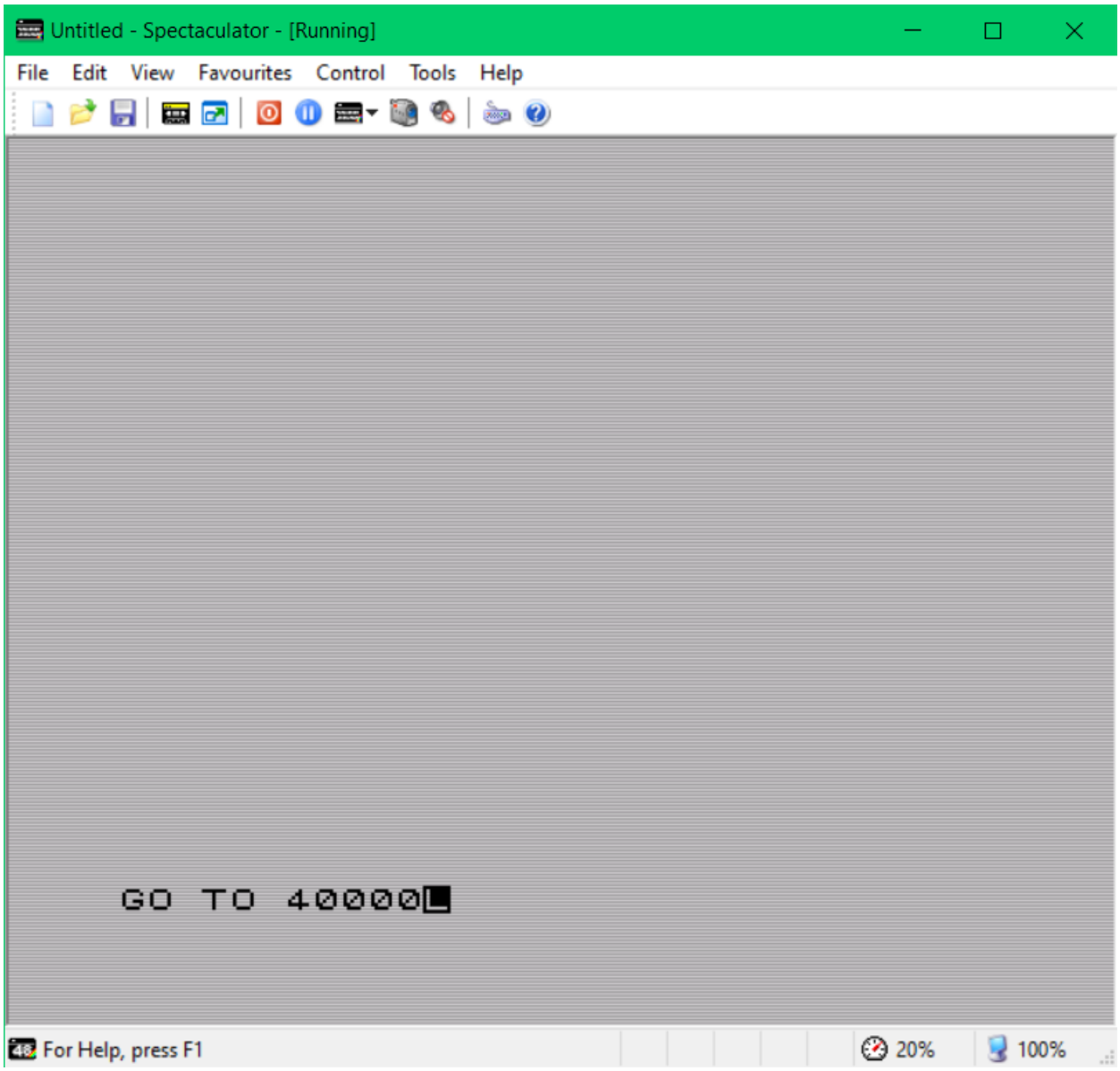


Рис. 198. Процесс набора команды GO TO 40000 натуральным методом.

Введите:

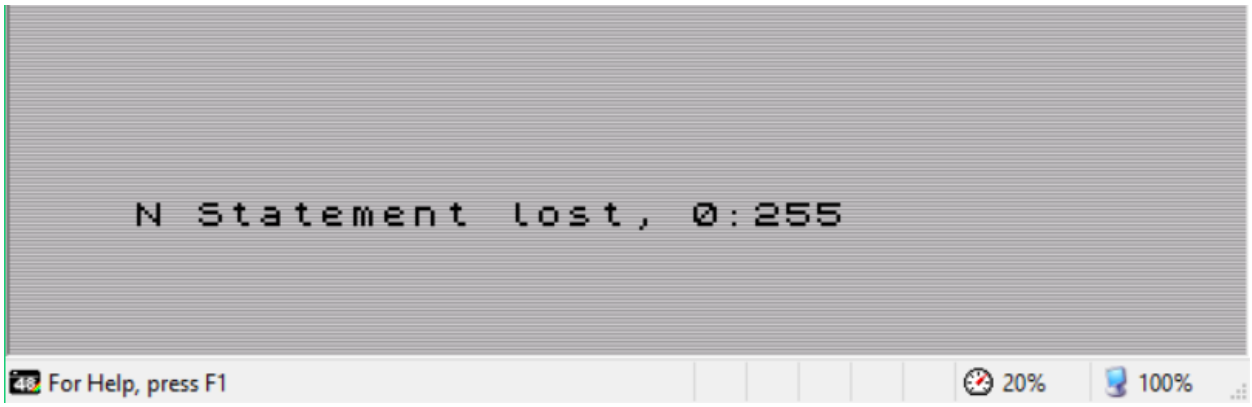


Рис. 199. Сообщение «Statement Lost» при попытке ввести GO TO 40000. Фрагмент нижней части экрана.

На этом добрая сказка о запускаемых строках кончается, и появляется первое сообщение об ошибке, причём достаточно экзотическое – «`N Statement lost`». Ладно, а теперь введите `GO TO 65535`:

Рис. 200. Сообщение «Integer out of range» при попытке ввести GO TO 65535. Нижний фрагмент экрана.

В результате прошлого опыта вы стали свидетелями важного события. Произошло открытие нового типа строк, которым присвоена классификация «невидимые незапускаемые». Кроме того, этот вид делится на подвиды: «невидимые незапускаемые по Statement lost» и «невидимые незаапускаемые по Integer out of range».

Нужно разбираться. Пока только представьте, что вы натуральным образом набрали и ввели команду `GO TO 65535`, открыли отладчик и наблюдаете за маршрутом.

Продвигаясь по главному циклу, голая безномерная строка проходит прямой путь сквозь все фильтры категорий, и с адреса 4864 направляется в комплекс подпрограмм `LINE RUN (7050)`. После подготовки начинается долгий путь идентификации команды с дроблением на группы, чтобы подобрать для нее нужную программу совершения действий.

Независимо от того, по какой команде выполнялся запуск (`GO TO` или `RUN`), в конечном итоге все дорожки снова сойдутся, и Стрелочка попадёт на подпрограмму `GO TO` по адресу 7783. Вот с этого момента дальнейшие события напрямую затрагивают рассматриваемый вопрос:

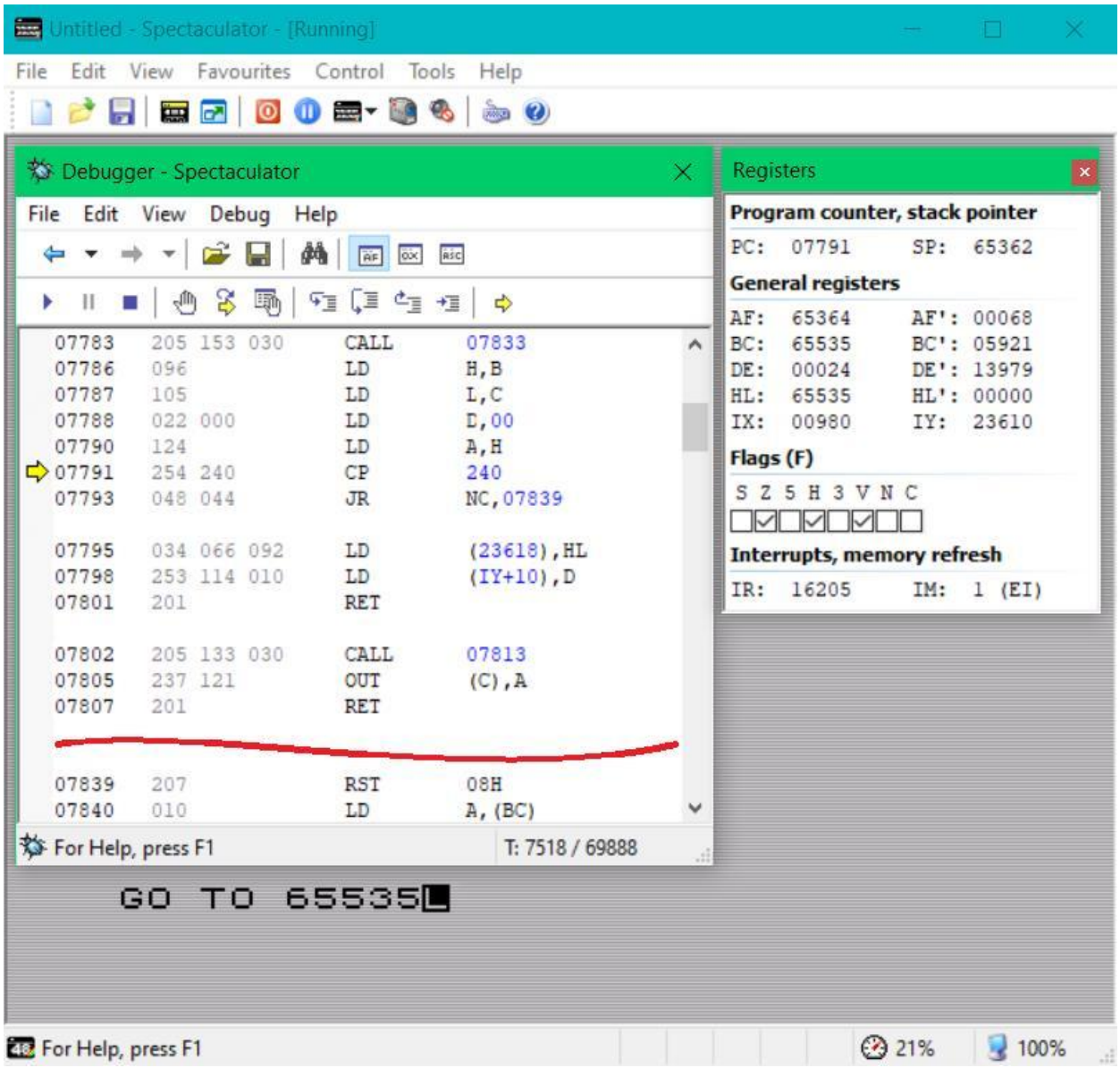


Рис. 201. Процесс остановки выполнения строки GO TO 65535.

Именно отсюда начинается самая важная часть, в которой рождается первое свойство невидимых строк. Подпрограммой `FIND INT-2 (7833)` формируется числовое значение из последовательности сырых символов, стоящих после команды (в текущем случае 65535). Вынырнув обратно, из «BC» в «HL» копируется номер строки. В «D» выставляется номер оператора в строке, с которого нужно начать выполнение. Следом берётся первый, он же старший (*256) байт номера строки и проверяется на очень ядрёный номер (`CP 240`). При встрече значения равного, или выше (240*256)=61440, галочка «C» снимается ☐. По следом стоящей команде `JR NC, 07839` выполнение программы прекращается и выдаётся сообщение об ошибке «`B Integer out of range`». Вот и первая преграда, которая рождает категорию строк «Незаапускаемые по Integer out of range». Из исследования понятно, что подвид строк «Невидимые незапускаемые по Integer out of range», имеют диапазон от 61440 по 65535.

Если номер строки в рамках допустимого на данном барьере проверки (по 61439), он записывается в переменную `NEWPPC (23618)`, а номер оператора для выполнения в `NSPPC (23620)`. Ну а после успешной записи параметров для `GO TO` происходит выход в программу `STMT-RET` по адресу 7030:

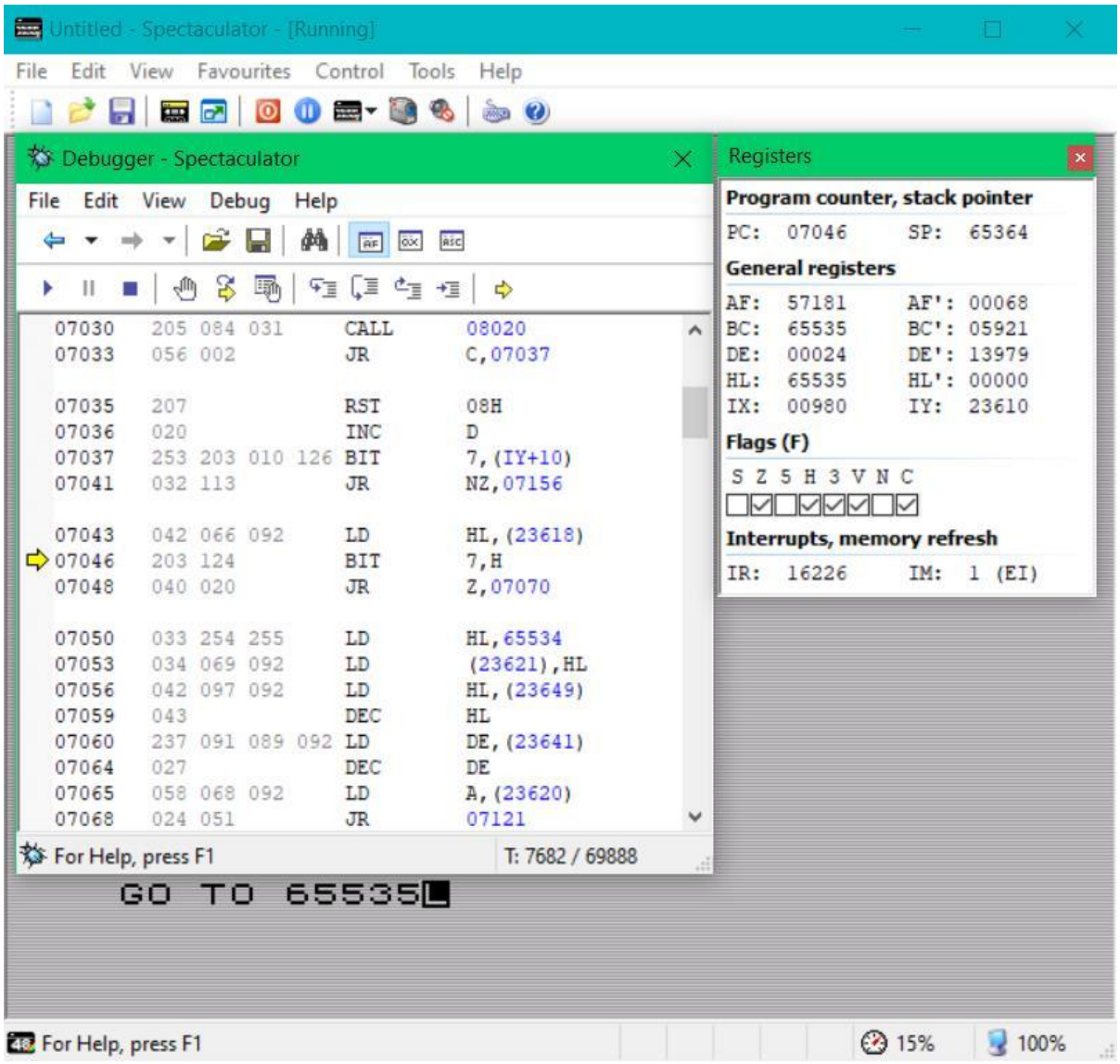


Рис. 202. Программа STMT-RET. Место появления ошибки при выполнении невидимых строк.

Первым делом вызывается программа `BREAK-KEY (8020)` и проверяется, были ли нажаты клавиши `BREAK` с `CAPS SHIFT` для вскрытия программы. Если да, значит, процесс выполнения прервётся и на экран выведется сообщение «`L BREAK into program`». Ну а если всё в порядке и ничего не нажималось, то можно идти дальше.

Не копируя значение, дистанционно прощупывается самый верхний бит переменной `NSPPC (23620)`. Если номер оператора превышает 128 и выше, значит, строка кончилась и нужно сделать выход.

А вот дальше стоит следующий барьер. Снова берётся значение строки из переменной `NEWPPC (23618)` и анализируется только самый верхний бит первого, в данном контексте *«старшего»* (*256) байта номера строки. (128*256=32768).

Если номер строки свыше этого значения, а подразумевается, что в данном случае он может быть только 65534 (*шифр введенной безномерной выполняемой строки, рассматривали в прошлых главах*), то убирается Z галочка «Z».

Поскольку номер строки выше 32767, то Стрелочка игнорирует команду перехода по галочке `JR Z, 7070` и продолжает выполнение программы. Провалившись на ступеньку вниз она оказывается… в первоначальной точке входа программы `LINE RUN (7050)`, на которую попала из главного цикла. А раз так, Стрелочка думает, что перед ней очередная новая строка, прибывшая из сортировочного центра главного цикла (4864).

В полной уверенности она заносит шифр «65534» в переменную PPC (23621) маркируя строку как выполняемую и не имеющую номера. Подготовив рабочие области к выполнению строки, с неправильными вводными данными, Стрелочка уходит в подпрограмму NEXT LINE (7121). Про номера строк уже всё давно забыли, и Стрелочка начинает выполнять введённую строку. Ничего хорошего из этого не выходит.

Так и не совершив переход к строке 65535, Стрелочка повторно сканирует введённую строку с GO TO, теряет галочку «Z», необходимую для продолжения процесса и проваливается на сообщение об ошибке «N Statement Lost» по адресу 7148.

Если строка прошла проверку, и номер попал в интервал по 32767 включительно, то происходит переход на 3-х этажную систему подпрограмм LINE NEW (7070) для определения адреса строки, которую нужно выполнить по номеру.

Не успев освоиться в новой программе, Стрелочка с первого же шага проваливается в подпрограмму LINE ADDR по адресу 6510:

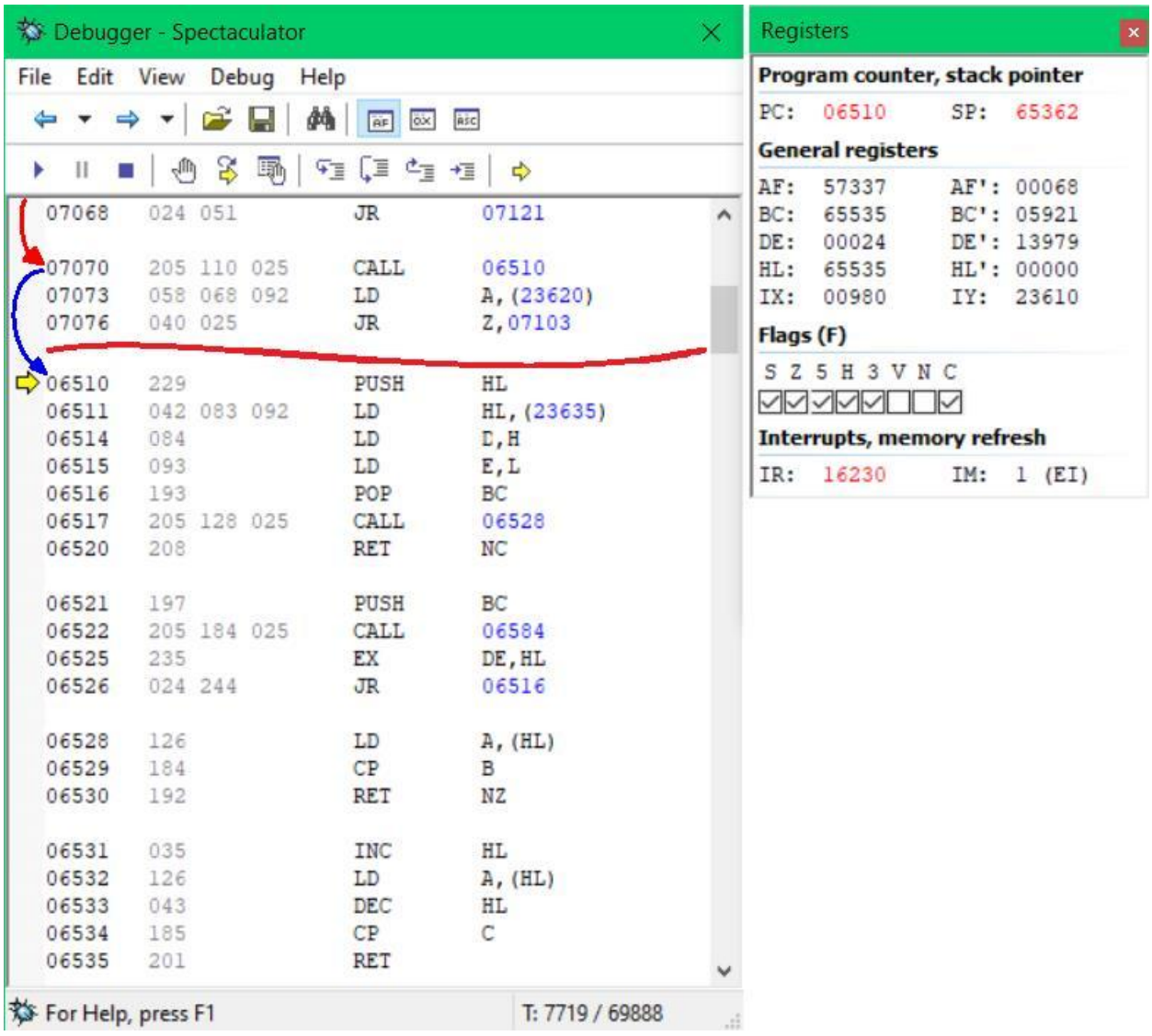


Рис. 203.

Из переменной PROG (23635) берётся адрес начала области BASIC и копируется в «DE». В «BC» перемещается номер строки и с этими входными параметрами Стрелочка бежит в расположенную по соседству программу CP LINES (6528).

Вынув номер строки из памяти по указанному адресу, она сравнивает его с эталонным значением из «BC». Если значения первого и второго байта строки совпали, значит, строка нашлась, и далее сравнивать не имеет смысла. Показав результат успешной

работы установкой Z ☑ галочки «Z», подпрограмма даёт команду на возврат из всей системы подпрограмм `LINE ADDR (6510)`, снятием C ☐ галочки из окошечка «C». В адресе 6520 происходит возврат по команде отсутствия этой галочки (`RET NC`).

Отсутствие результата, это тоже результат. Поняв, что номер строки меньше требуемого, происходит возврат на ступень выше в `LINE ADDR (6510)`. Не теряя оптимизма, программа не снимает C ☑ галочку «C», запрещая глобальный возврат в верхние уровни.

Вместо этого вызывается многоцелевая подпрограмма `NEXT ONE (6584)`, которая вычисляет адрес следующей строки или переменной. Это универсальный дешифратор всех возможных блоков данных в областях `PROG-VARS`:

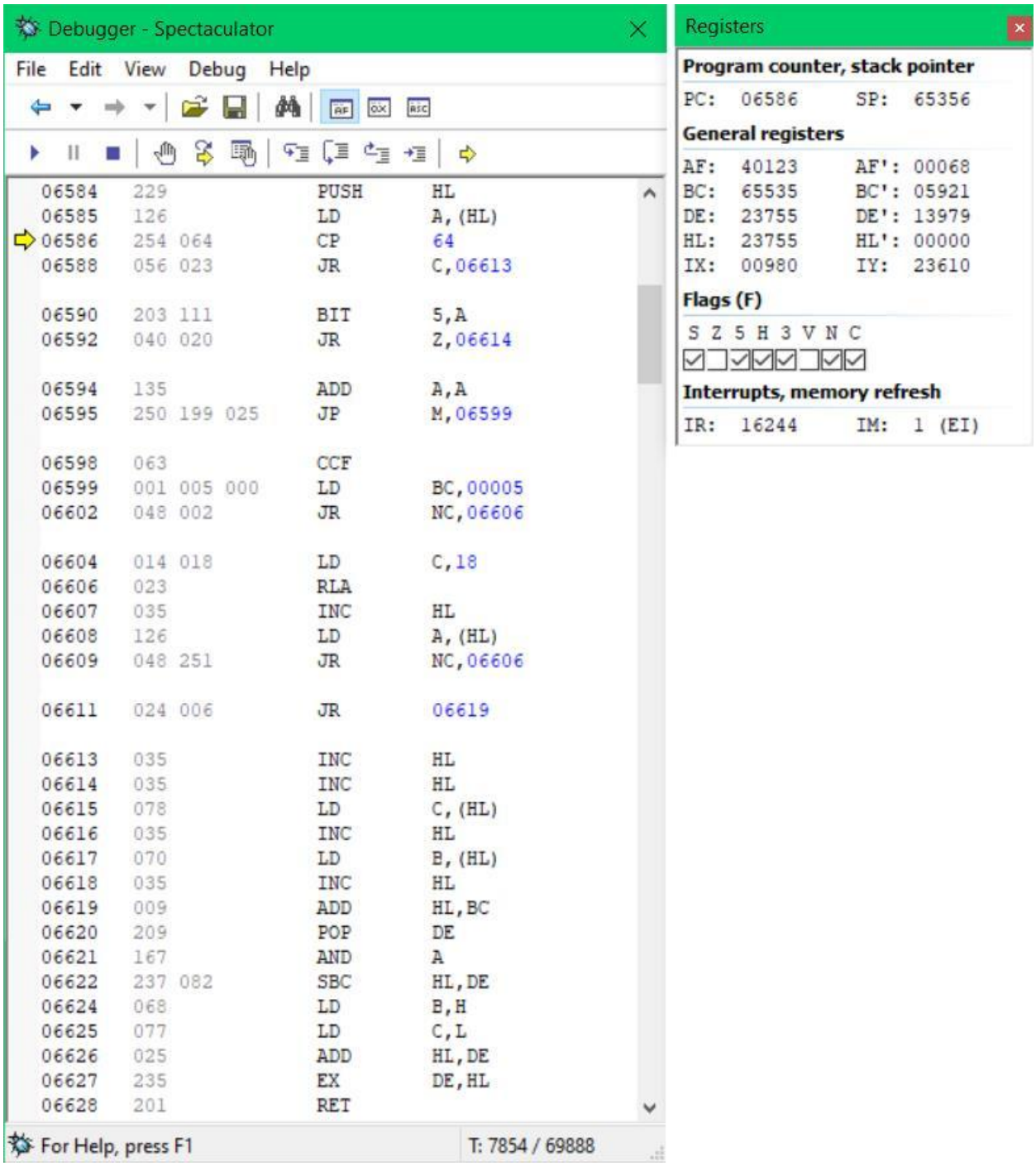


Рис. 204.

По прибытию в программу, первым делом определяется тип блока (программа или переменные), чтобы подобрать нужный алгоритм расшифровки для вычисления длины. Взяв начальный старший байт адреса строки, он проверяется на 64 – минимальное значение, которое может относиться к блоку переменных по задумке создателей BASIC. Если число меньше 64 (64*256=16384), значит это BASIC строка и можно смело переходить по адресу 6613 к алгоритму расшифровки, для вычисления длины.

Если первый байтом стоит число 64 и выше, то это обязательно должна быть переменная (ведь перед этим стоит несколько слоёв защиты, чтобы строки свыше 9999 было невозможно ввести). Проходя вниз, начинается выяснение, какому именно типу переменной требуется подобрать алгоритм.

Таким образом, вычисляется адрес следующего элемента (строка/переменные/массивы) в области `PROG-VARS`. В результате работы этой программы формируется адрес следующей строки или блока переменной и перед выходом записывается в «DE». Вернувшись в подпрограмму `LINE ADDR (6510)` ступенькой выше, Стрелочка с новыми значениями заходит в `CP LINES (6528)`. Предложив свежий вариант адреса, программа снова извлекает номер строки и сравнивает с эталоном требуемой строки.

Цикл будет продолжаться снова и снова, пока номер строки, считанный из памяти, не совпадёт с эталоном из «BC» или не кончатся строки с переменными. Уперевшись в маркер «128» происходит заключительное сравнение с эталонным номером. А маркер по расчетам, всегда должен быть больше старшего байта любого номера строки (128*256=32768), поэтому в результате сравнения, блокирующая галочка «C» снимется и произойдёт выход в `LINE NEW (7070)` с пустым результатом.

Таким образом, независимо от результатов сравнения произойдёт выход из подпрограмм `LINE ADDR (6510)`. В случае провала на выходе в «HL» останется адрес строки/переменной или маркера, с большим значением, чем нужная строка. А при успешном результате появится адрес строки для выполнения, дополнительно сопровождённый подтверждающей Z ☑ галочкой «Z»:

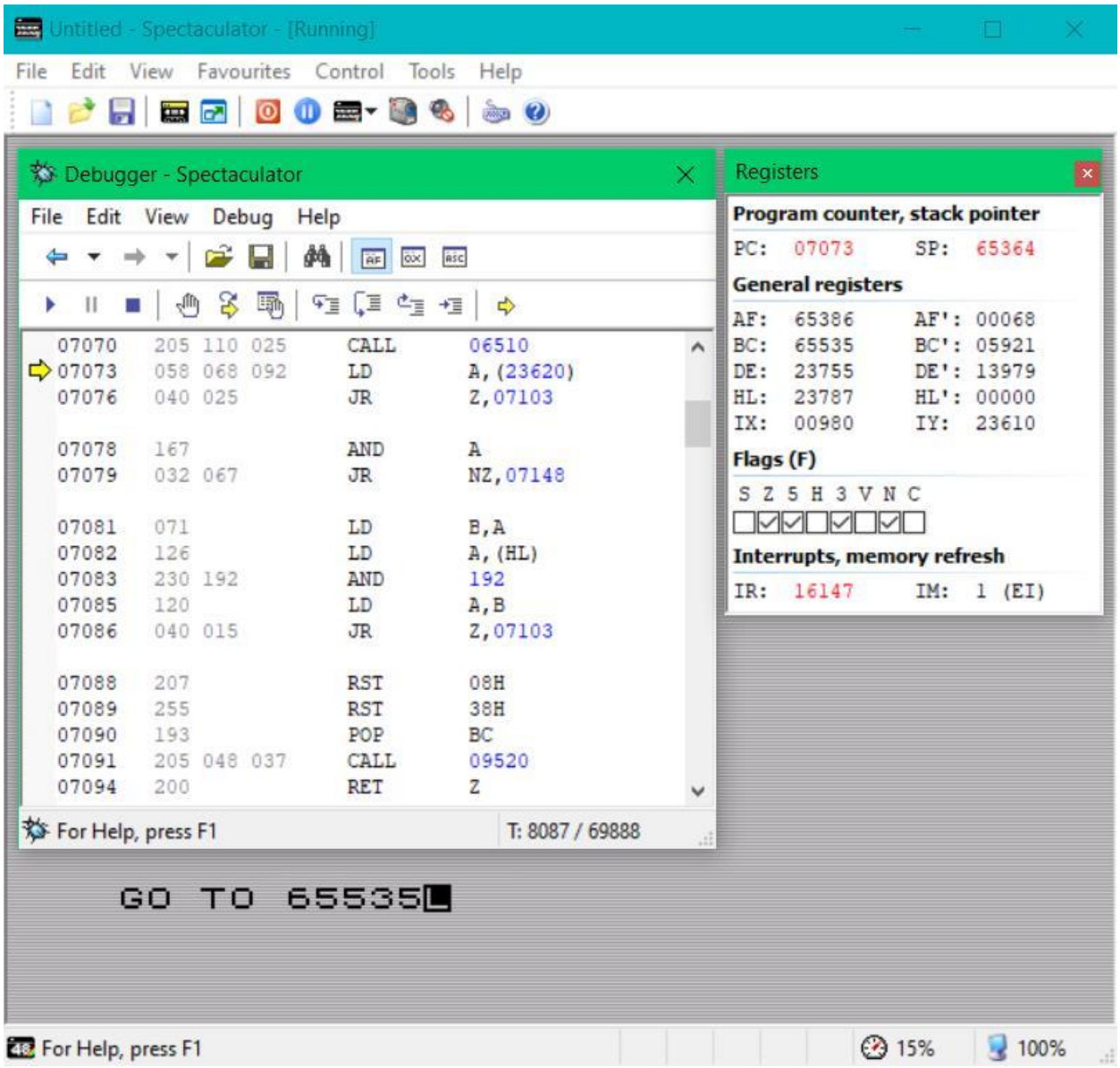


Рис. 205. Возврат с подарками.

Далее из переменной `NSPPC (23620)` берётся номер оператора, который требуется выполнить и проверяется успешность поиска. Если строка найдена, то происходит переход для её выполнения.

Если не найдена, то есть вариант, что запуск произошёл с помощью пустого `RUN` или `GO TO` с отсутствующим номером строки, но меньшим, чем имеется в BASIC программе. Поэтому делается еще одна проверка и даётся последний шанс на выполнение. Ну а если и дополнительная проверка не дала результатов, то можно заканчивать выполнение, выходить в основной цикл и печатать сообщение.

Глава 19
Способы выполнения невыполняемых строк

Краткое содержание: запуск строки 65535

Итак, из прошлой главы вы убедились, что выполнить невыполнимое, как и впихнуть невпихуемое, всё-таки можно. Предлагаю запустить «невидимую невыполнимую» строку с номером 65535. Для её выполнения нужно будет перебросить Стрелочку через три барьера. Продолжу экспериментировать на программе из прошлой ознакомительной главы:

```
Debugger
Dec
Add Breakpoints 7793, 7048, 6588
Trace
BASIC ← GO TO 65535 ENTER
AF ← AF+1
Trace
AF ← AF+64
Trace
AF ← AF+1
Remove Breakpoints 7793, 7048, 6588
Trace
```

Поскольку появляется новая команда языка «*God Mode*», процесс запуска рассмотрю пошагово с рисунками. Повторение не будет лишним.

Откройте *Debugger* (команда «*Debugger*»). Под новой командой «*Add Breakpoints 7793, 7048, 6588*» подразумевается очередная удобная функция отладчика, которую я предлагаю освоить. Она позволяет дистанционно поставить целый пакет малиновых точек прерывания для отлова Стрелочки.

Найдите на панели кнопку «Ручка трогает программу» (*Breakpoints, Ctrl+B*):

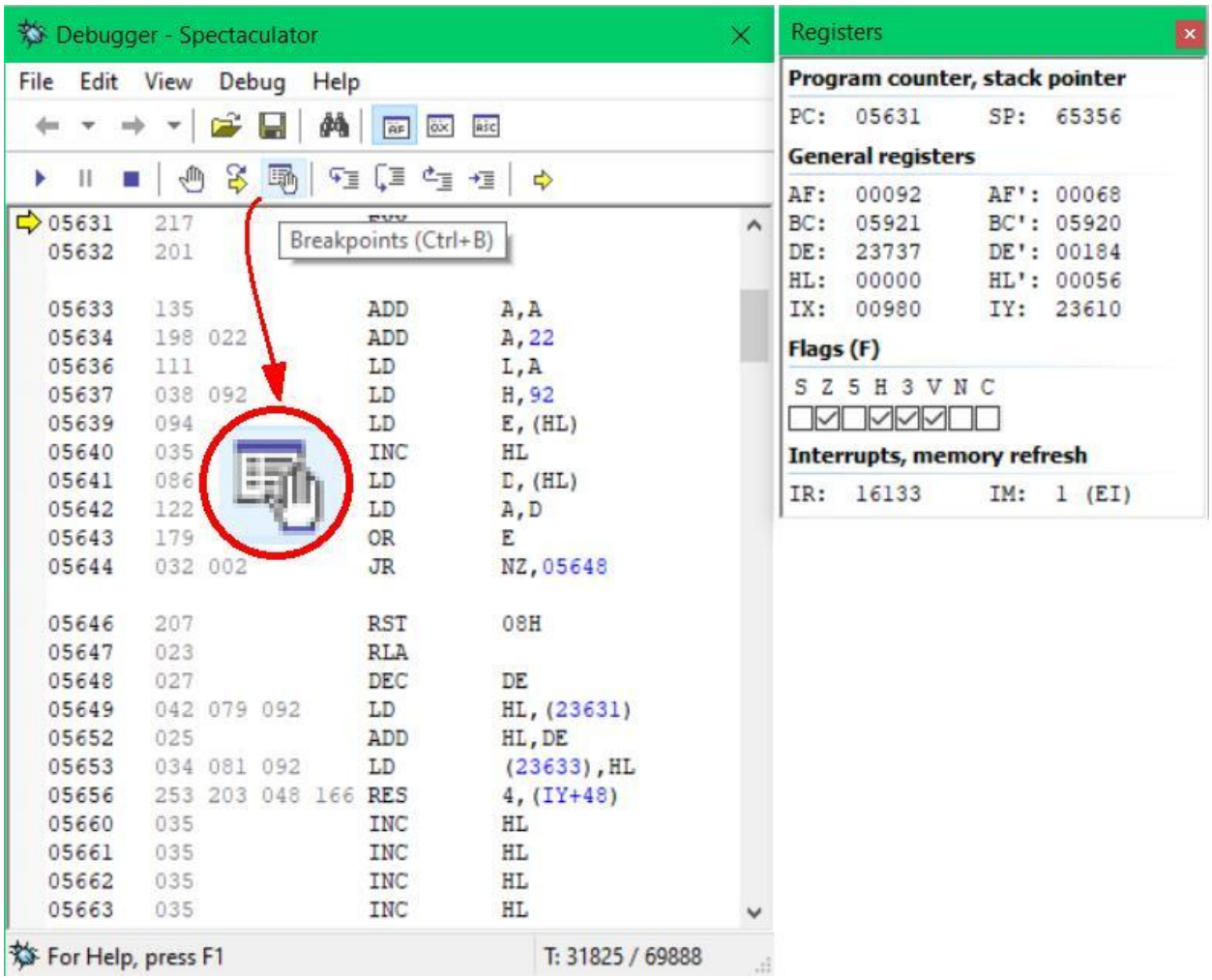


Рис. 206. Кнопка «Breakpoints» для оптовой установки малиновых точек прерывания.

Нажав на нее, откроется дополнительное окошко «*Breakpoints*», а в полоске «*Break At:*» замигает курсор:

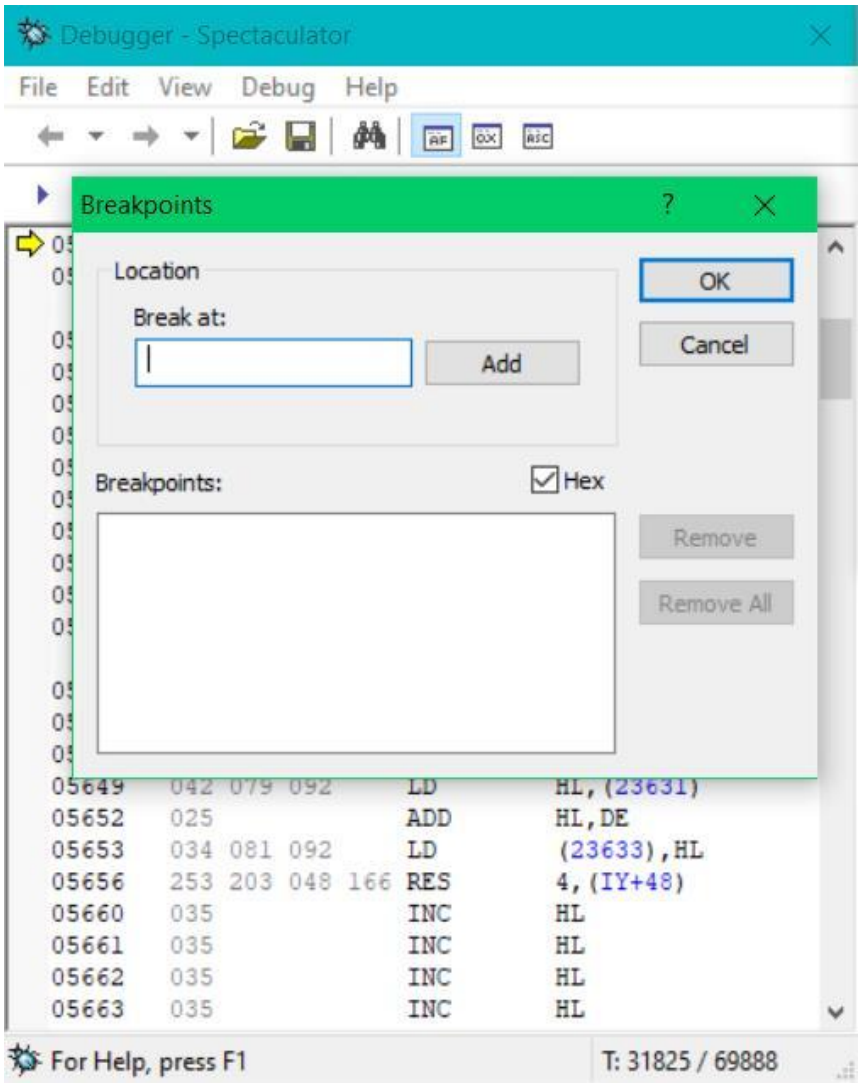


Рис. 207. Открытие окошка оптовой установки точек прерывания «Breakpoints».

Наберите в белое окошечко число 7793 для первой точки ● остановки:

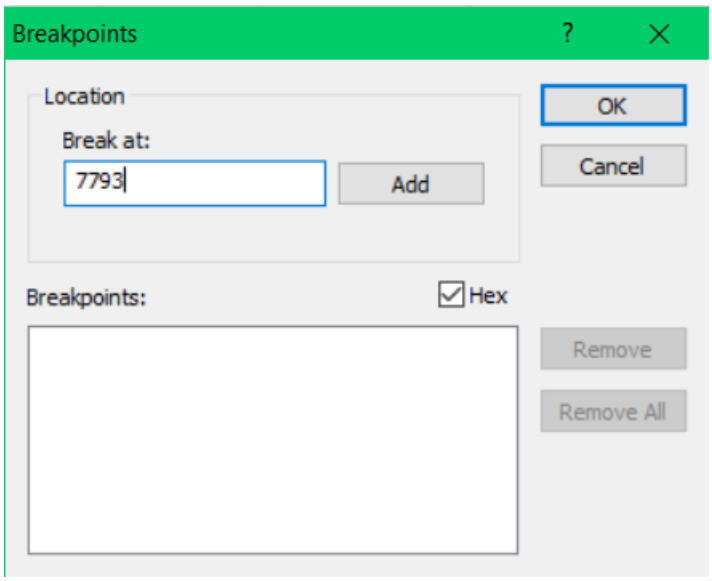


Рис. 208. Процесс ввода адреса точки прерывания в полоску «Break at:».

Нажмите кнопку «*Add*» слева от окошка, и число переместится в большое белое пространство (живот) «*Breakpoints*». Малиновая точка по вашему желанию поставлена дистанционно. Однако число в большом нижнем окне отобразилось в HEX-формате:

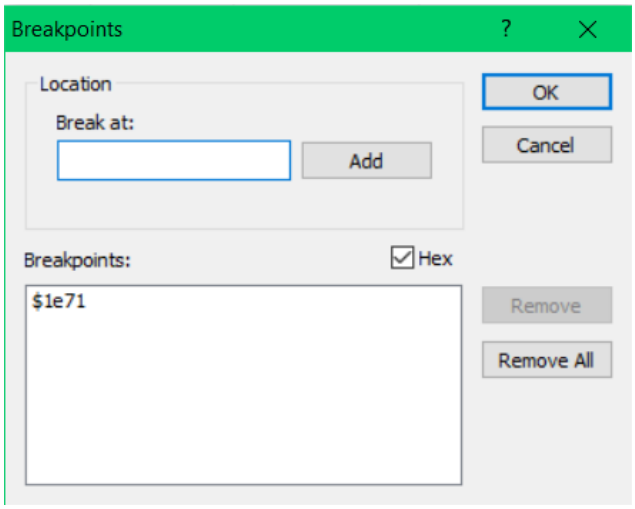


Рис. 209. Отображение введённой точки прерывания по умолчанию в шестнадцатеричном формате.

Снимите галочку в окошечке «Hex», для чего установите внутрь курсор мыши и нажмите левую кнопку:

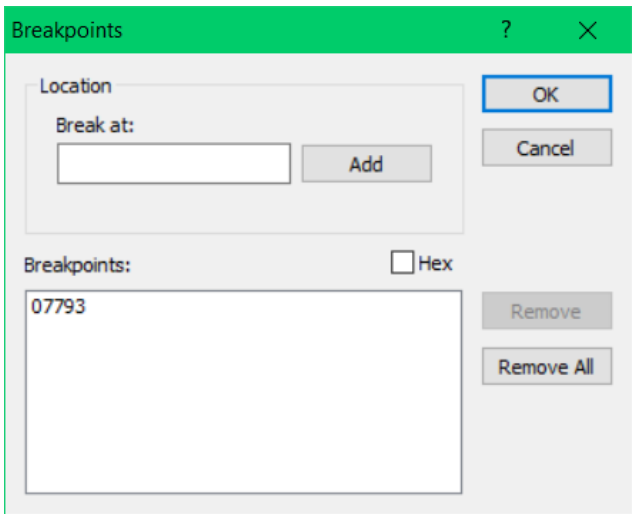


Рис. 210. Отображение введённой точки прерывания в десятичном формате после снятия галочки.

Число преобразилось в привычный вид. Снова установите курсор в тонкое окошко «*Break at:*». Аналогичным образом наберите и введите число 7048, и последним 6588. Получится вот такой список адресов, на которых разместятся ●●● малиновые точки:

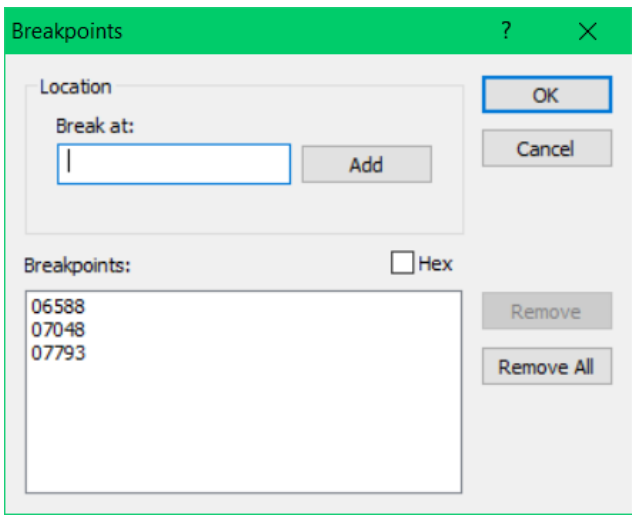


Рис. 211. Список малиновых точек в окошке Breakpoints.

Нажмите кнопку «*OK*». Окно «*Breakpoints*» исчезнет. Следом нажмите «*Trace (F5)*», чтобы закрылось окно отладчика.

Натуральным способом с клавиатуры наберите `GO TO 65535`:

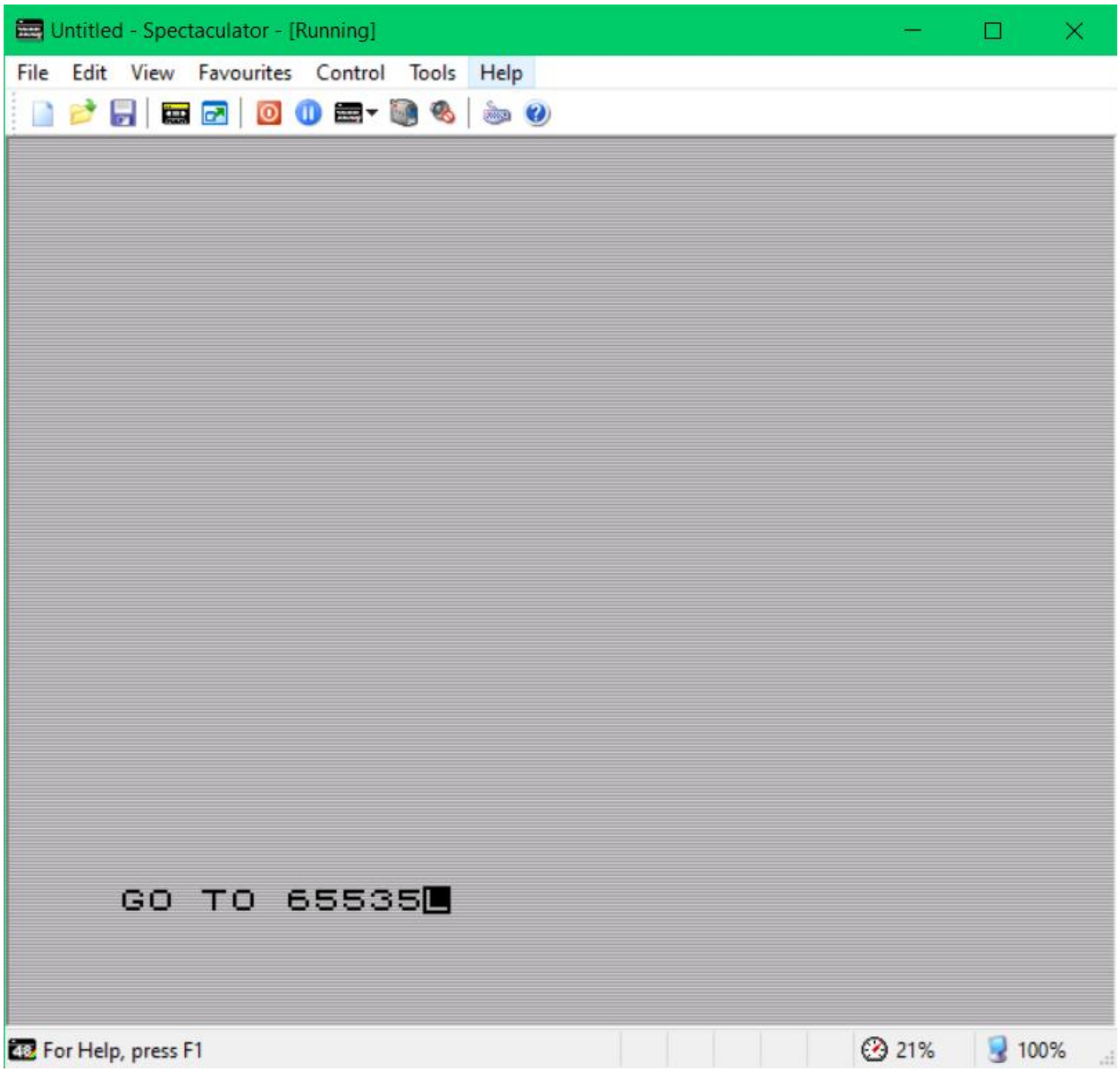


Рис. 212. Набор команды GO TO 65535 натуральным способом.

Вводите строку. Сразу откроется окошечко «*Debugger*», в котором Стрелочка встанет в точку напротив адреса 7793. Мышкой установите галочку в квадратик C под буквой «C» или прибавьте к содержимому «AF» единицу:

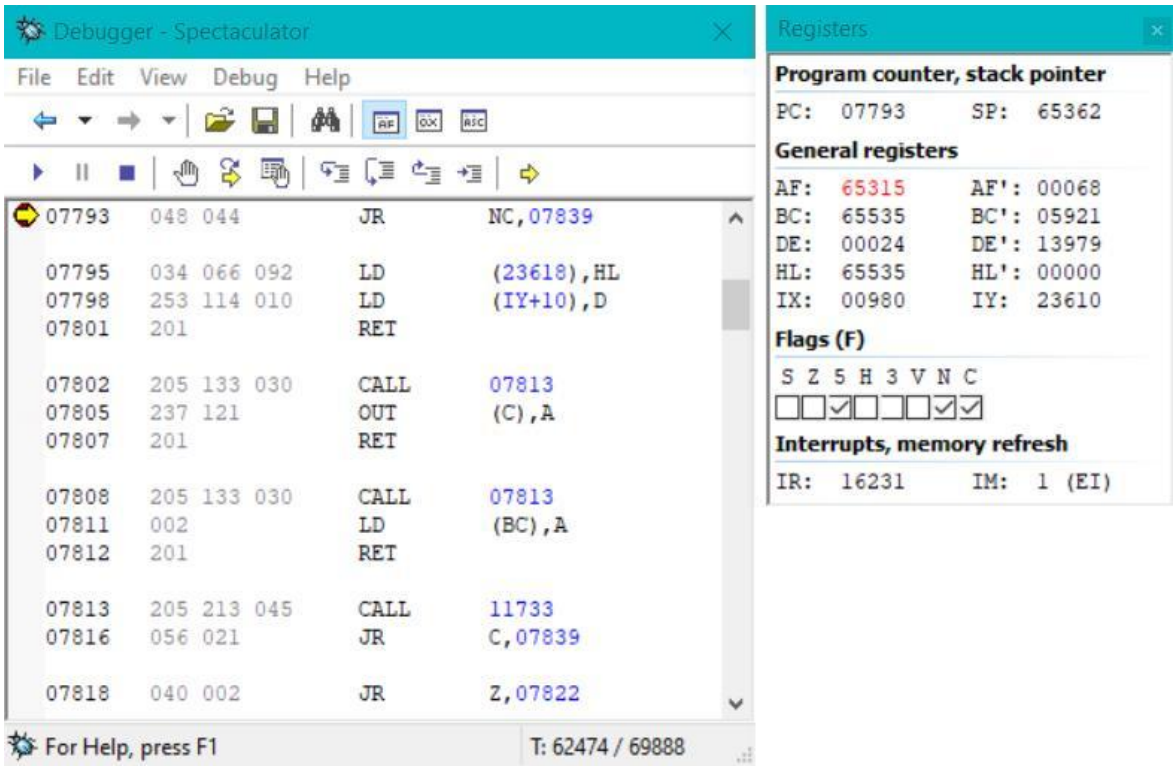


Рис. 213.

Галочка установилась в клетку, а значение «AF» покраснело и увеличилось на единицу. Нажимайте «*Trace (F5)*», и защита с сообщением «`B Integer out of range`» успешно преодолена.

Окошко мигнуло и вот уже Стрелочка стоит на второй точке, где нужно обойти ограничение на выдачу сообщения «`N Statement lost`». На этой остановке установите галочку Z ☑ в окошко под буквой «Z» или прибавьте к имеющемуся в «AF» число «64», чтобы Стрелочка перескочила на адрес 7070:

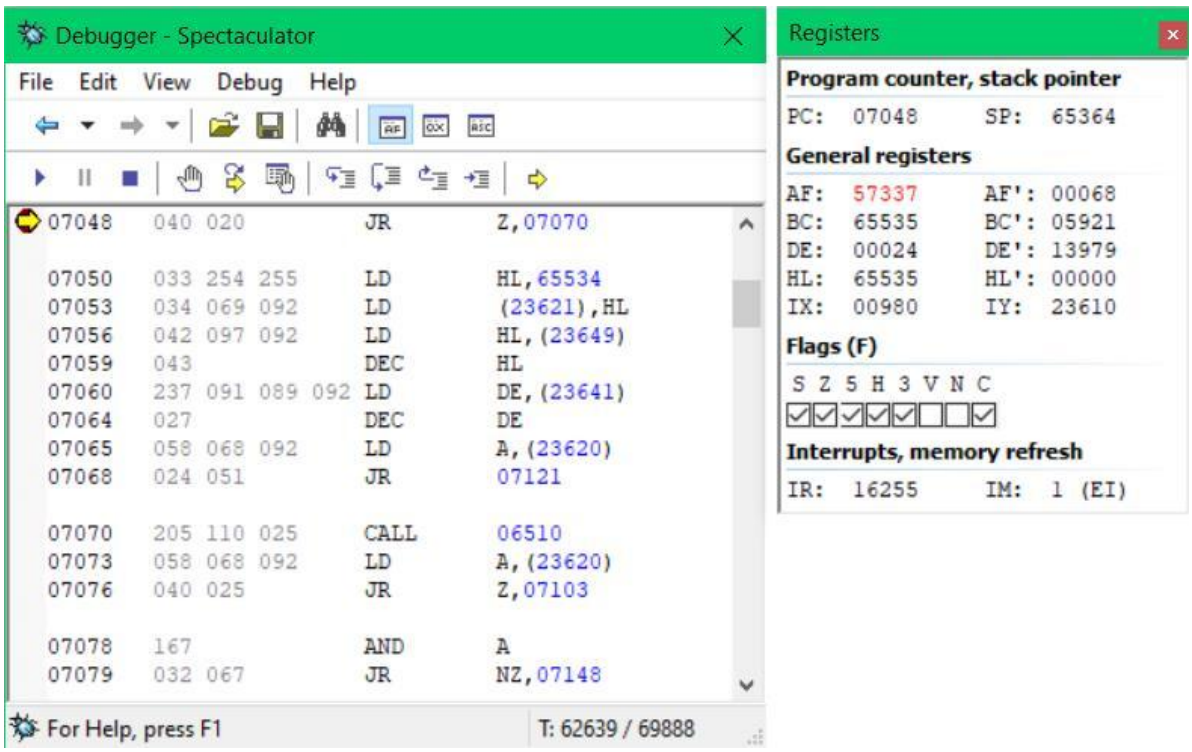


Рис. 214.

Нажимайте «*Trace (F5)*», и окошко откроется перед последней преградой на пути к успеху по адресу 6588. Снова установите C ☑ галочку «C» или прибавьте к имеющемуся в «AF» единицу:

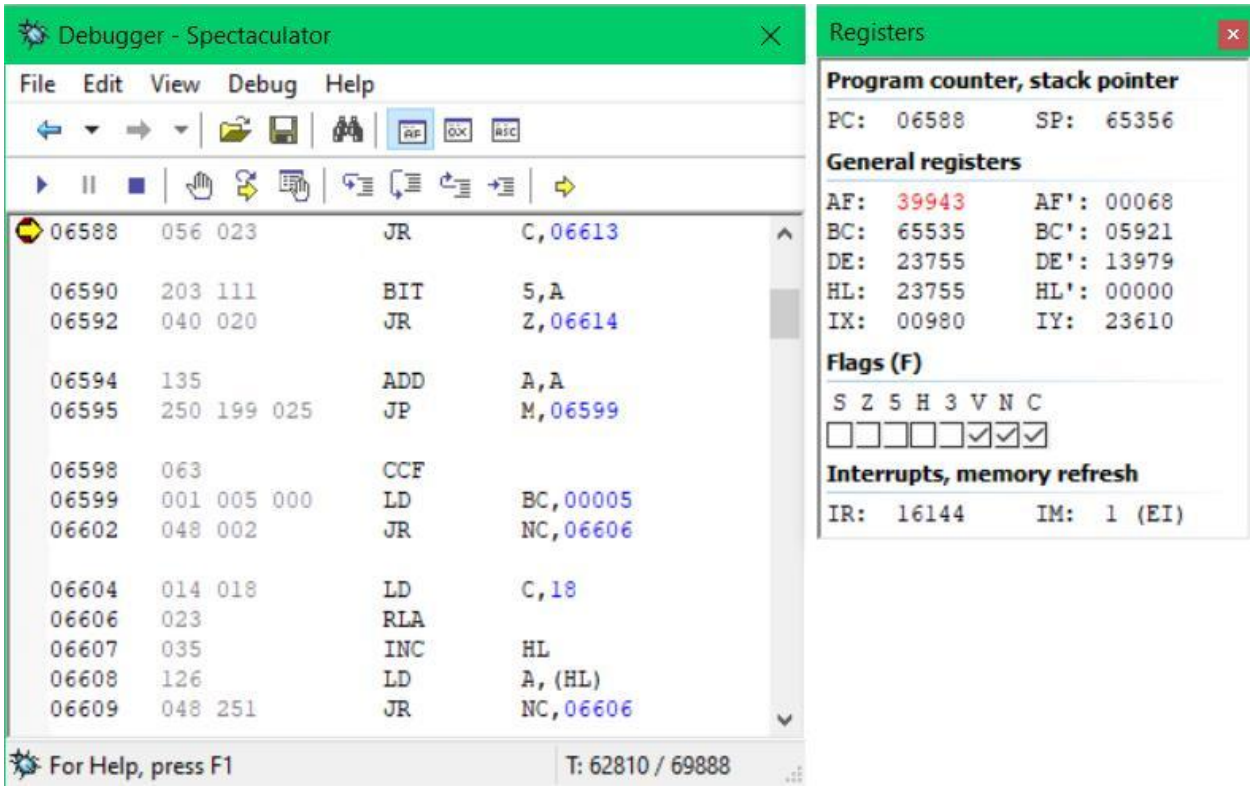


Рис. 215.

Последняя и самая жестяная преграда позади.

Именно на этом шаге часто происходит ошибка подбора алгоритма для вычисления размера строки. Проход дальше грозит тем, что для вычисления текущей строки 65535 будет взят микс из младшего байта номера строки и первого, тоже младшего байта, длины строки. Чем это закончится, вы наверняка догадались, вспомнив схематичный рисунок сравнения структуры невидимой строки и однобуквенной символьной переменной. Дальше по нарастающей пойдёт «придумывание» блоков с ошибочными длинами и номерами строк. Это приведёт к глухому зависанию и Стрелочка навсегда останется в системе подпрограмм `LINE ADR`, больше никогда не выбравшись на поверхность по команде «`RET NC`». Поможет в этом случае только принудительное перемещение Стрелочки или кнопка «*Reset*». В штатном режиме распаковать строку по алгоритму переменной (без одного байта) не дают барьеры, которые в настоящий момент ликвидированы.

Ну а теперь осталось выполнить новую команду «*Remove Breakpoints 7793, 7048, 6588*», которая оптом удалит все малиновые ● точки из памяти.

Снова откройте окошко «*Breakpoints (Ctrl+B)*»:

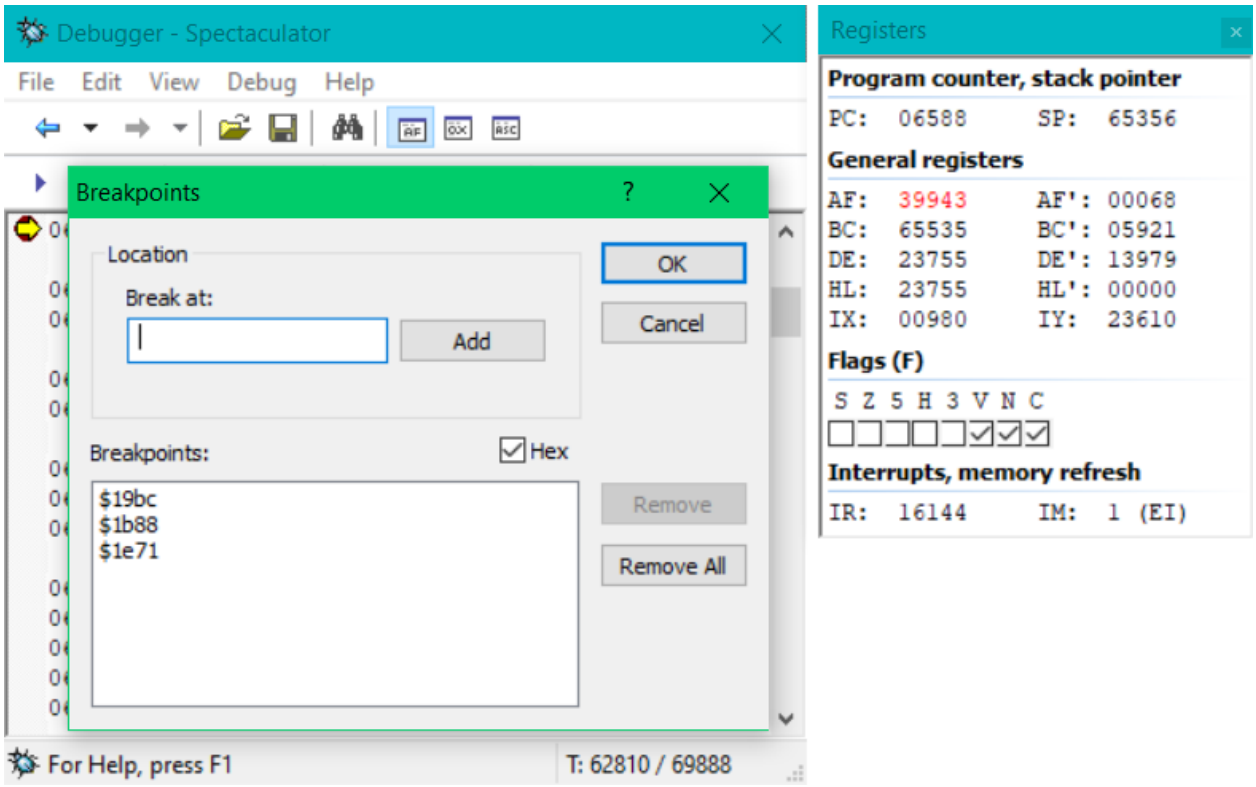


Рис. 216.

Нажмите кнопку «*Remove All*» и список адресов исчезнет:

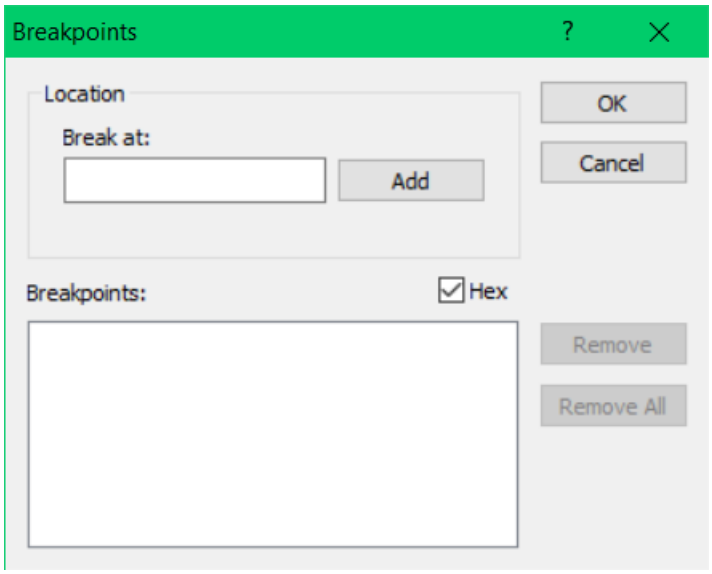


Рис. 217.

Теперь нажмите кнопку «*OK*», чтобы точки стёрлись из памяти, а следом закрылось и само окошко:

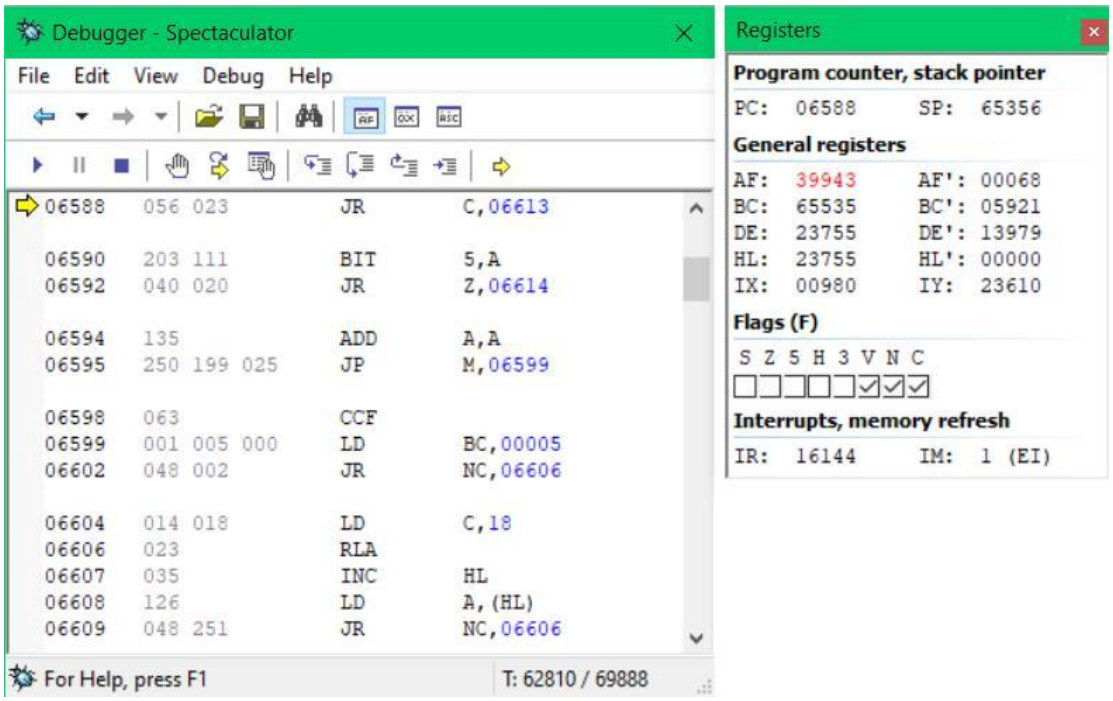


Рис. 218.

Настаёт торжественный момент. Нажимайте «*Trace (F5)*» и:

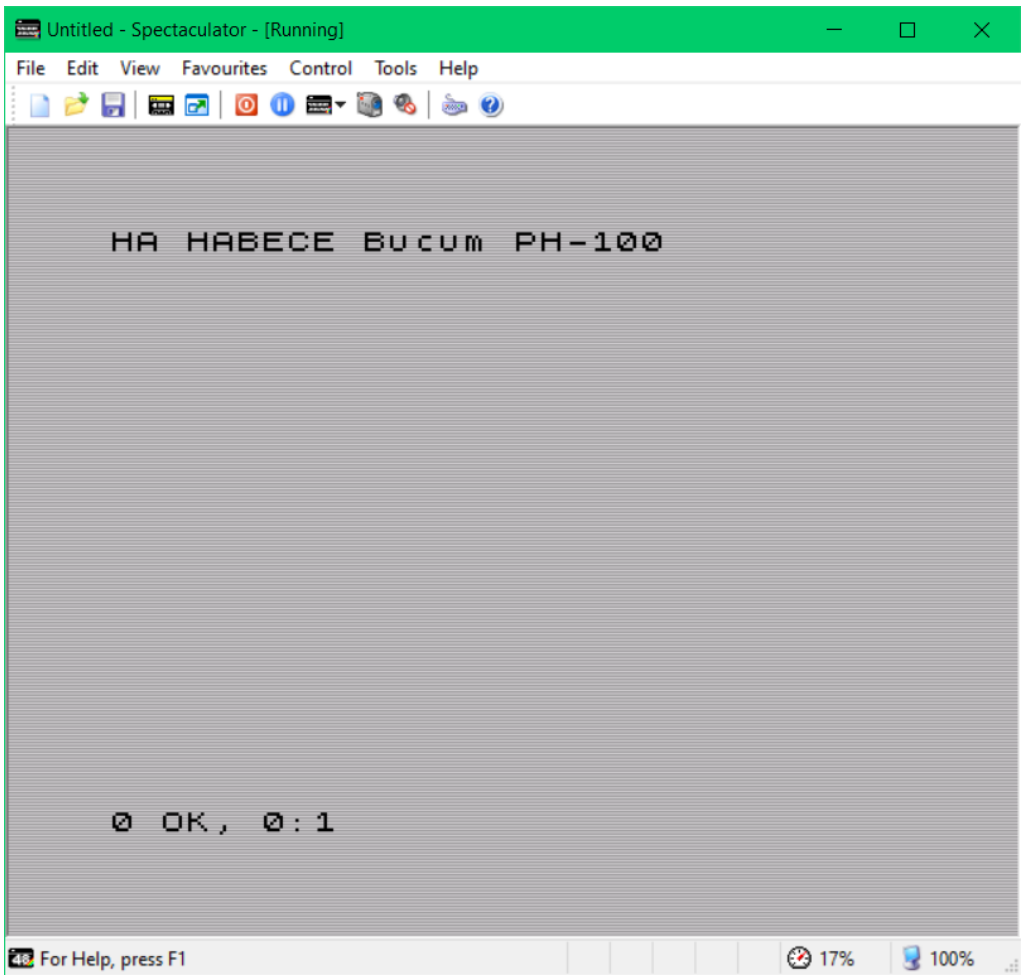


Рис. 219. Результат выполнения строки 65535 после обхода всех защит.

Невероятно! Строка 65535 выполнилась с сообщением «0 OK», не сбив систему. Остался открытым самый главный вопрос: что же такое РН-100?

Рис. 220. Пивной навес со светильниками РН-100. Альпийский переулок 30. Фото 8 марта 2006 года.

…За серым кирпичным девятиэтажным домом показалась стена спортзала, являющаяся частью торгово-бытового комплекса серии 2ЛГ-07-1. Миновав его и вход в 40-е отделение милиции, мы вышли к боковой стороне магазина.

Между дорогой и стеной магазина располагался Г-образный навес, стоящий на черных металлических колоннах. У стены магазина, лицом к навесу, стоял пивной ларёк. Около него уже скопилась небольшая очередь. Дедушка быстро подошёл к ларьку и занял очередь.

Стояли мы недолго. Вскоре из окна дедушке протянули вожделенную кружку с желтоватой жидкостью, покрытой сверху небольшим слоем белой пены.

Дедушка отошел на свободное пространство под крышу.

– Подержи, я лимонад тебе открою – сказал дедушка, аккуратно подав кружку с пивом.

Двумя руками я осторожно взял тяжёлую кружку. Дедушка достал бутылку с «Пепси-Колой» и открыл пробку об какую-то торчащую железку. Забрав обратно свою кружку, он вручил мне бутылку с лимонадом.

Всюду кучковались компании мужиков, равномерно распределившись под навесом. Они также как и дедушка, пили пиво из кружек. Со всех сторон слышались разговоры с матюжками, разбавляемые редкими раскатами гогота. Голоса сливались и превращались в невообразимую мелодию.

Я стоял, пил «Пепси» и смотрел по сторонам на пустырь с недавно построенным исполкомом и парк за проспектом Славы, в конце которого виднелись трубы кирпичного завода.

Дул приятный тёплый ветерок, добавляя весенних ароматов к запаху подкисшего пива или мочи с напольных плит.

Я посмотрел наверх. Высоко на потолке висели продолговатые матовые плафоны с тонкой металлической решеткой. Светильники располагались в два ряда по всей длине навеса…

(Прогулка на сортировочную. Из воспоминаний о весне 1987 года)

Вот такой предмет висит на навесе. Это популярный рудничный светильник конца 1960-х. Светильники РН-60 и РН-100 вешали на сараи, склады и под навесы. Эта модель также засветилась в некоторых популярных советских фильмах. В Ленинграде, такими светильниками оснащали навесы купчинских торговых центров серии 2ЛГ-07-1. Но больше всего в народе он прославился как «пивной светильник», потому что частенько встречался на пивных ларьках. В современное время светильниками с решетками часто любят оформлять пивные рестораны в стиле «Лофт».

Ну а теперь запустите строку 40000. Не обновляя экран, введите следующую алгоритмическую программу:

```
Debugger
Dec
Go To 7048
Add Breakpoint 7048
Trace
BASIC ← GO TO 40000 ENTER
Remove Breakpoint 7048
AF ← AF+64
Trace
```

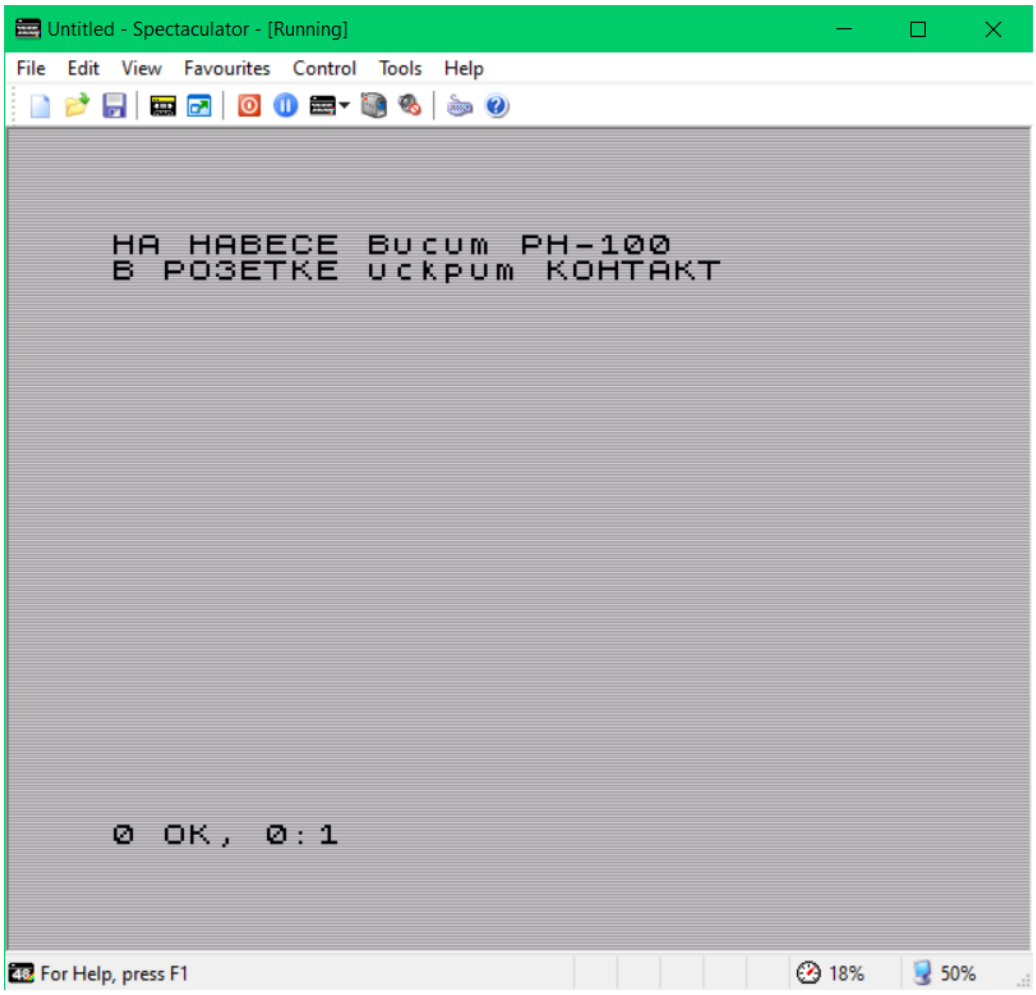


Рис. 221. Результат выполнения строки 40000 после обхода защит.

Обратите внимание, насколько легче запустить «невидимую незапускаемую по Statement lost», чем «невидимую незаапускаемую по Integer out of range».

На этом со строками пока предлагаю закончить. Кого заинтересовало, не переживайте. Эту тема не раз будет затрагиваться в будущем.

Глава 20
RETURN без GO SUB как Попорук без Рукопопа

Краткое содержание: GO SUB-заряд, RETURN-запуск

Часто в играх и даже в BASIC подпрограммах можно встретить такой тип перехода, когда после какой-либо работы, командой `PUSH` сохраняется значение, а следом по `RET` происходит возврат на сформированный адрес. За примером далеко идти не надо. Достаточно взглянуть на фрагмент подпрограммы `JUMP-C-R (7190)`:

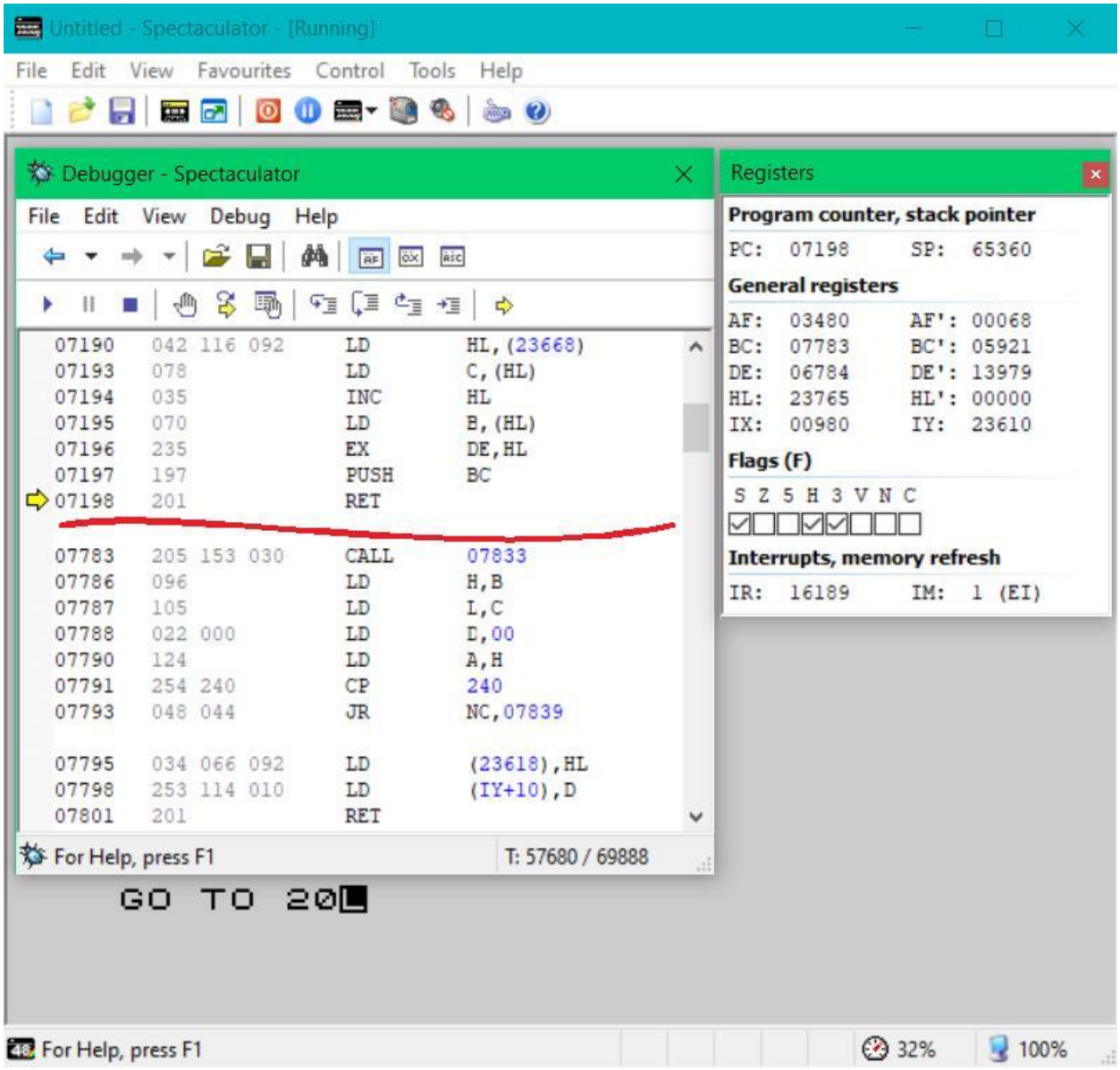


Рис. 222. Переход по методу PUSH-RET.

В данном случае вычисляется адрес набора действий для команды `GO TO` и в 7198 происходит переход на соответствующую подпрограмму.

В текстовых BASIC программах для вызова отдельных строк с последующим возвратом, существует команды GO SUB (*жарг. ГО СУБ, Голубиный Суп, и пр.*) и RETURN. Очевидно, что без участия GO SUB, задав где-то искусственно адрес возврата, можно по RETURN попробовать запустить программу с определённой строки и оператора. Таким образом, возврат из подпрограммы получится в строку, из которой она не вызывалась.

Давайте посмотрим на саму подпрограмму, которая формирует команду GO SUB:

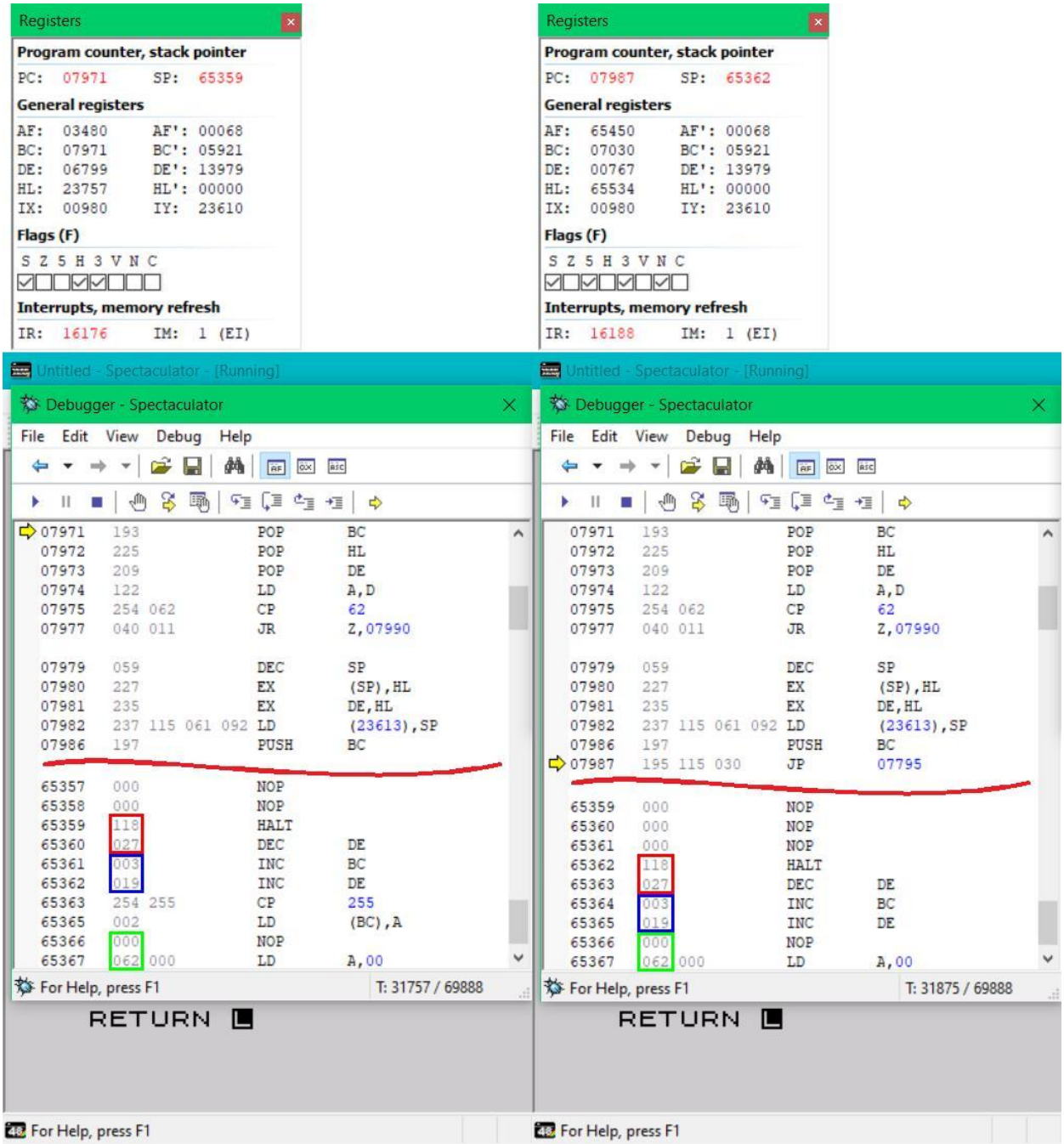


Рис. 223. Схема работы команды GO SUB.

На момент перехода к подпрограммам действий для BASIC команд, столбик «SP» состоит из 6 значений или трёх пар: фундамент (0, 62), выход в главный цикл после успешного выполнения программы (3, 19 = 4867) и адрес возврата для сканирования «BREAK» с печатью сообщения об ошибках (118, 27 =7030).

В 7917 начинается выполнение программы действий команды GO SUB. В «DE» запоминается текущее значение из башенки «SP» с подпрограммой выполнения BASIC

строк STMT-RET (7030). Из SUBPPC (23623) берётся номер текущего выполняемого оператора. Для будущего возврата по команде RETURN, его значение увеличивается на единицу и ставится… в SP-столбик на воздушную прослойку над фундаментом с маркером «62». Следом из PPC (23621) считывается номер BASIC строки с выполняемым оператором и ставится сверху. Этажом выше возвращается на место значение возврата в главный цикл, которое стояло на основании до этой реконструкции. Над этим значением вставляется адрес возврата для указателя ERR_SP (23613).

Таким образом, при выполнении команды GO SUB номер оператора и строки для возврата по RETURN просто подсовывается под столбик значений регистра «SP» и устанавливается на самое основание.

Ну а RETURN просто считает отсюда данные и запустит программу и вытащит свои три значения, опустив текущую башенку на три этажа вниз:

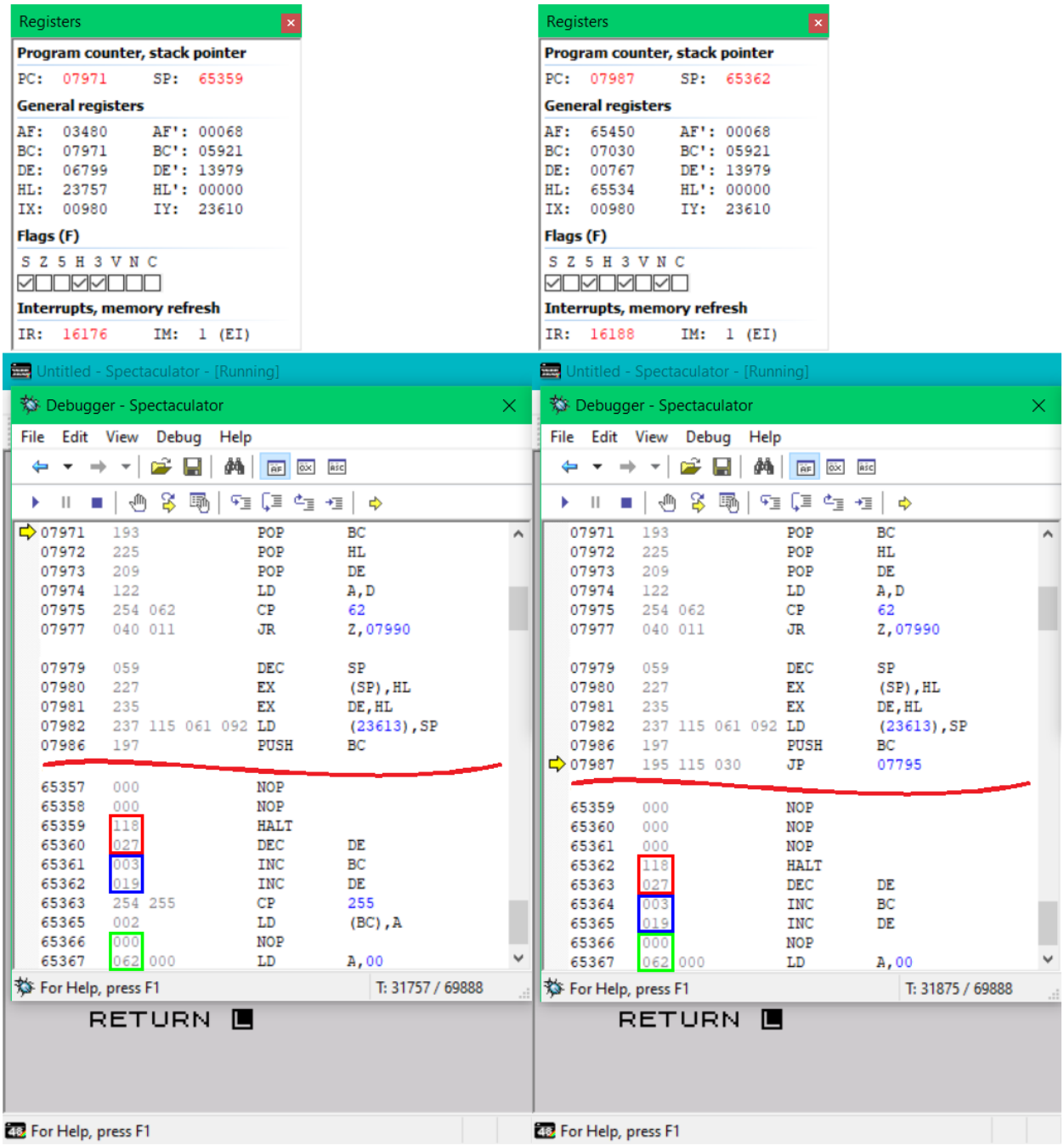


Рис. 224. Схема работы команды RETURN.

К моменту выполнения RETURN, столбик «SP» также имеет четыре рабочих этажа и фундамент. Поэтому Стрелочка спускается вниз на 6-й этаж. Смотрится значение

6-ю этажами ниже. Если это «62», значит, уперлись в фундамент и никаких данных от GO SUB тут не лежит. Происходит выход из программы с сообщением об ошибке «7 RETURN without GO SUB».

А если там какие-то числа, то снимаются все данные, столбик «SP» ставится на место, а в переменные NEWPPC (23618) и NSPPC (23620) записываются номер строки с оператором и далее происходит стандартное выполнение строки по методу RUN/GO TO.

По сравнению с прошлыми главами, GO SUB не кажется чем-то сложным.

На чистый *Spectaculator* введите следующую программу:

```
Debugger
Dec
Go To 23613
23613 ← 65361
23627 ← 23781
23641 ← 23782
23649 ← 23784 23784 23784
23755 ← 0 10 22 0 245 34 110 111 110 111 112 121
23767 ← 107 34 58 245 34 80 121 107 111 110 111
23778 ← 110 34 13 128 13 128
65363 ← 10 0 2
SP ← 65363
PC ← 4770
Trace
```

После ввода экран обновится и появится следующая строка:

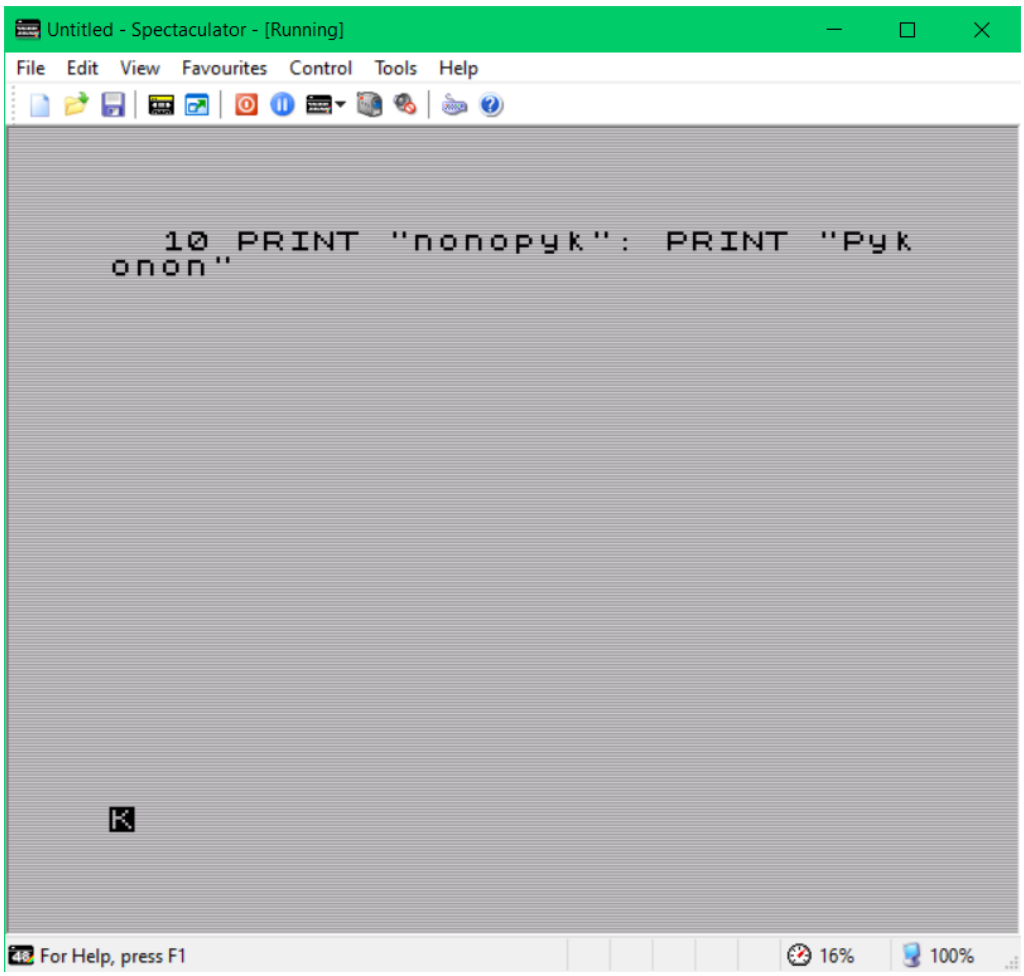


Рис. 225.

Осталось натуральным способом набрать RETURN:

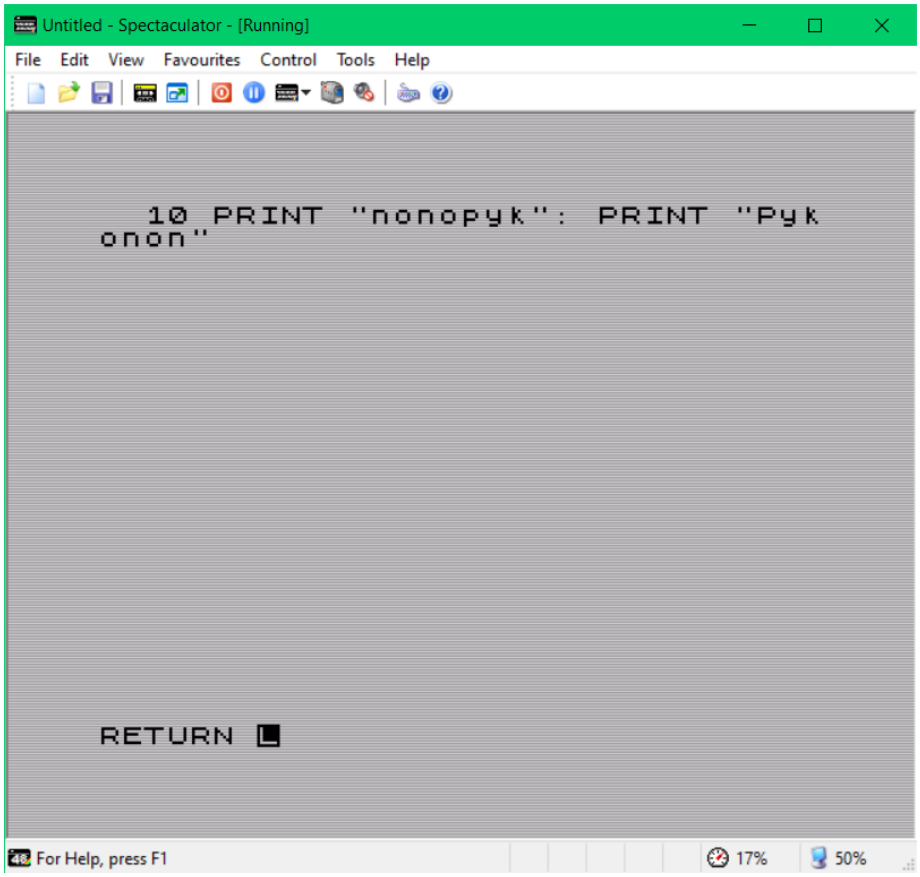


Рис. 226. Ввод RETURN натуральным способом для вывода текста.

Вводите и:

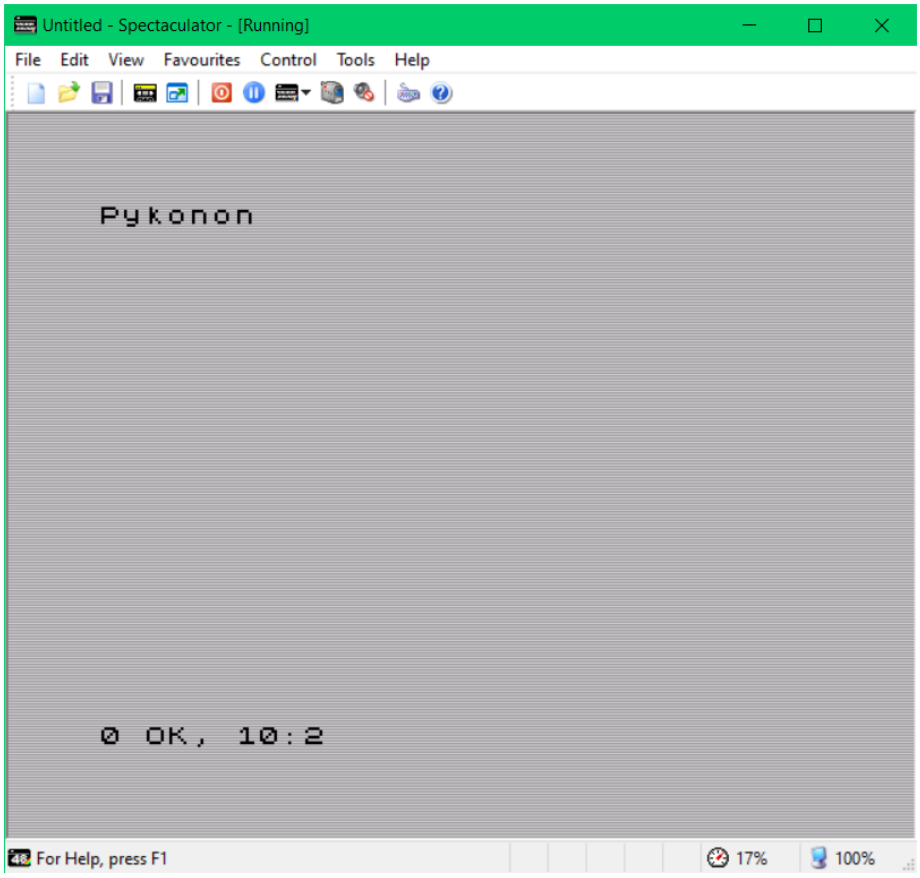


Рис. 227.

Рукопоп собственной персоной к вашим услугам. Всё отлично, кроме одного момента: Рукопоп без Попорука, это как Зита без Гиты.

Наберите и введите `RETURN` ещё раз:

Рис. 228. Реакция на RETURN при отсутствии «»GO SUB заряда.

Искусственно синтезированный `GO SUB 10/2` как и ожидалось, оказался одноразовым. Выработав предыдущим РЕТУРНОМ, он исчез.

На перспективу вы можете зарядить несколько разных `ГО СУБ`'ов. Потом набрать программу и потихоньку их расходовать. Только учтите, при заряженных `GO SUB`'ах нельзя пользоваться командой `RUN`. Она ликвидирует все адреса возврата и ставит столбик «SP» на место.

Введите и наберите программу, которая зарядит «SP» сразу двумя адресами возврата. Первый это эквивалент `GO SUB 50/2`, а второй `GO SUB 20`:

```
Debugger
Dec
Go To 23613
23613 ← 65358
65360 ← 50 0 2 20 0 1
SP ← 65360
PC ← 4770
Trace
```

После ввода, когда замигает курсор, натуральным способом введите следующую BASIC программу:

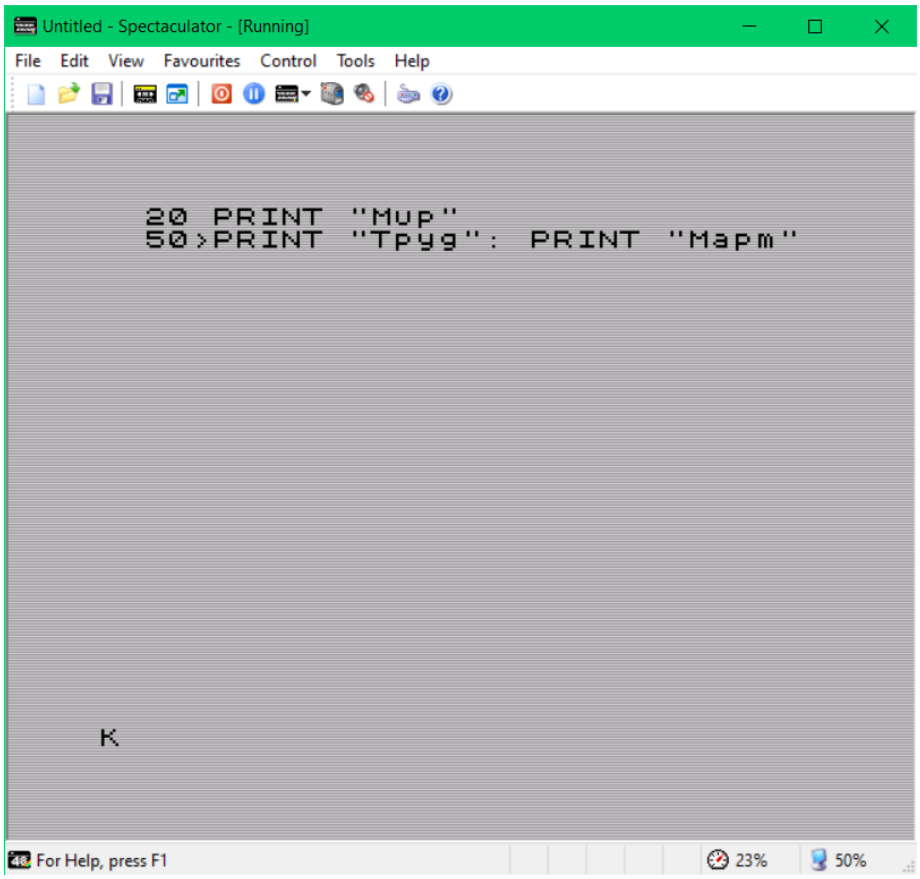


Рис. 229.

Вы даже можете проверить её работоспособность командой `GO TO 1`. Еще раз напомню: ни в коем случае не вводите `RUN`! Введите первый раз `RETURN`:

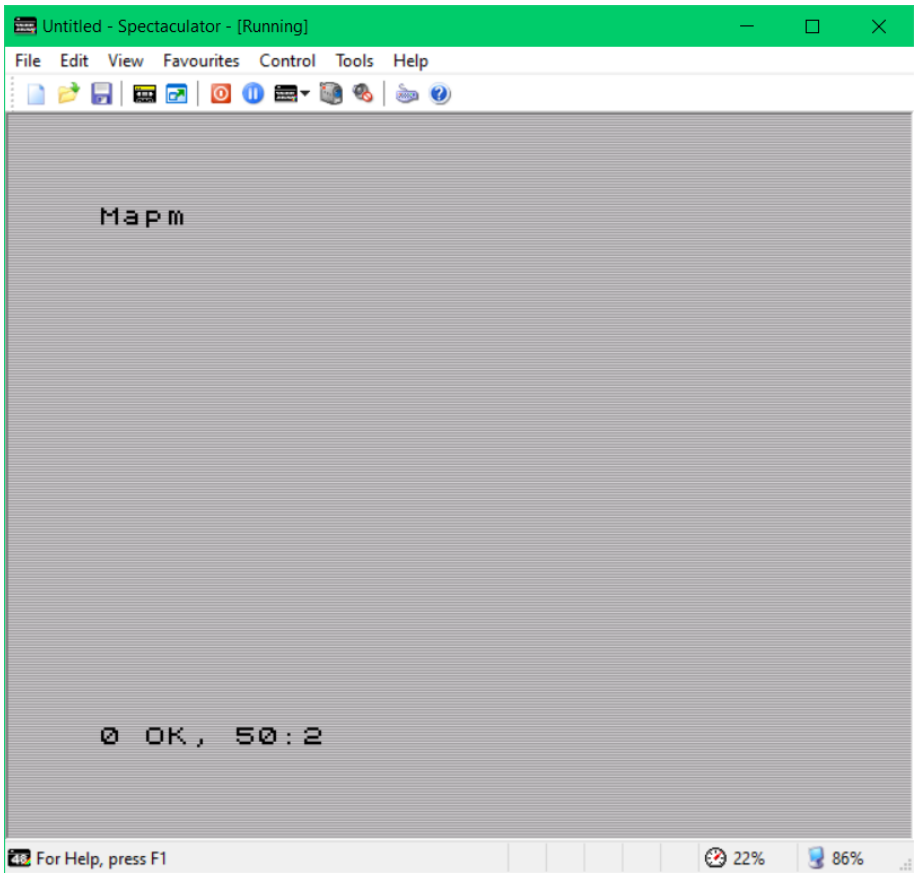


Рис. 230.

Введите второй раз RETURN:

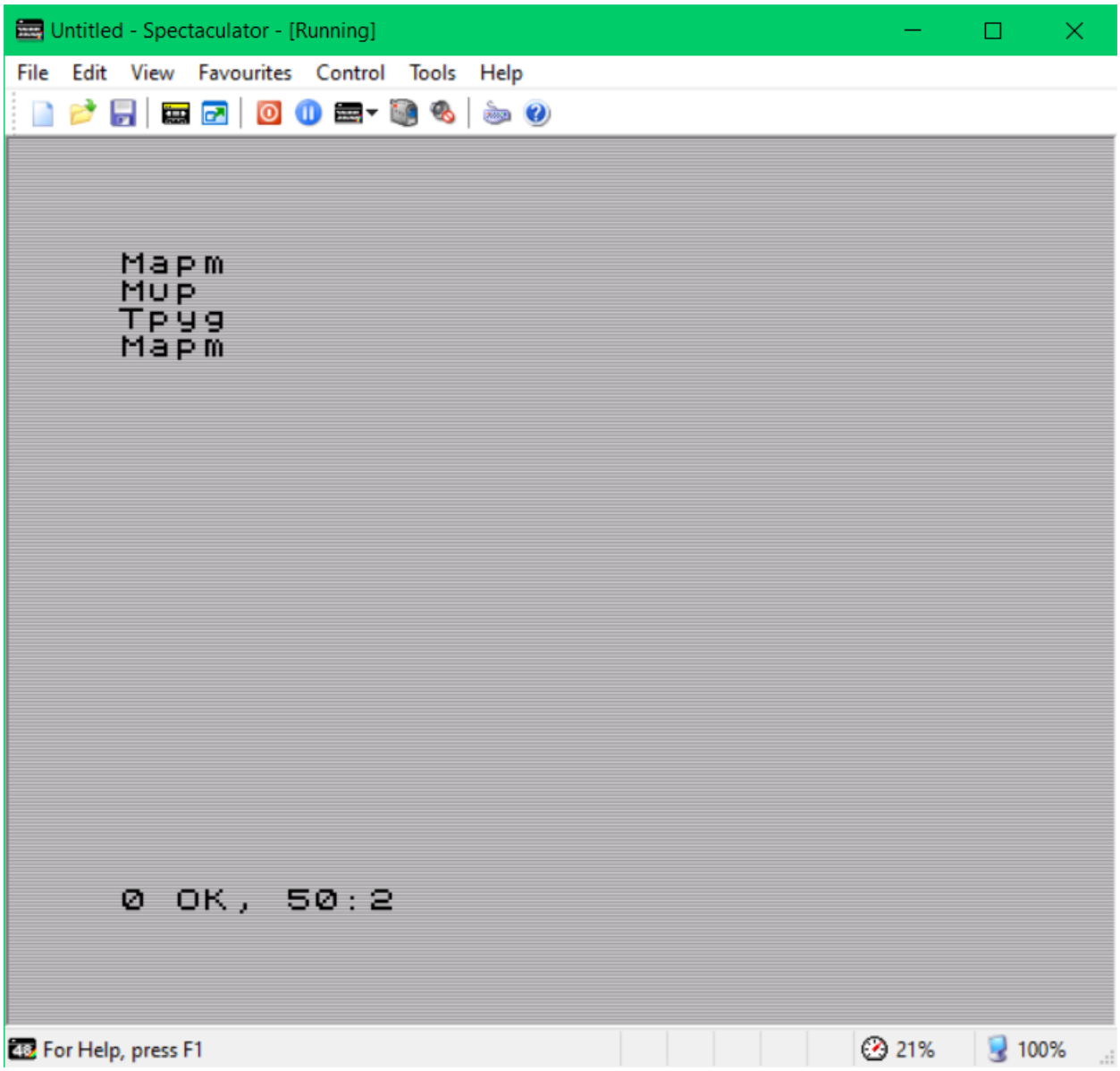


Рис. 231.

Всё как заказывали. Первый RETURN вывел вторую половину строки 50, следующий запустил всю программу, начиная со сроки 20. Именно в таком порядке они и были положены.

Если с командами всё понятно, то может возникнуть вопрос другого плана: а почему март? Ведь советский лозунг звучал: «Мир Труд Май». Да верно, но, «Мau» в шрифте Спектрума смотрится менее эстетично чем «Мapm». Ну и время сейчас такое. Глобальное потепление на всей планете. Скоро март будет как май, поэтому точно также на перспективу я модифицировал лозунг.

Обратите внимание, на интересный момент, а точнее слабое место подпрограммы RETURN (7971) по адресу 7975. Напрямую проверяя на маркер «62», разработчики вряд ли подумали, что кому-нибудь придёт в голову сделать строку с номером 15872, которая в точности повторит основание SP-башенки (0, 62).

Пусть имеется такая BASIC строка:

```
15872 GO SUB 16000
```

Запустив её, на основание SP-башенки отложится 0, 62, 2. А теперь, если попытаться командой `RETURN` снять записанное значение, в ответ выскочит сообщение «`7 RETURN without GO SUB`» и ничего не произойдёт. Кроме того, заряженные значения не израсходуются, и будут стоять на фундаменте SP мёртвым грузом:

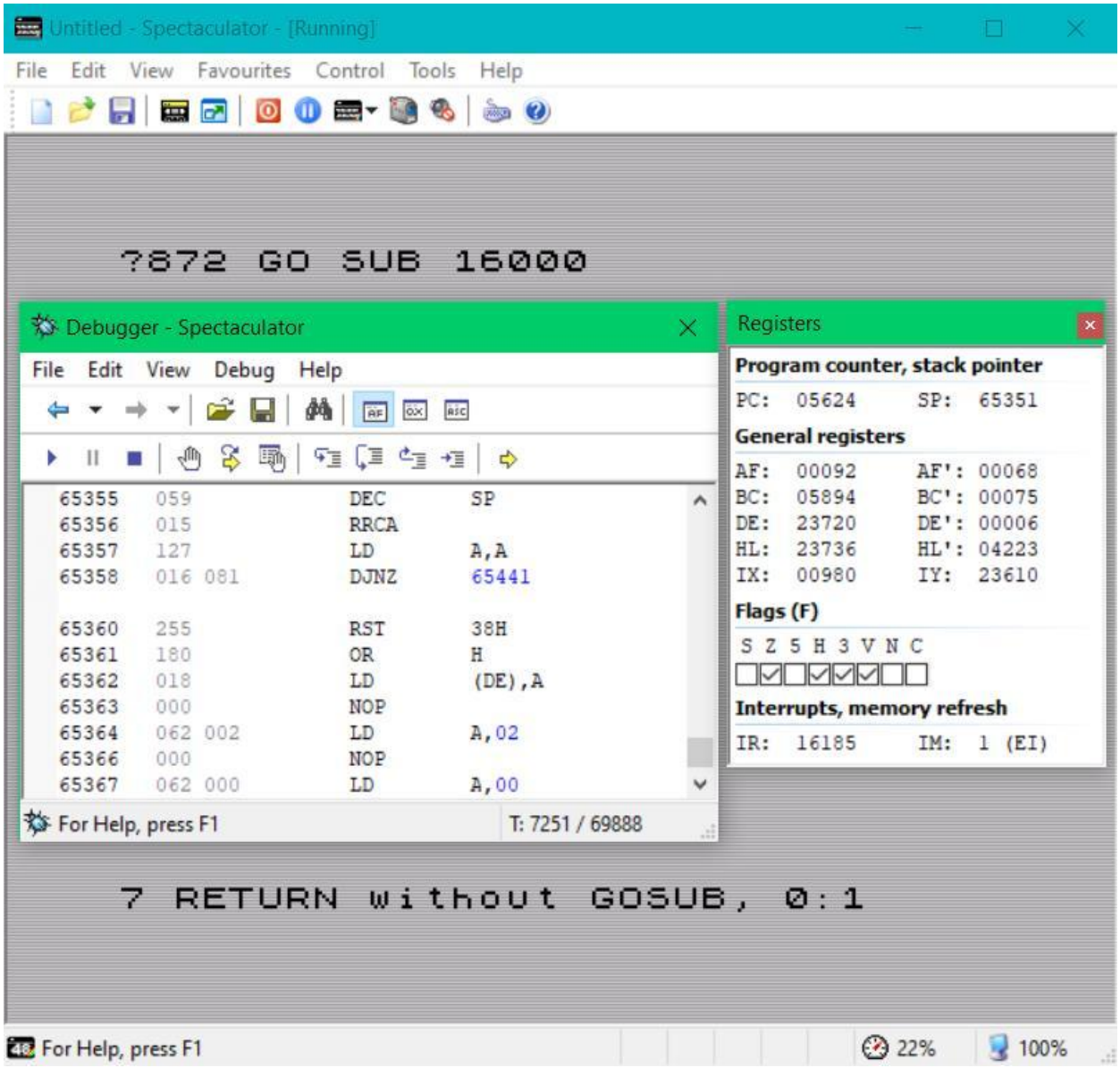


Рис. 232.

Удалить с фундамента «GO SUB-заряд» можно командой `RUN` с номером, большим, чем строка 15872.

Глава 21
CONTINUE

Краткое содержание: CONTINUE

Существует ещё один вариант запуска BASIC программы по команде `CONTINUE` (ЦОНТИНУЕ):

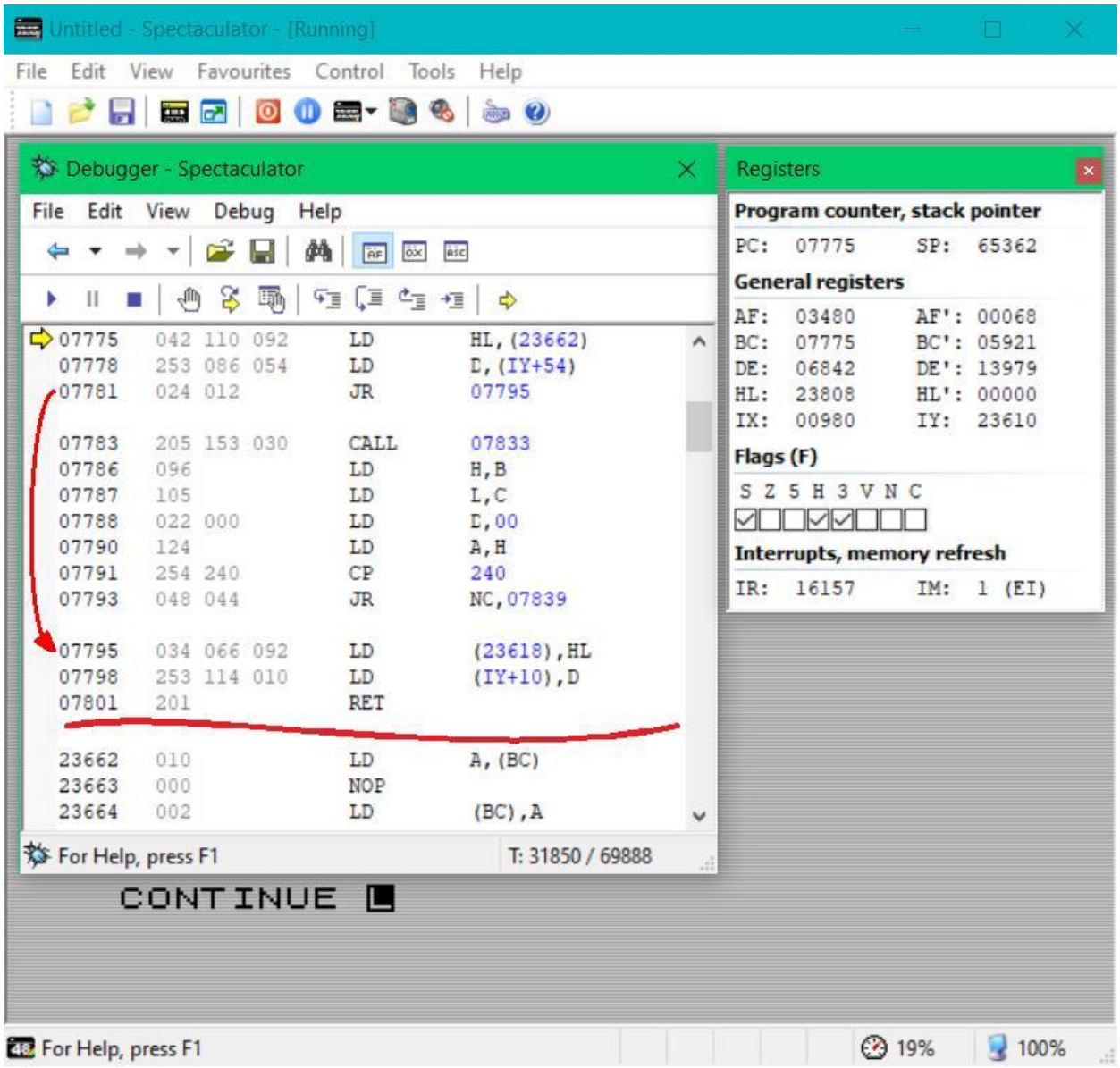


Рис. 233. Схема работы команды CONTINUE.

Принцип действия команды еще более простой, чем `GO SUB`. Из переменной `OLDPPC (23662)` берётся номер выполненной строки, а из `OSPPC (23664)` номер только что выполненной команды. На этом программа этой команды заканчивается. Дальше с помощью второй половины программы `GO TO` всё копируется в `NEWPPC (23618)` и `NSPPC (23620)` и начинается выполнение программы.

Попробуйте ввести следующую программу:

```
Debugger
Dec
Go To 23560
23560 ← 13
23611 ← 32
23627 ← 23806
23641 ← 23807
23649 ← 23809 23809 23809
23662 ← 10 0 2
23755 ← 10 0 27 0 245 34 66 32 110 111 112 109
23767 ← 121 34 58 245 34 72 65 32 75 65 72 79 72
23780 ← 69 80 75 69 34 013 0 20 16 0 245
23791 ← 34 114 111 112 117 109 32 84 65 72 75 69
```

```
23803 ← 80 34 13 128 13 128
Trace
```

После ввода программы наберите с клавиатуры натуральным способом команду `CONTINUE`:

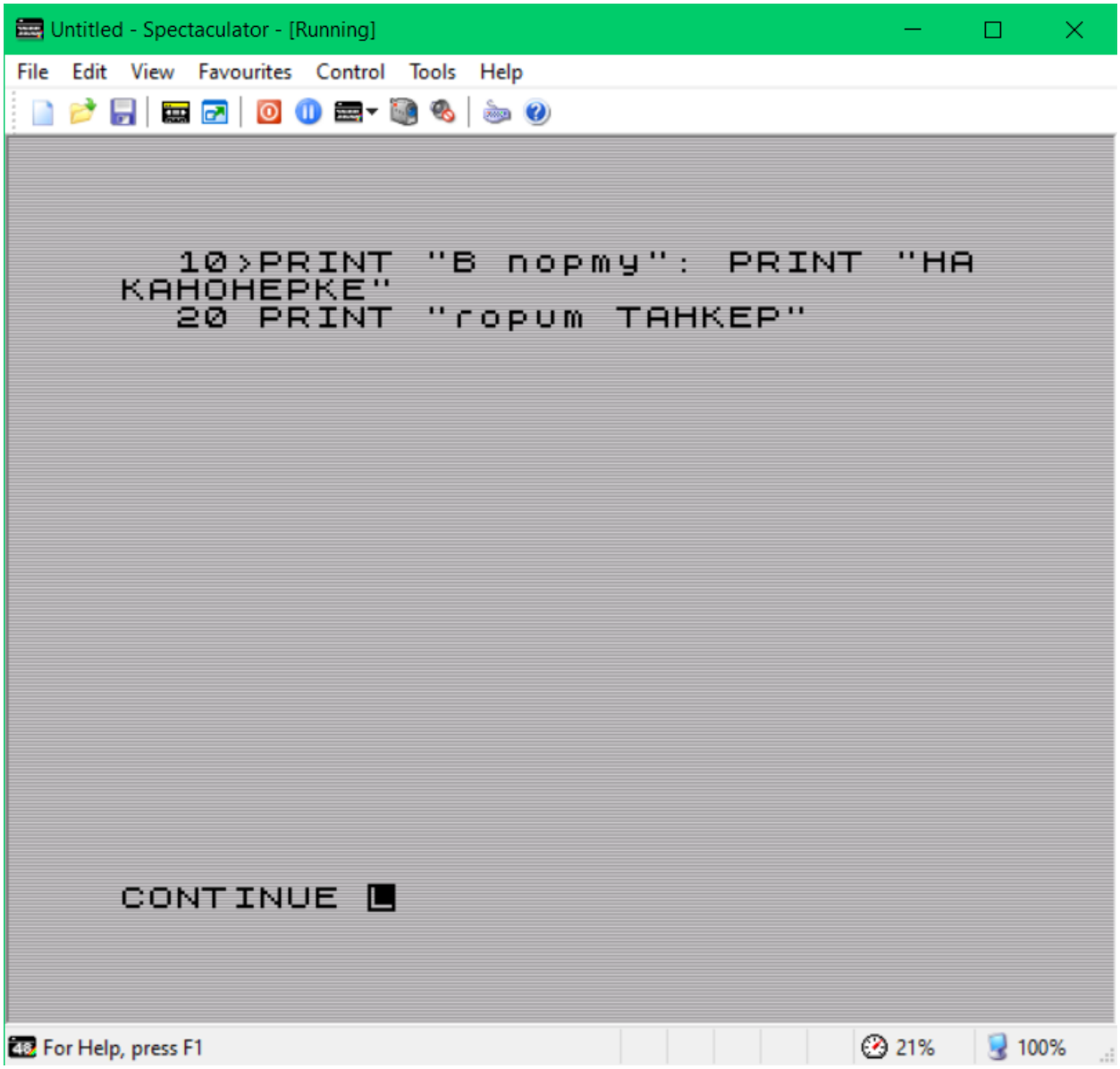


Рис. 234. Процесс набора команды CONTINUE для запуска программы.

Вводите команду:

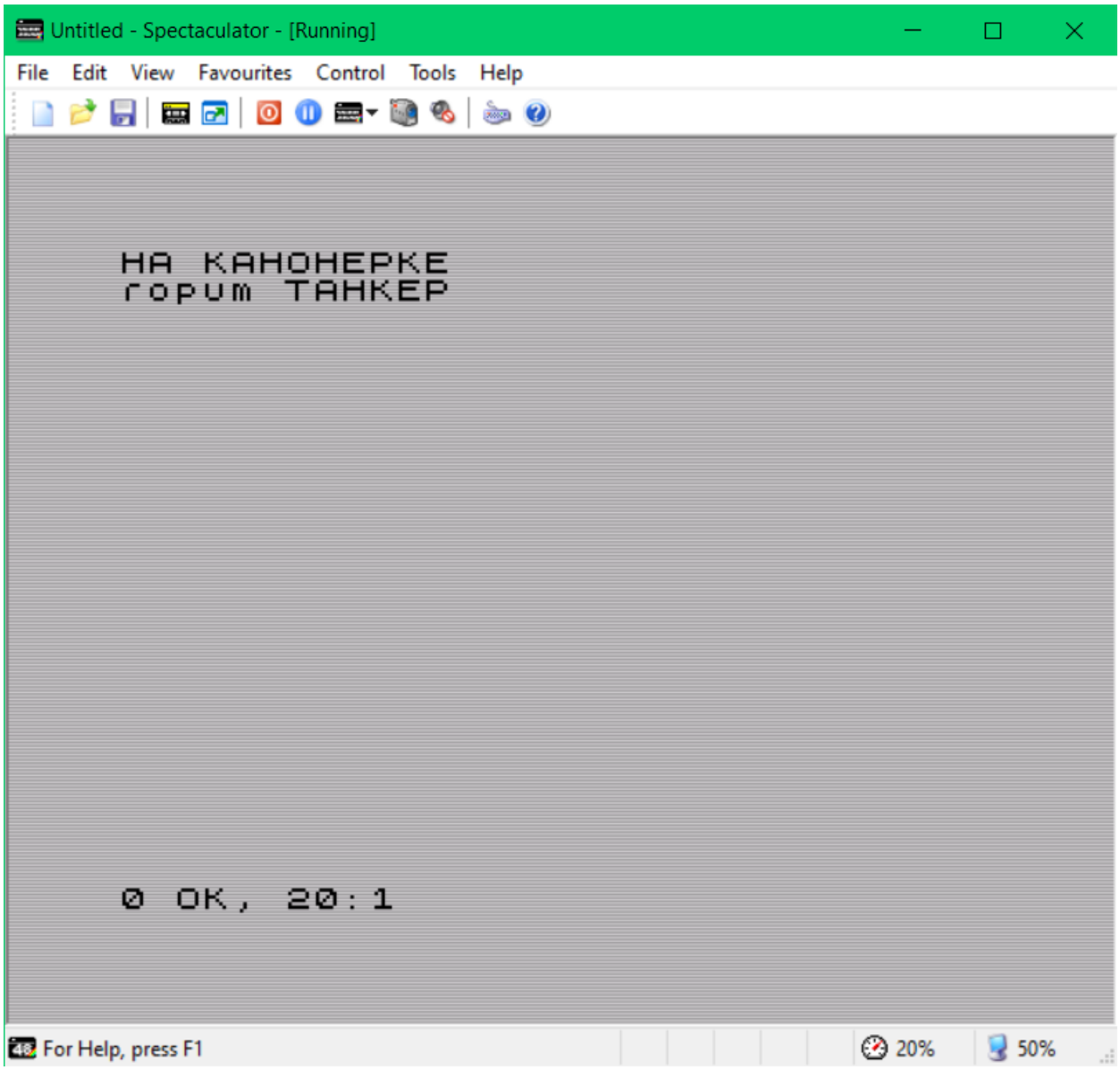


Рис. 235.

Как и планировалось, программа запустилась со второй половины строки 10.

Глава 22
NEXT без FOR как лемпира без квачи

Краткое содержание: VARS (23627), FOR, NEXT

Каким нестандартным образом ещё можно запустить программу? Оказывается с помощью команды `NEXT`. Сложная система `FOR-NEXT` является гибридом переменной с элементами запуска программы. Её результаты точно также хранятся в области системных переменных, на которую указывает `VARS (23627)`.

Введите следующую программу:

```
Debugger
Dec
Go To 23560
23560 ← 13
23611 ← 32
23627 ← 23812
23641 ← 23813
```

```
23649 ← 23815 23815 23815
23755 ← 0 1 18 0 245 34 121 32 103 112 121 114
23767 ← 97 34 58 245 34 66 32 34 59 13 0 2 31 0
23781 ← 245 34 75 65 80 77 65 72 69 32 34 59 66
23794 ← 59 34 32 75 111 112 101 117 99 107 117
23805 ← 120 32 66 79 72 34 13 128 13 128
Trace
```

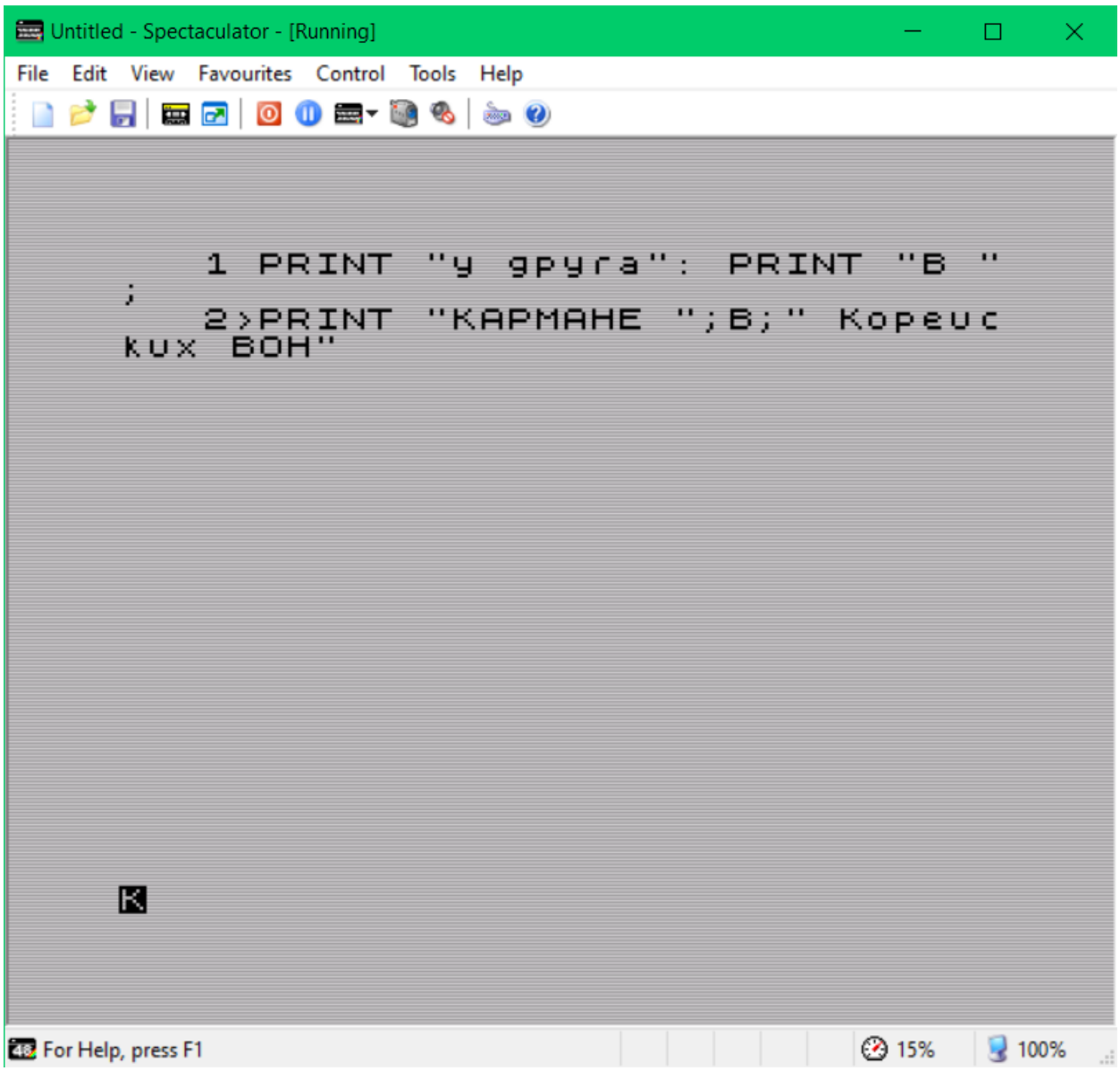


Рис. 236.

Да, друг выбрал валюту с расчётом на будущее. Ладно, открою страшную тайну. Меня вообще очень сильно тревожит, что последний год Замбийская квача сильно просела к Гондурасской лемпире:

Рис. 237. Текущий курс Гондурасской лемпиры к Замбийской кваче.

Теперь снова возвращаюсь к программе. Для запуска нужно ввести результат выполнения строки:

```
FOR B=2 TO 6 STEP 2
```

Итогом работы должна стать такая числовая последовательность:

```
226 0 0 2 0 0 0 0 6 0 0 0 0 2 0 0 1 0 2
```

где,

226 – индекс переменной +224 смешанный со 2-й буквой алфавита «b» (224+2=226)
0 0 2 0 0 – числовое значение в рыхлом виде, изначально приравниваемое к «b».
0 0 6 0 0 – числовое значение в рыхлом виде для команды TO.
0 0 2 0 0 – числовое значение в рыхлом виде для команды STEP
1 0 – номер строки, с которой запустить программу командой NEXT.
2 – номер оператора (куска строки до двоеточия) с которого запустить программу.
128 – маркер конца области.

Для виртуального набора этого цикла введите программу:

```
Debugger
Dec
Go To 23641
23641 ← 23832 23832
23649 ← 23834 23834 23834
23812 ← 226 0 0 2 0 0 0 0 6 0 0 0 0 2 0 0 1 0 2
23831 ← 128 13 128
Trace
```

После выполнения алгоритма и выхода в Spectaculator наберите `NEXT b`:

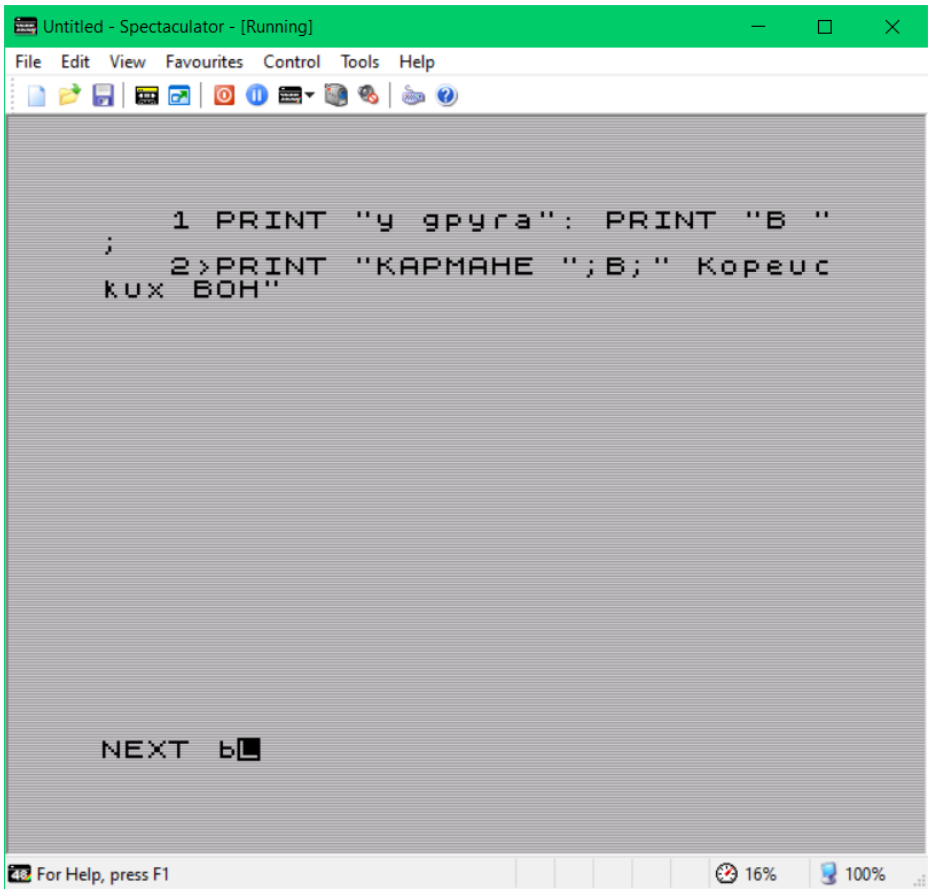


Рис. 238. Подготовка к проверке искусственно синтезированного цикла FOR.

Введите и снова наберите `NEXT b`:

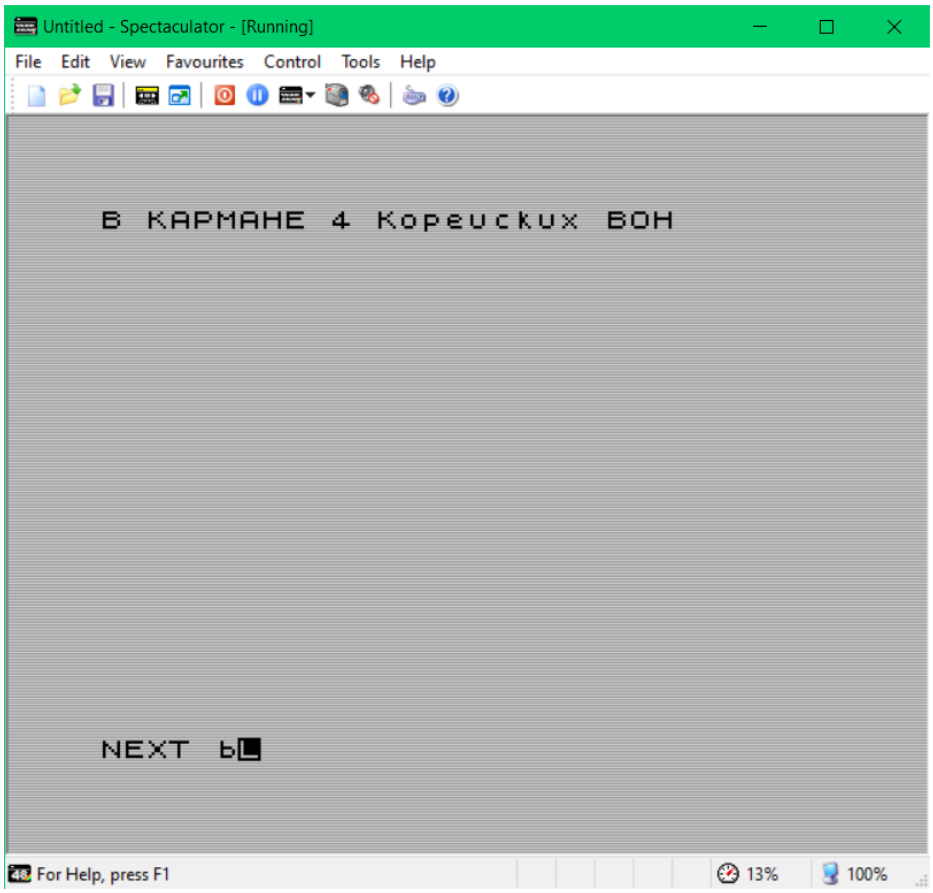


Рис. 239.

В кармане лежат целых 4 воны. Обратите внимание, как друг мастерски перевёл стрелки на вас, по тихому убрав себя из списков миллиардеров, по версии журнала «*Forbes*». Как и ожидалось, выполнилась вторая половина строки 1, и остальные до конца. Переменная «b» стала равной 4.

Вводите второй раз, а затем наберите третий раз NEXT b:

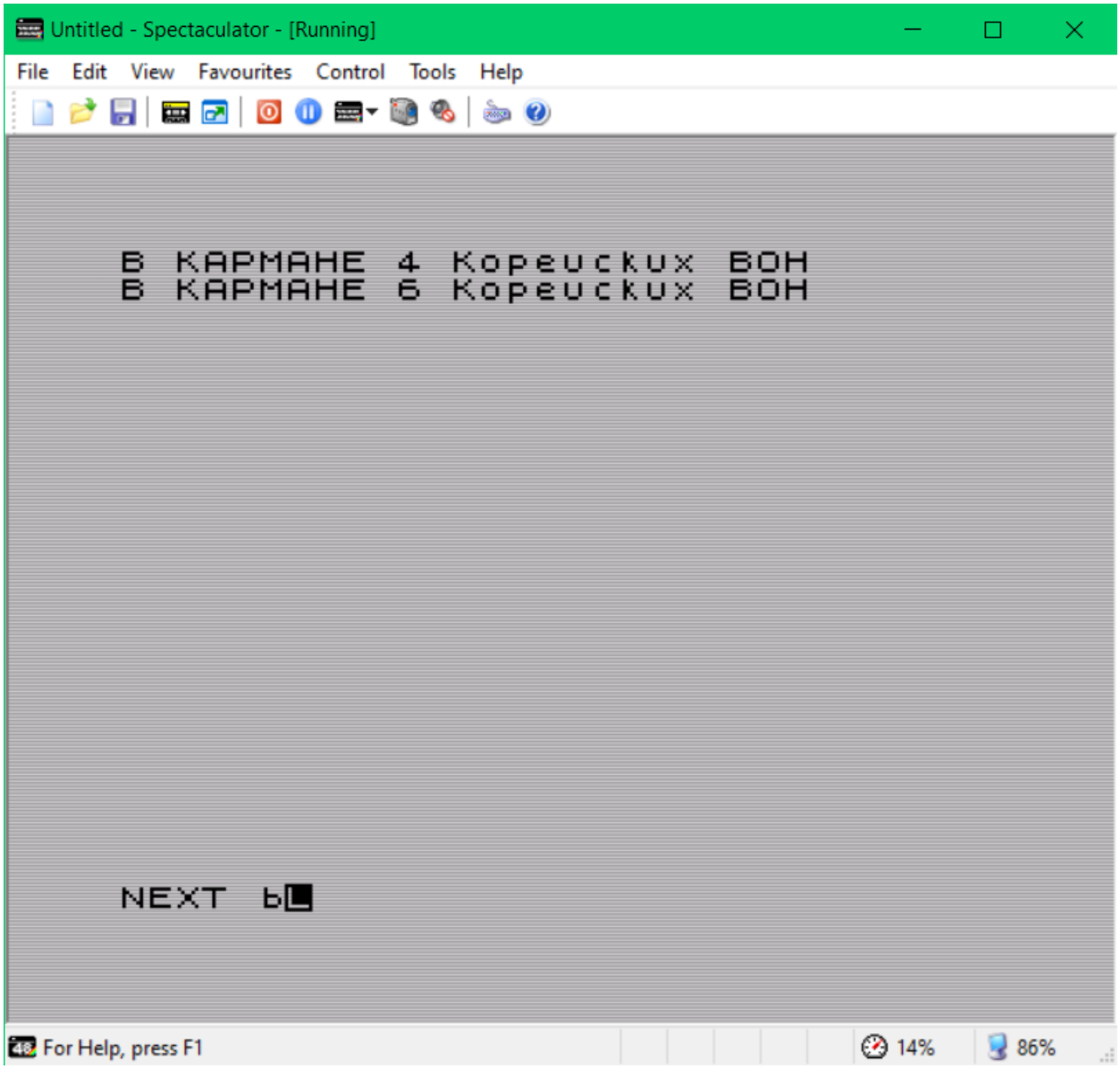


Рис. 240.

После ввода на экран выдалась обновлённая информация о текущем счете в кармане спортивных штанов. Отыскались завалившиеся еще 2 воны. Епть, да я бохатый!

Заключительный раз введите строку. На этот раз выведется сообщение «0 OK» и ничего не произойдёт. «Зарядка» NEXT закончилась.

Откройте «*Debugger*» и посмотрите на вводимую переменную с адреса 23812:

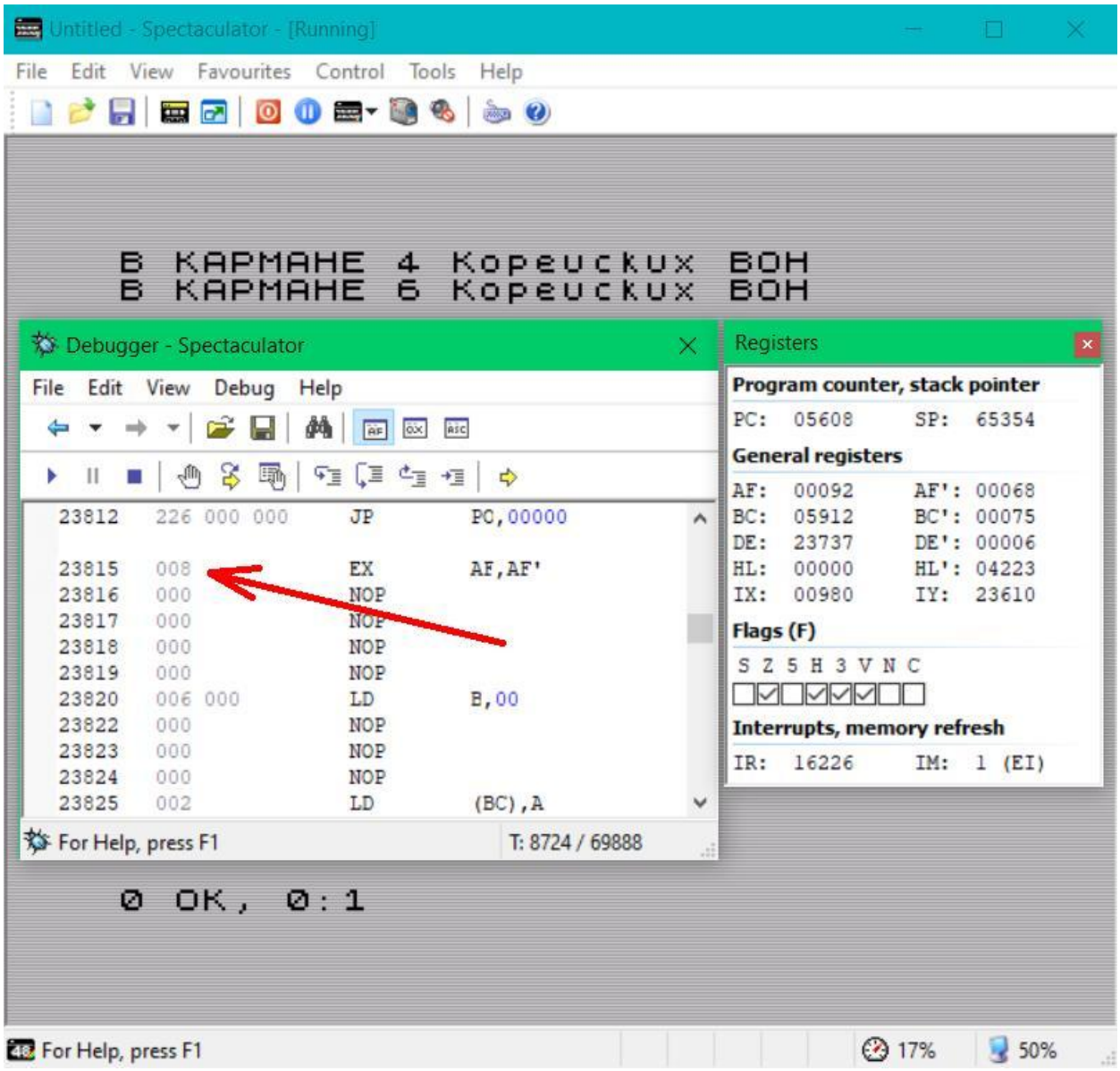


Рис. 241.

Обратите внимание на изменения, которые произошли в ячейке 23815: число 2, которое вы записывали изначально, превратилось в 8. Выполняясь, команда `NEXT` увеличивает это значение на шаг прироста `STEP` (23825) и выполняет программу с указанного фрагмента. Это происходит до тех пор, пока оно меньше или равно значению `TO` в ячейке 23820. Как только значение превысит этот предел, запуски программы прекратятся, и `NEXT` будет выполняться вхолостую, но продолжая увеличивать значение ячейки 23820.

Восстановить работоспособность `NEXT` можно несколькими способами. Достаточно вернуть разность значений `FOR` и `TO`. В данном случае значение в ячейке 23815 должно быть меньше, чем в 23820, и `NEXT` снова заработает. Можно выставить назад значение для `FOR`, но я предлагаю поменять значение `TO` в 23820 на 10:

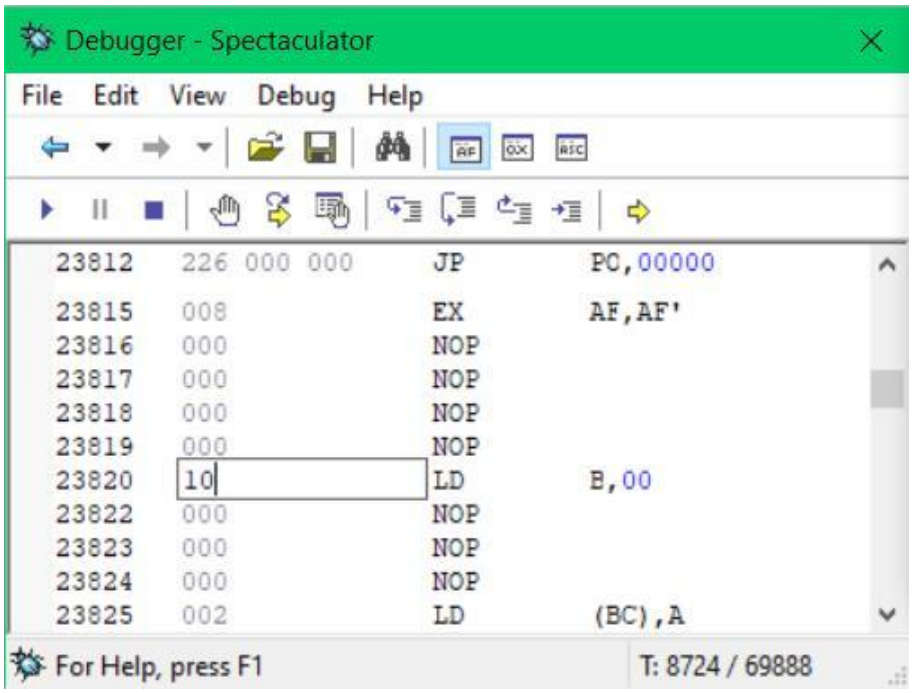


Рис. 242.

Выйдете из отладчика и наберите на клавиатуре NEXT b: NEXT b:

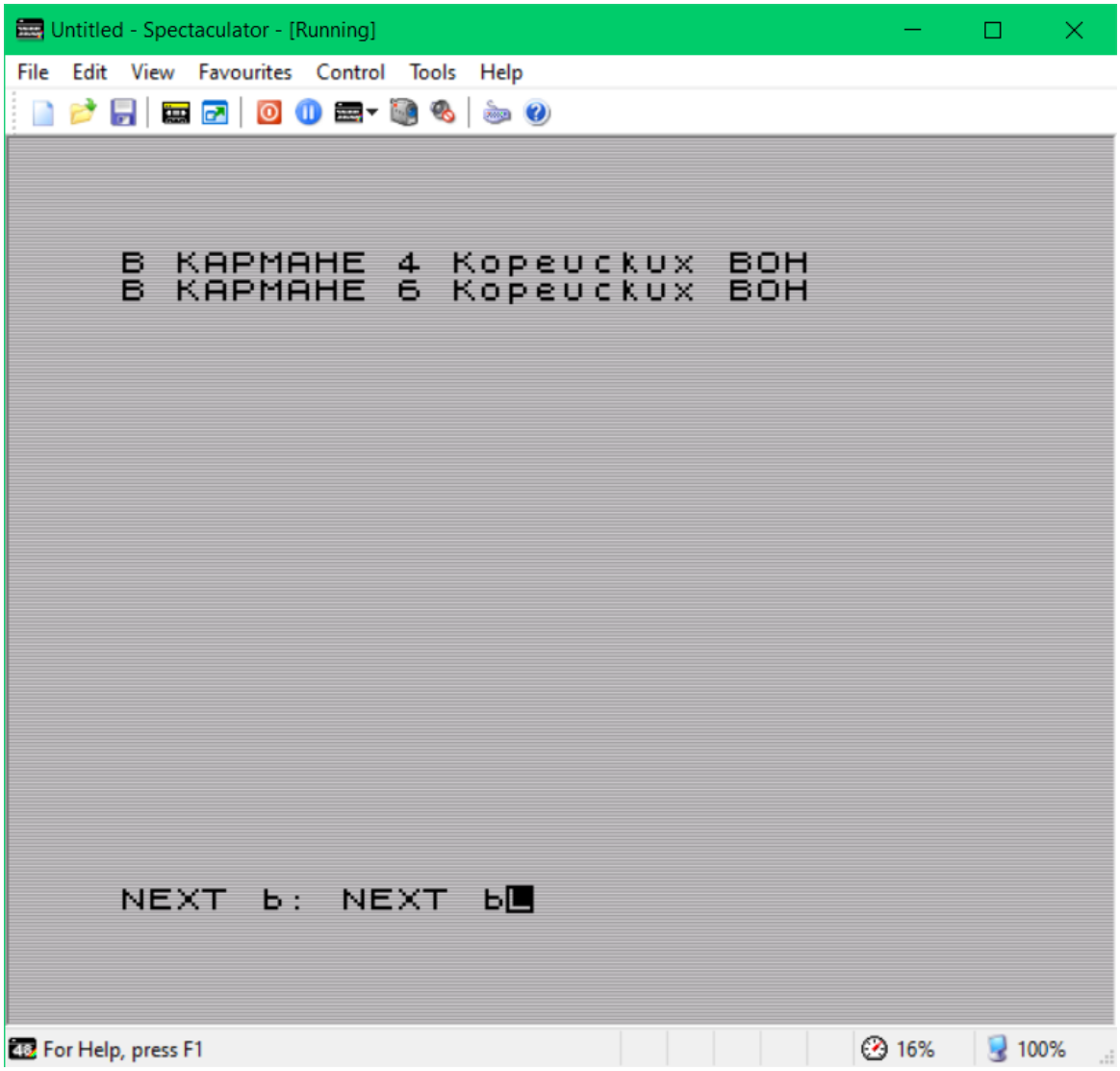


Рис. 243.

После ввода, NEXT запустил программу всего один раз, и снова отключился:

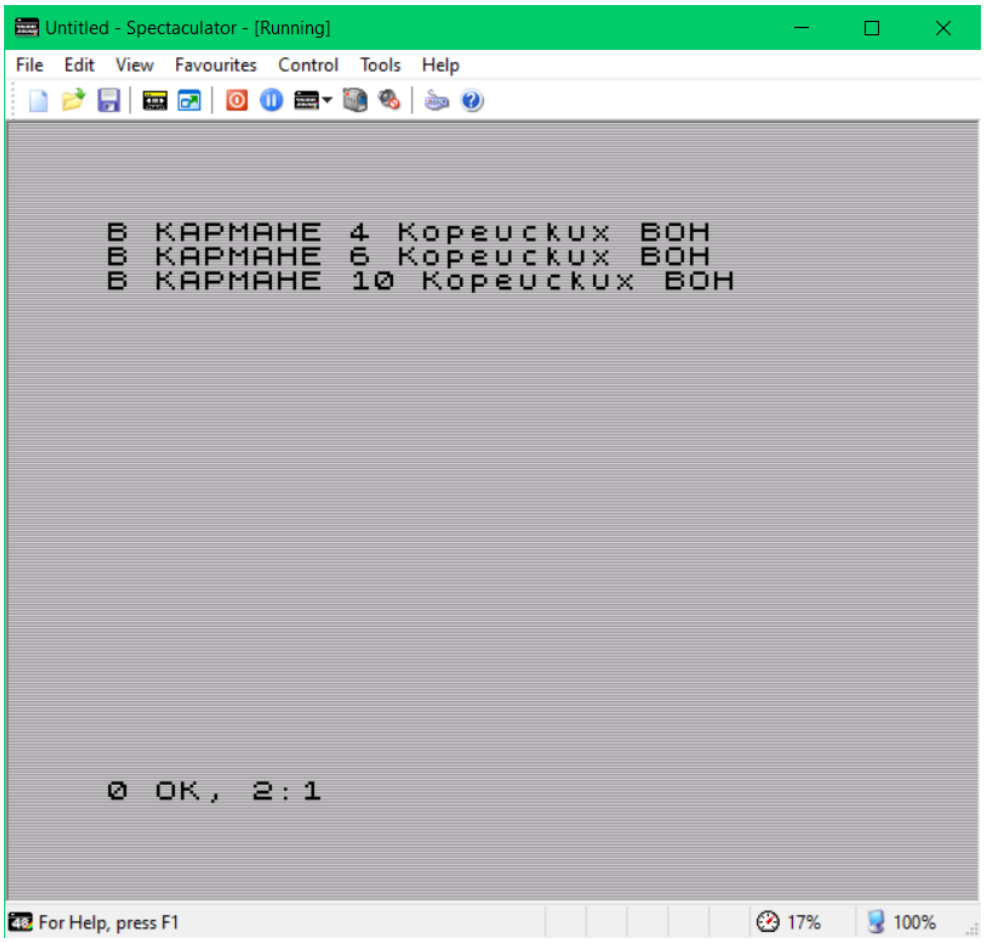


Рис. 244.

О!!! Целых 10 Вон. Ну всё, тушите свет и на перерыв. Я побежал на шоппинг тратить деньги. До встречи в следующей главе.

Глава 23
Числа и их обложки

Из прошлых глав (да из любых других книжек) вы помните, что структура числа после раскрытия в строке жрёт очень много памяти. До текущей главы я не акцентировал на это внимание, потому что были более приоритетные темы. Кажется, сейчас дошла очередь до чисел.

Вот, например, BORDER 0. Перед выполнением, вопреки здравому смыслу команда разместится в ячейках как «231 48 14 0 0 0 0 0», а не «231 48». Помимо символьного представления, добавляется разделитель «14», а следом вот такое чудо из пяти нулей. Для чего это было нужно? В 1982 году предполагалось, что главное предназначение этого компьютера будет для обучения языку BASIC, поэтому модель затачивалась под него. Короче, чтобы дети на занятиях командой PRINT учились складывать не только целые числа, но и всякие дроби-хуёби с синусами. Любопытные свойства структуры чисел выяснились позже, когда начался игровой бум. Вскрывающиеся глюки от недоработки вывода чисел и арифметических операций стали использовать в «защитах» во время загрузки.

Дело в том, что в такой рыхлой структуре, числа до 65535 записываются в привычном 2-х байтном формате, где для кодировки используются 3-й и 4-й байт. Начиная с 65536, или появления дробных значений формат меняется, и для кодирования начинают использоваться все 5 байт. Однако, некоторые малые числа можно записать как

в двухбайтном, так и в полном пятибайтном формате. При этом они будут не только корректно отображаться, но и работать.

В том виде, в котором планировалась работа с числами, лично для меня никакого интереса не представляет, а вот разные интересные глюки изучить стоит. Мало того, при детальном исследовании, всегда рождаются эксклюзивные открытия.

Предлагаю ввести такую BASIC строку, которая выведет на экран красную рамку:

```
1 РАМКА В КРАСНОМ ОТТЕНКЕ
```

Если что, я не перегрелся. На языке «*God Mode*» введите и запустите следующую программу:

```
Debugger
Dec
Go To 23560
23560 ← 13
23611 ← 32
23627 ← 23799
23641 ← 23800
23649 ← 23802 23802 23802
23755 ← 0 1 40 0 231 46 8 8 8 8 8 8 8 8
23769 ←РАМКА В КРАСНОМ ОТТЕНКЕ
23792 ← 14 0 0 2 0 0 13 128 13 128
Trace
```

Запустив алгоритм, на экране действительно появилась эта строка. Следом натуральным способом наберите RUN:

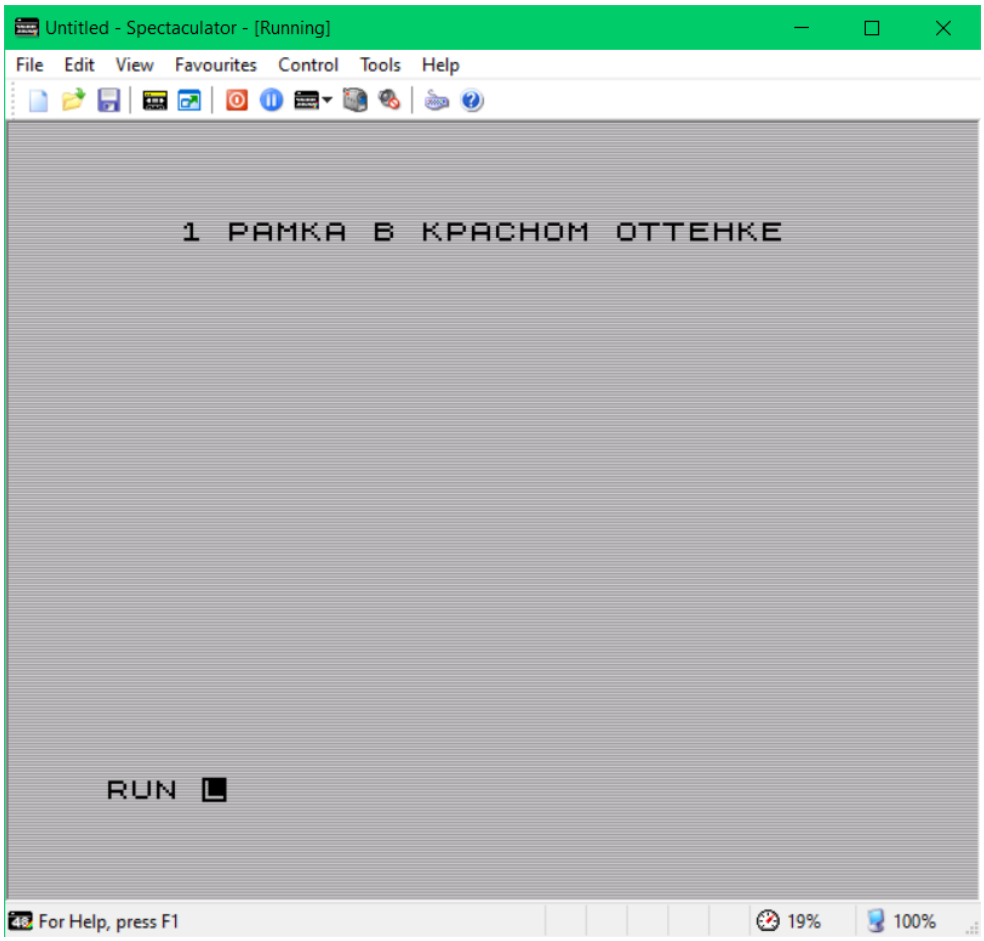


Рис. 245.

Вводите команду:

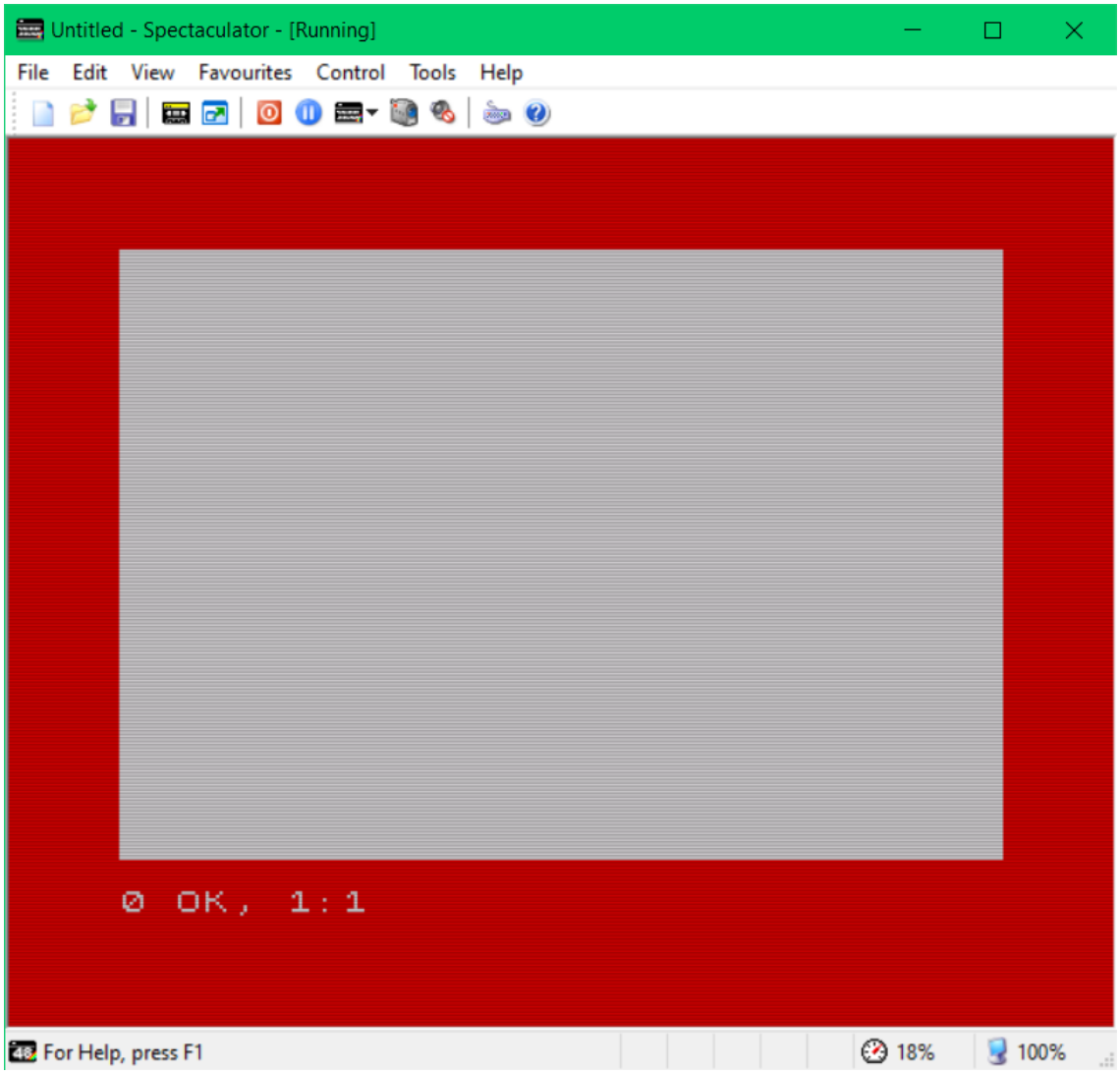


Рис. 246.

Высветилась обещанная красная рамка. Как это получается? Дело в том, что число в расширенном формате условно можно разделить на три части:

Первая часть, это символьное представление числа. Своего рода формальная обложка для отображения на экране.

Следом за символьной цепочкой стоит маркер-разделитель «14», который указывает, на то, что начался блок зашифрованного числового значения.

Третьим фрагментом стоит пять байт числа, которое и считывает для вычислений программа.

При вводе традиционным способом, видовое (символьное) представление соответствует числовому, ну а внешним методом вы можете в эти куски ввести противоречивую информацию, но для успешного выполнения программы есть несколько ограничений.

Если в числовую часть можно ввести любое значение от «0 0 0 0 0» до «255 255 255 255 255», то у видовой имеются ограничения.

Первый символ «обложки» должен начинаться с кодов 46, 48, 49, 50, 51, 52, 53, 54, 55, 56 или 57. Иначе при попытке запуска строки выдастся сообщение: «C Nonsence in BASIC». Второй символ может быть уже любым (0-255), при условии, что первый был из вышеуказанного интервала.

Самым главным преимуществом может стать размещение в видовой части числа машинной программы, а адресе запуска разместить в числовой.

Теперь обратный пример использования такого формата. «Обложку» чисел можно оставить без изменений, а внутренности чисел нафаршировать адресом запуска и программой в машинных кодах, которая для примера выведет синюю рамку. Внешне строка будет выглядеть вот так:

```
1 LET a=USR 1+1
```

Для её ввода выполните следующую «*God Mode*» программу:

```
Debugger
Dec
Go To 23560
23560 ← 13
23611 ← 32
23627 ← 23779
23641 ← 23780
23649 ← 23782 23782 23782
23755 ← 0 1 20 0 241 97 61 192 49 14 0 0 221 92
23769 ← 0 43 49 14 62 1 211 254 201
23778 ← 13 128 13 128
Trace
```

После запуска получилась такая неприметная строка с двумя сюрпризами. В первой единичке адрес запуска, а во второй сама программа. Наберите RUN:

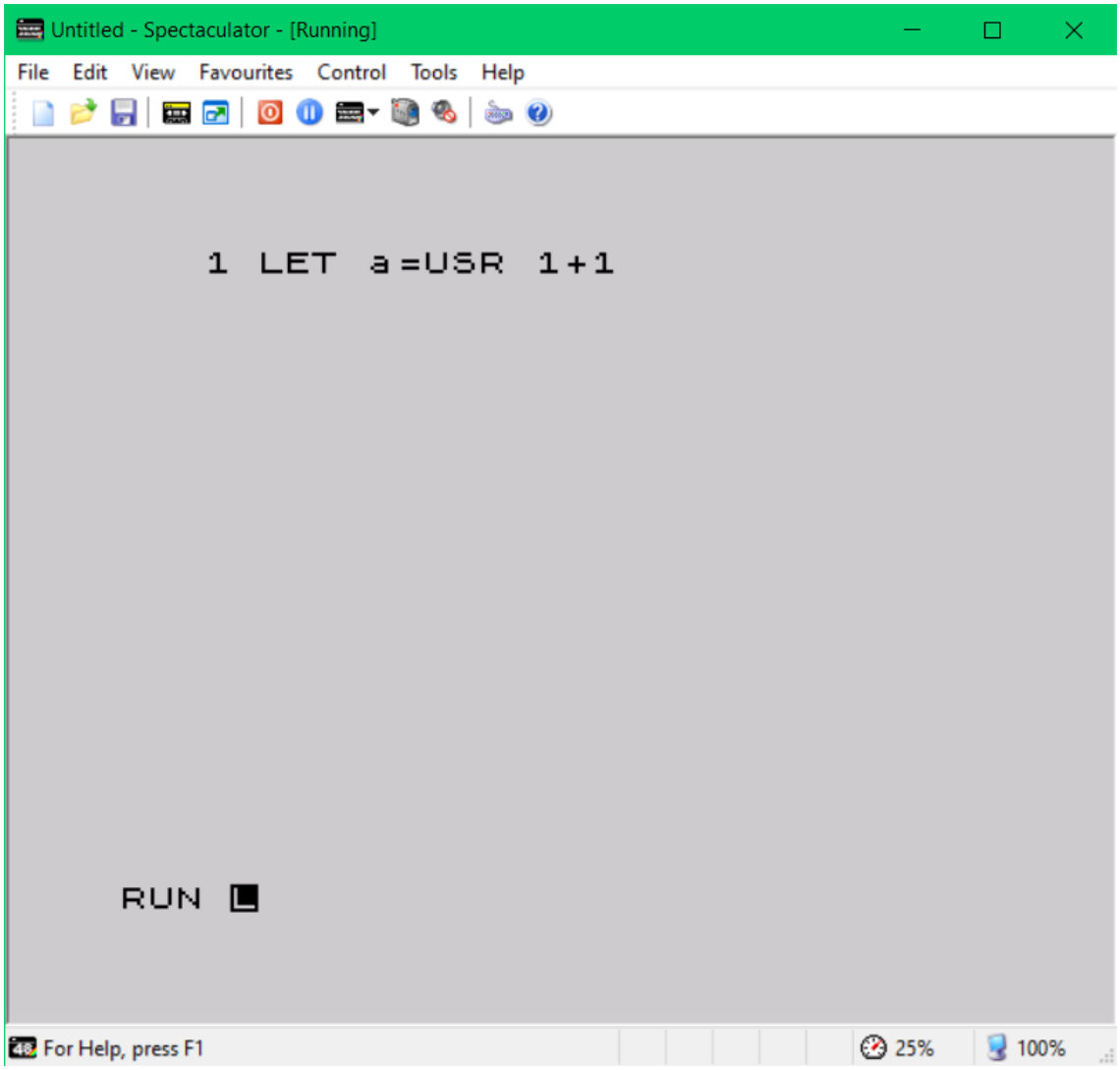


Рис. 247. Ввод BASIC команды RUN натуральным способом.

После нажатия ENTER на экран высветится синяя рамка. Программа выполнилась, как это и было задумано:

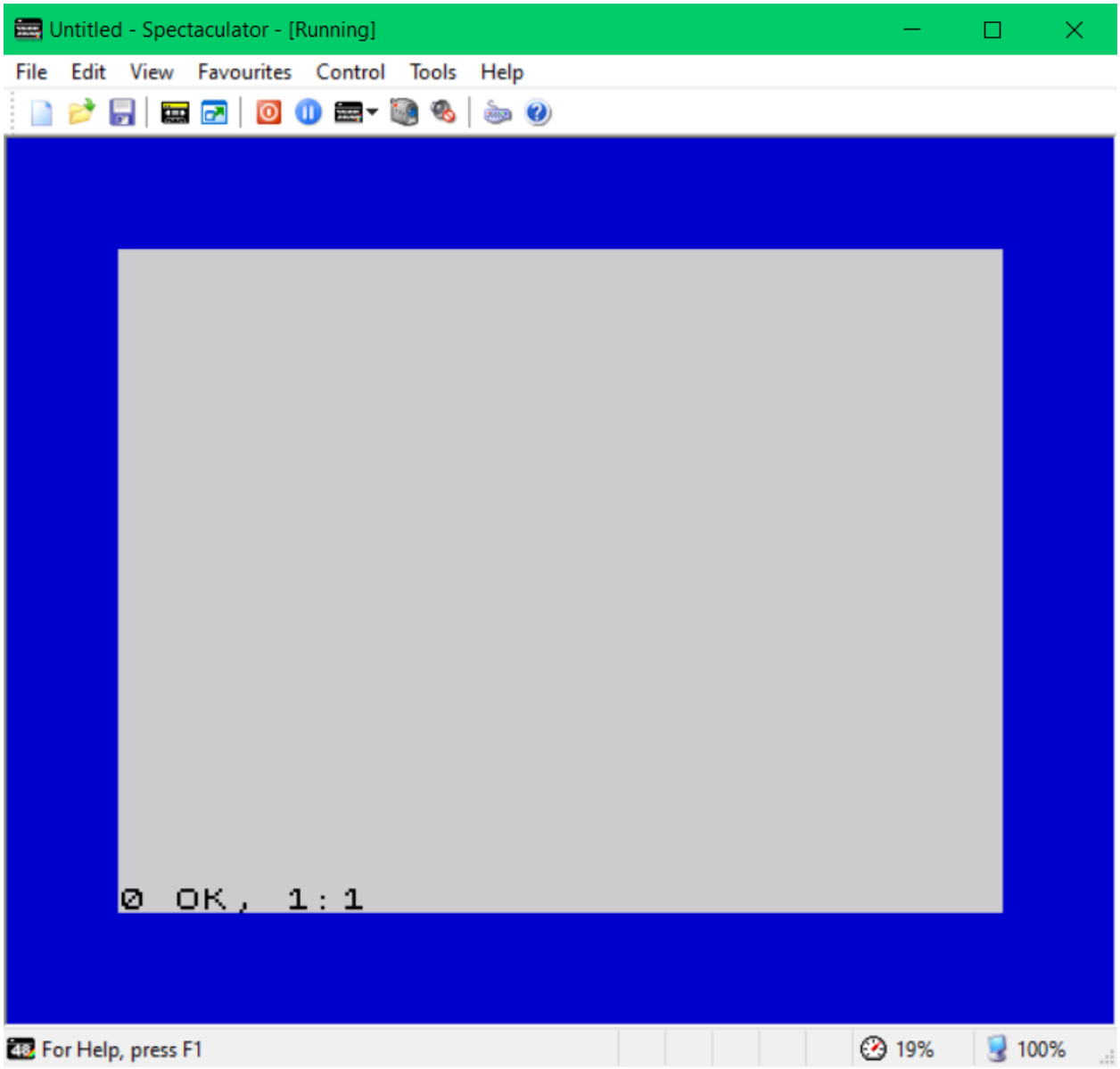


Рис. 248.

Таким образом, можно запихать и длинную программу, в любые подряд стоящие числа. Однако, её нужно дробить на кусочки по 5 байт, два из которых придётся отдавать команде «`JR +3`» (24, 3). Итого на код программы в каждом числе остаётся по три свободные ячейки. А вот 4-х байтные команды, типа `LD (IY+..)`, .. `SET .. (IY+..)` и прочие не прокатят. Негусто, но текст вывести можно.

Чтобы убедиться, введите следующую программу:

```
Debugger
Dec
Go To 23560
23560 ← 13
23611 ← 32
23627 ← 23822
23641 ← 23823
23649 ← 23825 23825 23825
```

```
23755 ← 0 1 10 0 249 192 50 14 0 0 224 92 0 13; 1 RANDOMIZE USR 2
[23776]
23769 ← 0 2 49 0 228 48 14 62 2 0 24 3 44     ; 2 DATA 0, [LD A, 2]
23782 ← 49 14 205 1 22 24 3 44                ; 1, [CALL 5633]
23790 ← 50 14 62 69 215 24 3 44          ; 2, [LD A, буква1 RST 16]
23798 ← 51 14 62 110 215 24 3 44         ; 3, [LD A, буква2 RST 16]
23806 ← 52 14 62 109 215 24 3 44         ; 4, [LD A, буква3 RST 16]
23814 ← 53 14 62 65 215 201 0 13 128 13 128   ; 5, [LD A, буква4 RET]
Trace
```

После ввода на экране появилась такая красивая, но бессмысленная программа. Наберите RUN:

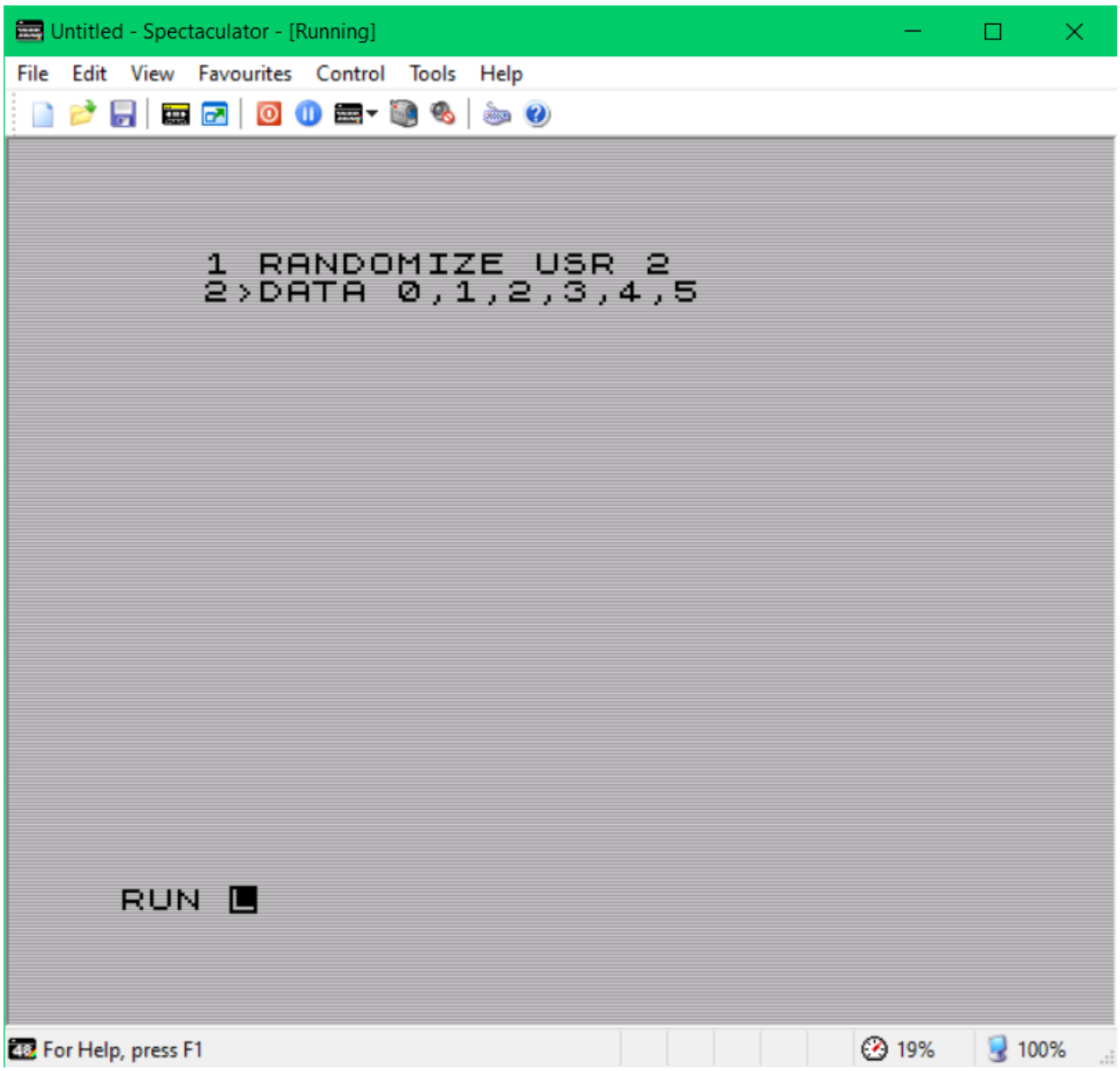


Рис. 249.

Запускайте программу и на экране появляется:

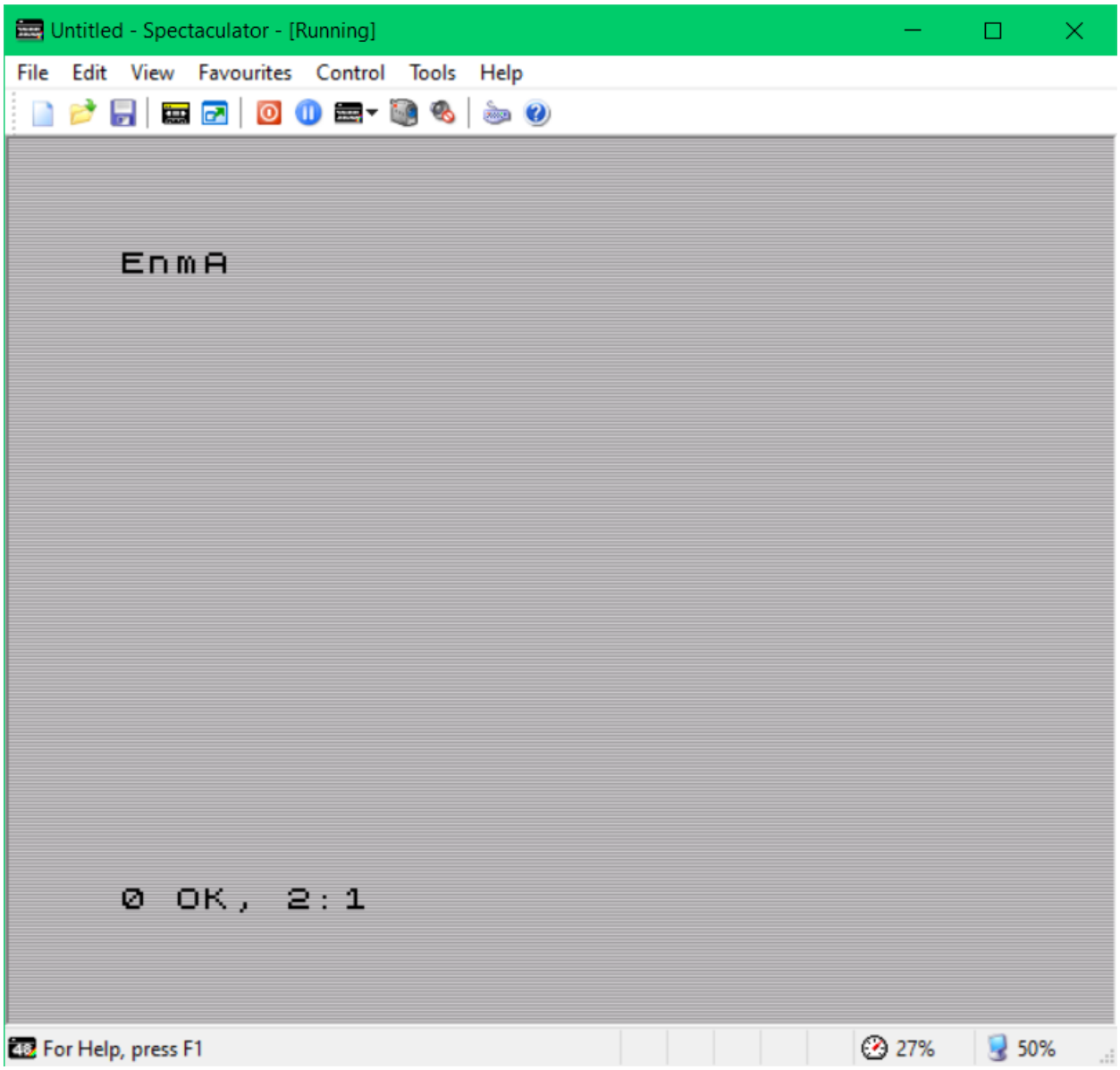


Рис. 250.

Ёпта собственной персоной! Такая красивая и аккуратная программа, и вдруг раз – а внутри «Ёпта». Прямо как в реальной жизни с двумя параллельными вселенными. Например, включаешь телевизор, а в нём показывают сказки про разных фей и волшебные страны со счастливыми людьми. Выходишь после этого в реальность, а там…

Рис. 251. Жизнерадостная Ёпта на заборе Фрунзенского СПХ. Южное шоссе 59. Фото 12 марта 2015 года.

Ёпта со всей вытекающей реальной жизнью! «Но Ёпта ведь жизнерадостная» – возразите вы, глядя на картинку. Соглашусь, но от этого она не переставЁт быть «Ёптой».

Глава 24
Симметричные подпрограммы. Основы маршрутизации Стрелочки

Краткое содержание возвратные и невозвратные подпрограммы, сопровождение, маршрутизация Стрелочки

Что такое жизнь? Это дорожка с разнообразными препятствиями, по которой несёт время без возможности остановки и повторной попытки пройти проваленные головоломки. Есть, конечно, куски относительно спокойные, но они недолгие.

В жизни Стрелочки всё точно также. Куда бы вы её не поставили, она будет идти вперед, а точнее вниз. Точно также дорожки, по которым ходит Стрелочка, на каждом шагу таят в себе ямы, заборы, катапульты и топкие болота. Если поставить Стрелочку в произвольное место наугад, скорее всего её путь будет недолгим. Чаще всего она просто улетит в неизвестность на первой «RET-катапульте», или увязнет в топком болоте графических данных.

Но не всё так грустно. При определённых вводных данных, можно пускать Стрелочку по этим участкам и ловить в конце. Для этого нужно проанализировать кусок дорожки и найти глухие барьеры, через которые нет переправы. Как правило, отрезок пути, начинающийся после JP или RET-телепортёра, будет началом какой-либо автономной или вспомогательной подпрограммы.

Грамотно задав маршрут и сказав доброе слово, можно провести Стрелочку по заданному участку. Но даже относительно ровные и спокойные куски таят в себе опасности. Попадающиеся временные «CALL-телепортёры», иногда могут запустить Стрелочку в один конец, без возможности возврата. Как вы догадались, в этой главе речь пойдет о типах и видах подпрограмм, а также способах сопровождения Стрелочки по их фрагментам.

Итак, типов подпрограмм, из которых создан BASIC, можно разделить на два класса: симметричные (возвратные) и несимметричные (невозвратные).

Наиболее простыми являются симметричные программы. С них и предлагаю начать знакомство. Структурная схема выгладит следующим образом:

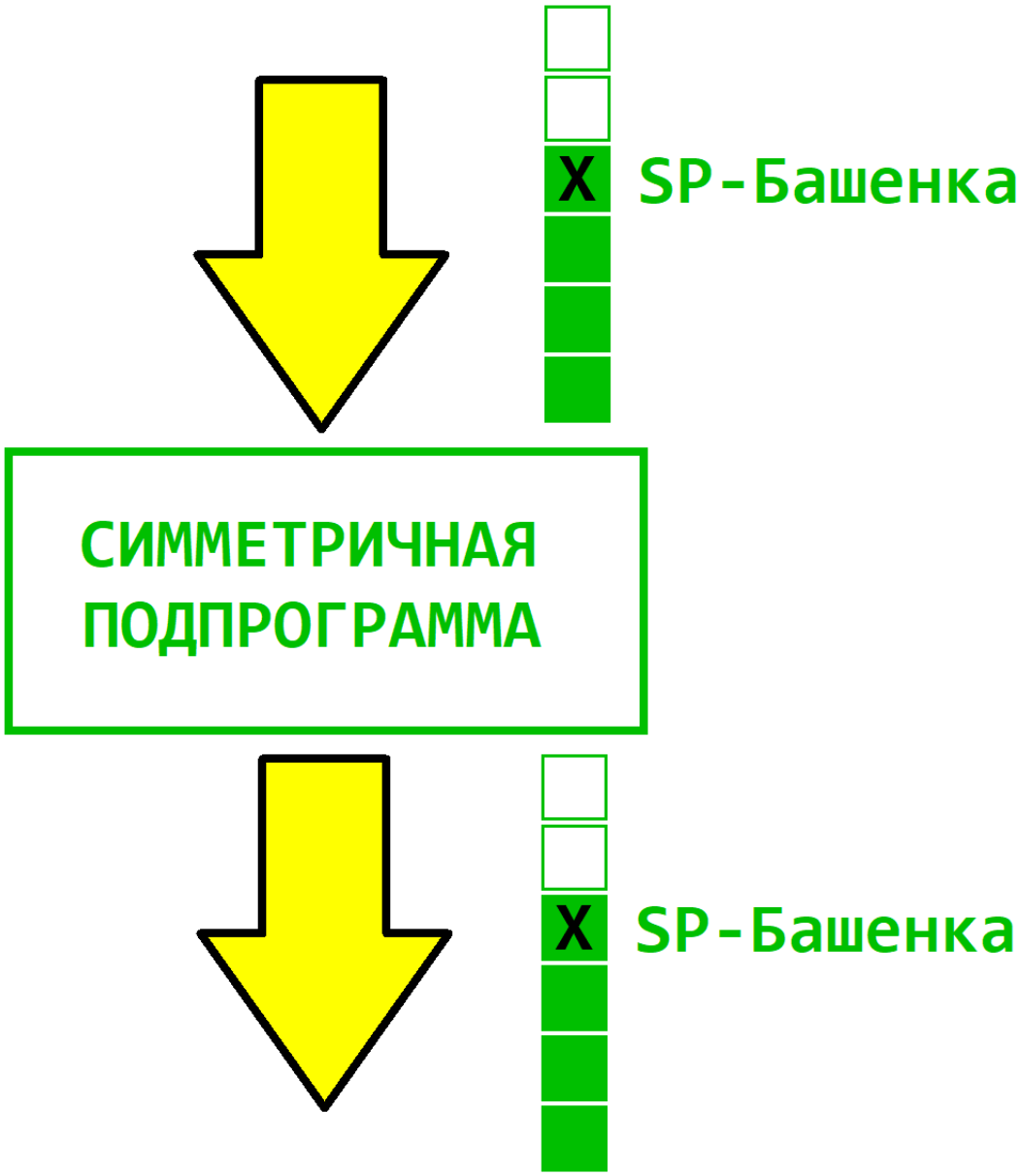


Рис. 252.

Что такое симметричная подпрограмма? Это когда при заходе внутрь и выходе назад столбик «SP» сохраняет равновесие, не укорачиваясь и не вырастая.

Рассмотрим симметричную программу, которая формирует BASIC команду `PAUSE 5`:

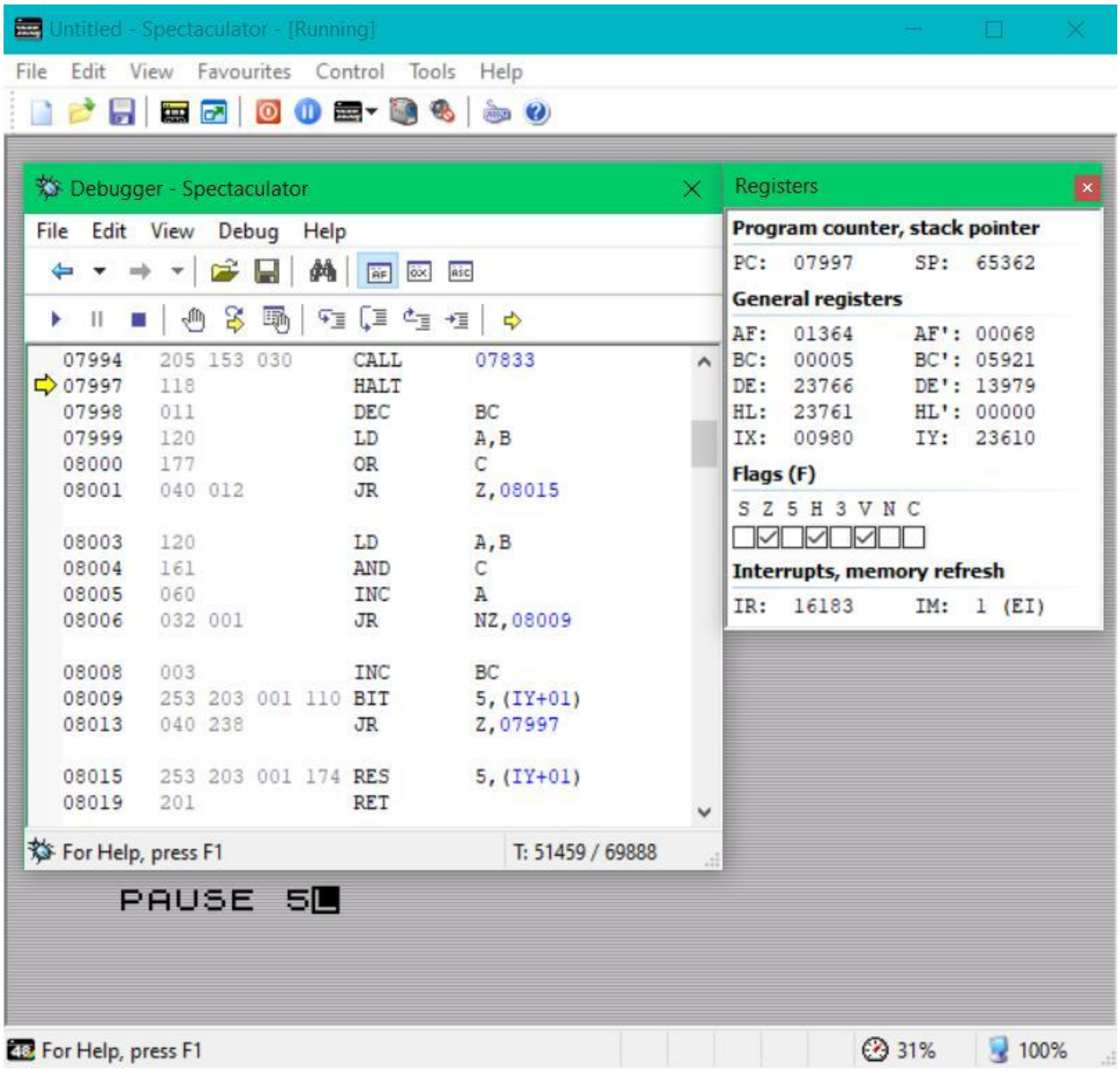


Рис. 253.

После подбора действий для введённой команды, с адреса 7198 Стрелочка попадает на подпрограмму команды PAUSE, расположенную в 7994. Быстро преобразовав символы за командой в числовой вид (CALL 7833) Стрелочка записывает полученное значение в «BC» и возвращается назад. Вот вам первый пример симметричной программы. Перед выполнением команды CALL 7833 и после выхода из неё SP-столбик остался неизменным со значением 65362.

А с адреса 7997 начинается подпрограмма, которая и даёт визуальный эффект при вводе команды PAUSE. Выполнив её, Стрелочка подходит к «RET-телепортёру» с тем же самым значением SP-столбика 65362:

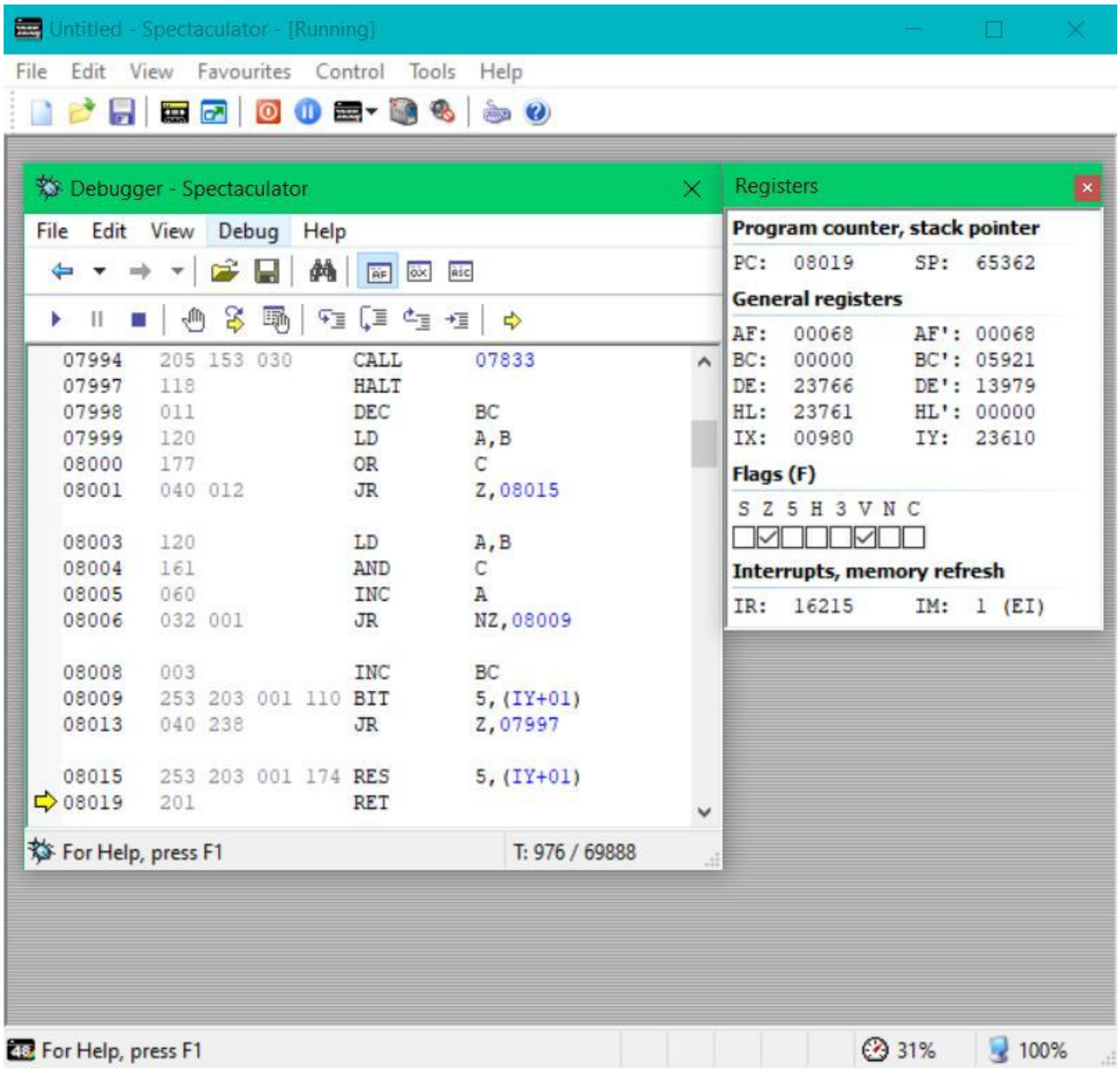


Рис. 254.

Хотите на практике выполнить команду `PAUSE 0`?

Наберите и выполните следующую алгоритмическую программу, а её работу разберу потом:

```
Debugger
Dec
Go To 23613
23613 ← 65364
23610 ← 255
65364 ← 4867
SP ← 65364
BC ← 0
PC ← 7997
Trace
```

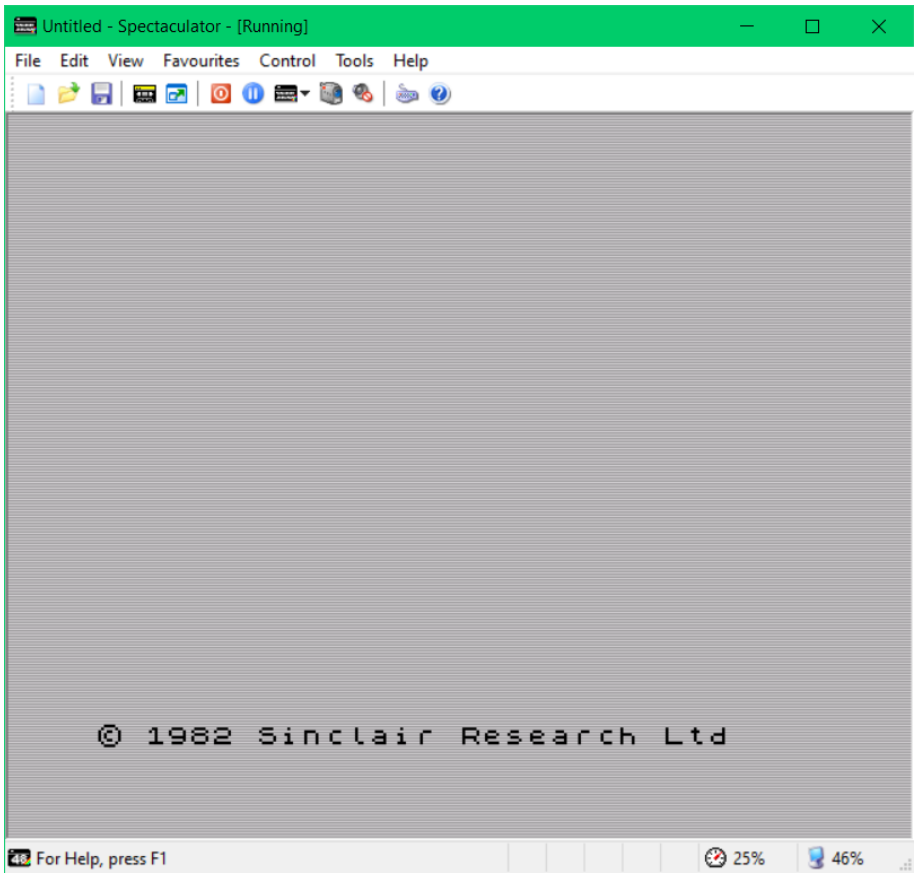


Рис. 255.

Вроде бы ничего не произошло, и Юрец наколол. А теперь нажмите любую клавишу:

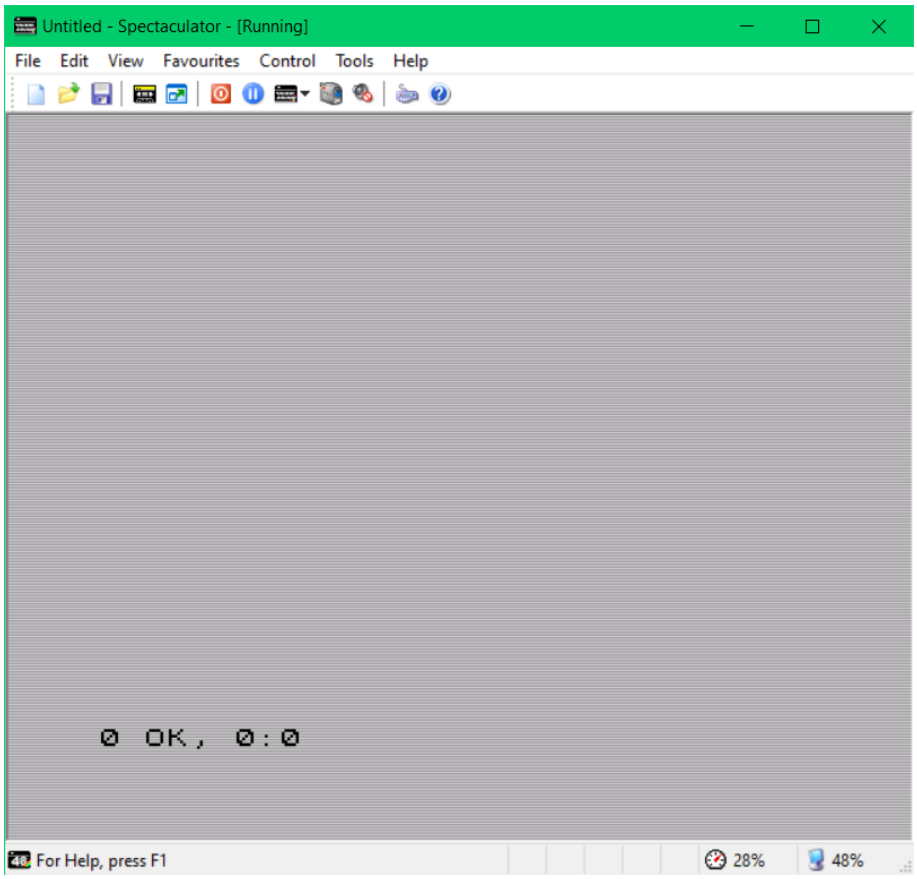


Рис. 256.

А, не, всё ОК, да еще и по нулям. Не даром же книжка называется «BASIC с н00000ля».

А вот теперь можно разбираться, как оно работает. Всем известно, что королевы живут в замках. В нормальном состоянии Стрелочка живёт в своей резиденции EDITOR (3884). Выходя по своим королевским делам, она всегда возвращается назад. Выяснять, из какой именно комнаты роскошного замка, вы её выдернули, долго и муторно, поэтому после выполнения подпрограммы, «RET-телепортёр» по адресу 8019 заряжается адресом дорожки на вход в замок (65364 ← 4867). Сам SP-столбик укорачивается до единственного значения (SP ← 65364). Сверху на крышу столбика кладётся настил (23613 ← 65364).

Опустив прелюдию с формированием числа из введённого символа, в «BC» напрямую задаётся задержка 0. (BC ← 0). Напоследок Стрелочка бережно переносится на дорожку по адресу 7997 (PC ← 7997).

Выполнив программу, Стрелочка встаёт на «RET-телепортёр», который вы заранее зарядили маршрутом в сторону родного замка. Увидев свои владения, Стрелочка выводит сообщение «0 OK, 0:0» (23610 ← 255) и самостоятельно возвращается домой.

Таким образом, можно подсоединить даже две подряд программы, естественно, если второй программе не требуются задавать дополнительные входные данные в окошке «*Registers*».

Следующая программа запускает PAUSE 0. После нажатия клавиши выполняет CLS 3435 и выходит по адресу 4867 для вывода сообщения «0 OK»:

```
Debugger
Dec
Go To 23613
23613 ← 65364
23610 ← 255
65362 ← 3435 4867
SP ← 65362
BC ← 0
PC ← 7997
Trace
```

Приводить картинку с пустым экраном нет смысла. После запуска компьютер замрёт в ожидании, а после нажатия клавиши выведется сообщение «0 OK».

Глава 25
Несимметричные программы

А теперь посмотрим, что такое Несимметричные программы. Это такие программы, в которых значение SP-столбика перед выполнением и после будет разным. В зависимости от того, выше или ниже первоначальной высоты станет столбик, программы можно разделить на «Положительно несимметричные» и «Отрицательно несимметричные»:

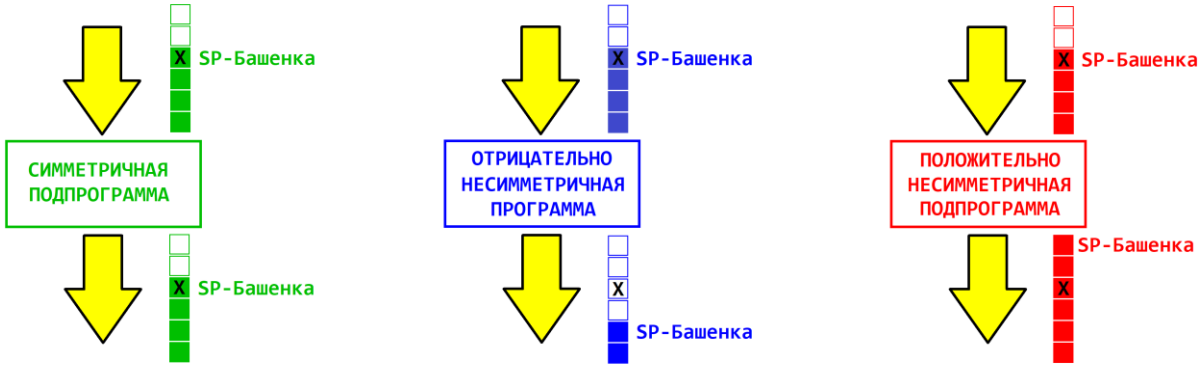


Рис. 257.

В принципе любая сложная симметричная программа может стать несимметричной, если запускать её определённые фрагменты:

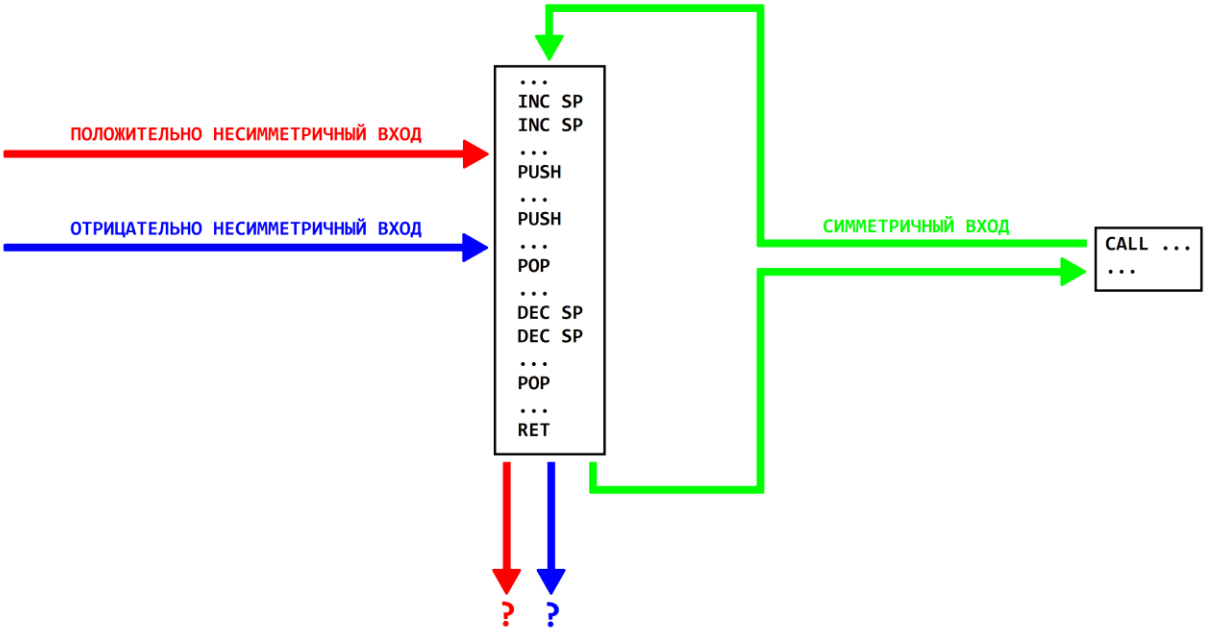


Рис. 258.

Предлагаю на практике запустить несимметричный фрагмент программы 6700-6727, который имитирует вывод номера «символьно-числовой» строки:

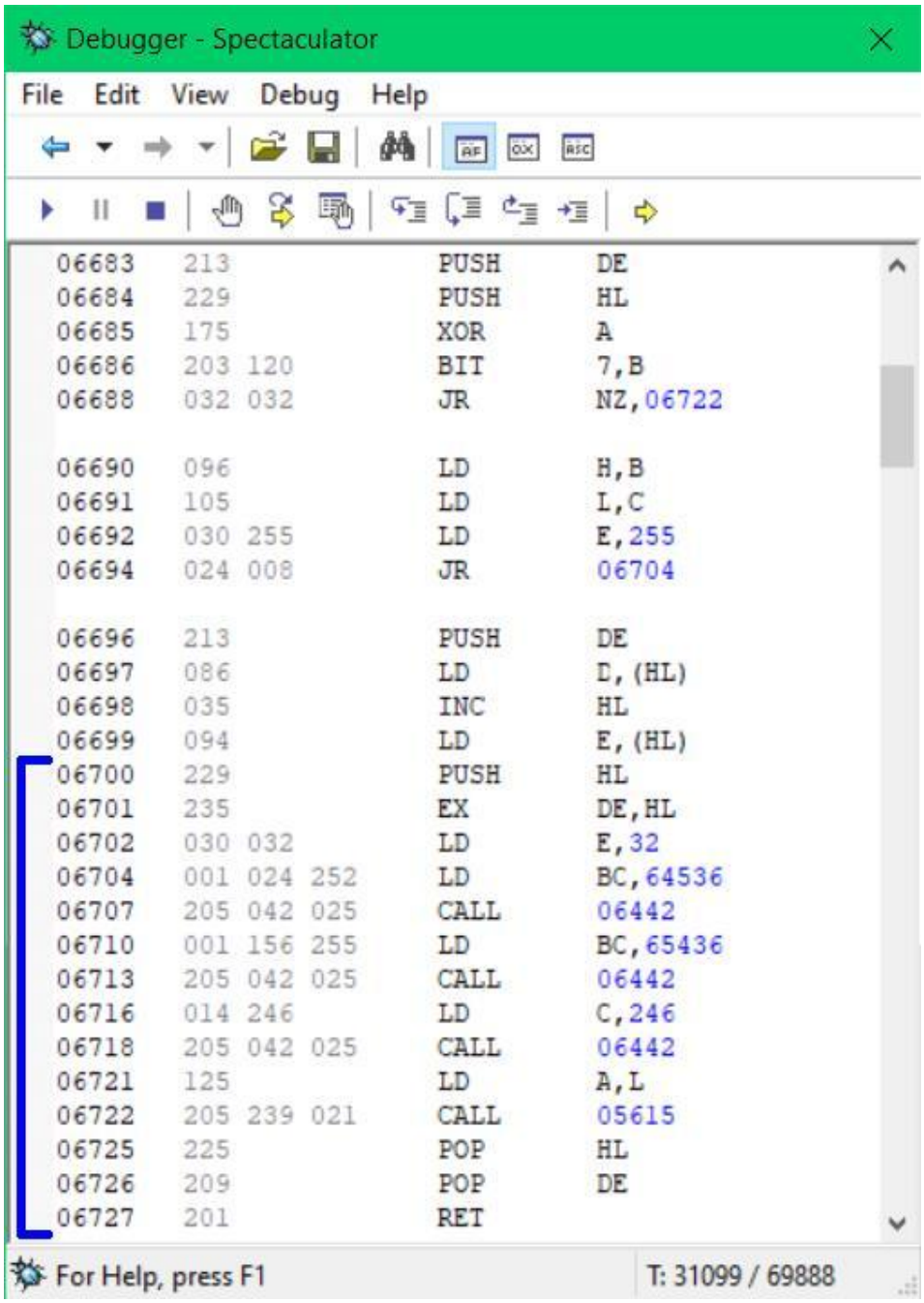


Рис. 259.

Итак, это будет отрицательно несимметричный фрагмент программы, поэтому в SP-башенке придётся оставить пару холостых значений, после которых добавить адрес возврата в резервные ворота «BASIC города» (4777).

Наберите и введите следующую программу:

```
Debugger
Dec
Go To 23613
23613 ← 65364
23612 ← 32
65364 ← 4777
SP ← 65362
DE ← 10000
PC ← 6700
Trace
```

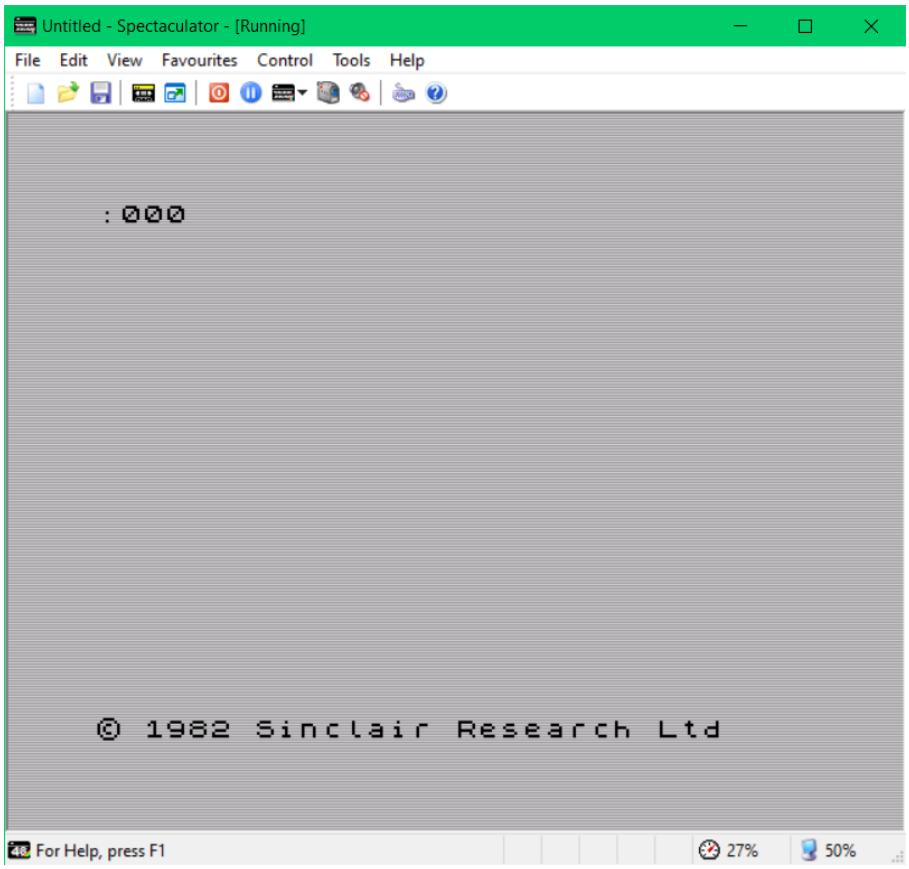


Рис. 260.

Не повредив заставку, на экран вывелось число 10000 в «символьно-числовом» формате.

Глава 26
Вывод Стрелочки из зависаний и защит в BASIC систему

Краткое содержание: возврат из зависаний, искусственный возврат в BASIC, LINE-ADR, NEXT-ONE

Представьте себе такую картину. Вы в поте лица старались и набрали вот эту сложную программу:

```
10 PRINT "СТРОКЕ ТЕКСТА НАСТАНЕТ ХАНА"
20 RESTORE 65000
```

Можно не только представить, но и ввести с помощью следующего «*God Mode*» алгоритма:

```
Debugger
Dec
Go To 23560
23560 ← 13
23611 ← 32
23627 ← 23807
23641 ← 23808
23649 ← 23810 23810 23810
23755 ← 0 10 31 0 245
23760 ←""СТРОКЕ ТЕКСТА НАСТАНЕТ ХАНА
23788 ← 34 13 0 20 13 0 229 54 53 48 48 48 14
23801 ← 0 0 232 253 0 13 128 13 128
Trace
```

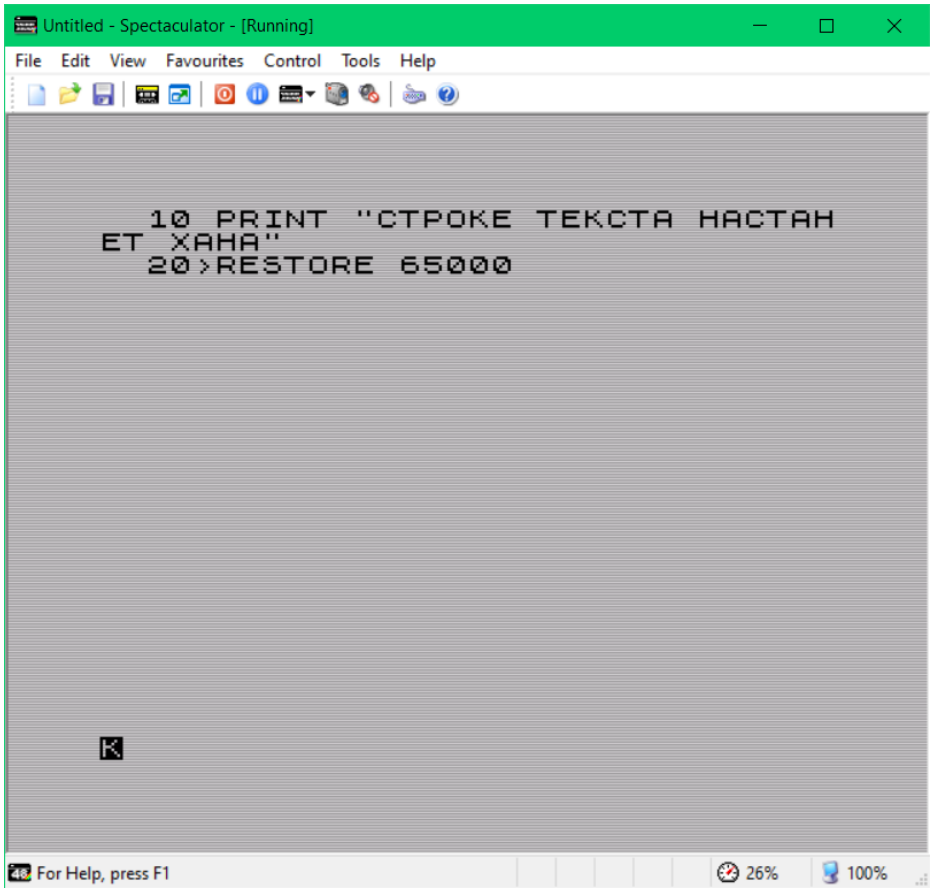


Рис. 261.

Набрав RUN, вы радостно запускаете программу ENTER'ом:

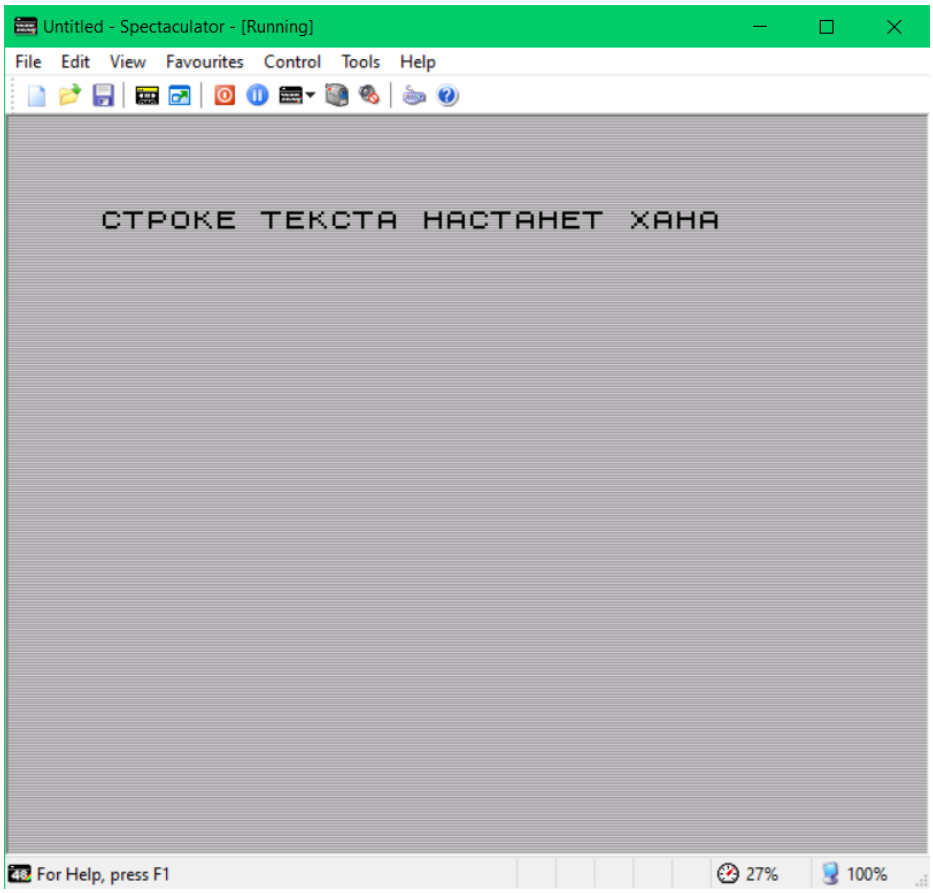


Рис. 262.

Хренак! Ничего не нажимается. Всё! Писец вашей ценной программе. Что раньше делали в этом случае? Горевали и нажимали сброс. А что можно сделать в 2024 году?

Вывести заблудившуюся Стрелочку из ступора 🤪.

Для оценки ситуации откройте «*Debugger*»:

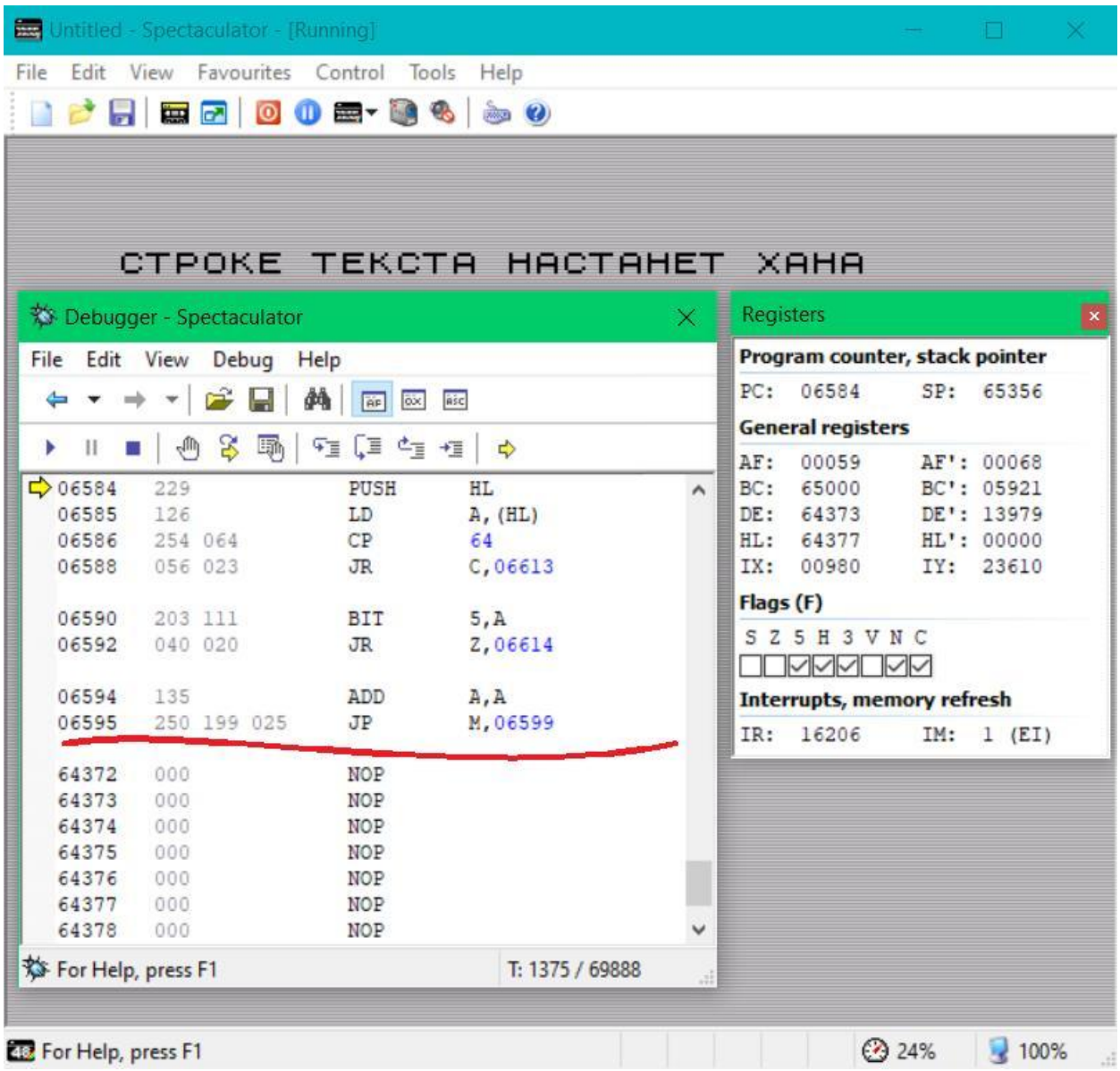


Рис. 263.

О! Старая знакомая связка `LINE ADDR - NEXT ONE`! При попытке вычислить длину строки из пустых ячеек памяти, Стрелочка блуждает в интервале адресов 6516-6628 и никак не может выбраться.

Как помочь бедняжке в данном конкретном случае понятно. Нужно поймать её на адресе 6520 малиновой 🔸 точкой, снять C ☐ галочку «C», из-за которой эти проблемы начались, а потом отпустить по своим делам.

Чтобы помочь Стрелочке выбраться, выполните следующий алгоритм:

```
Debugger
Go To 6520
Add Breakpoint 6520
Trace
```

Remove Breakpoint 6520
AF ← AF-1

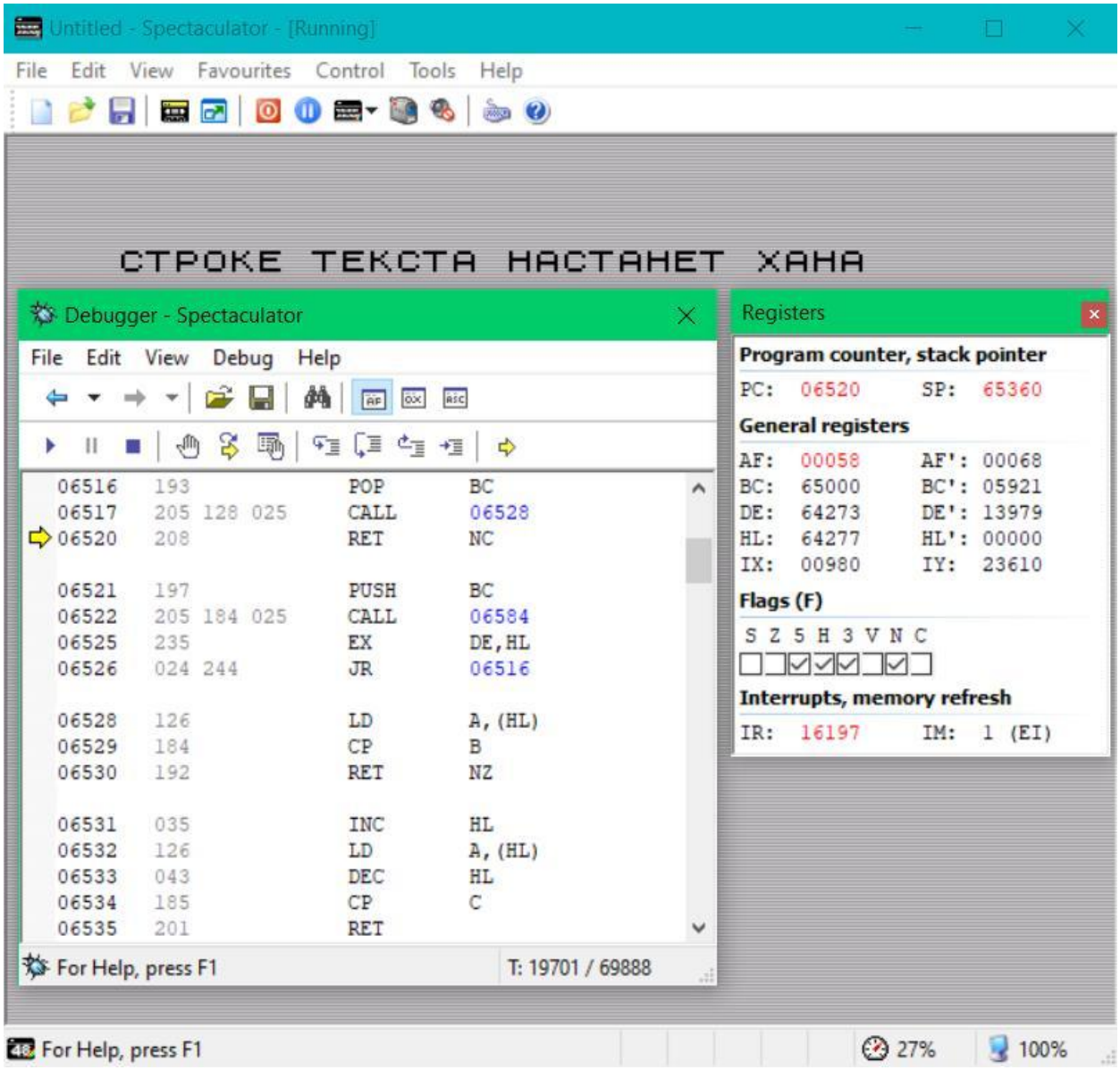


Рис. 264.

А теперь нажимайте «*Trace (F5)*»:

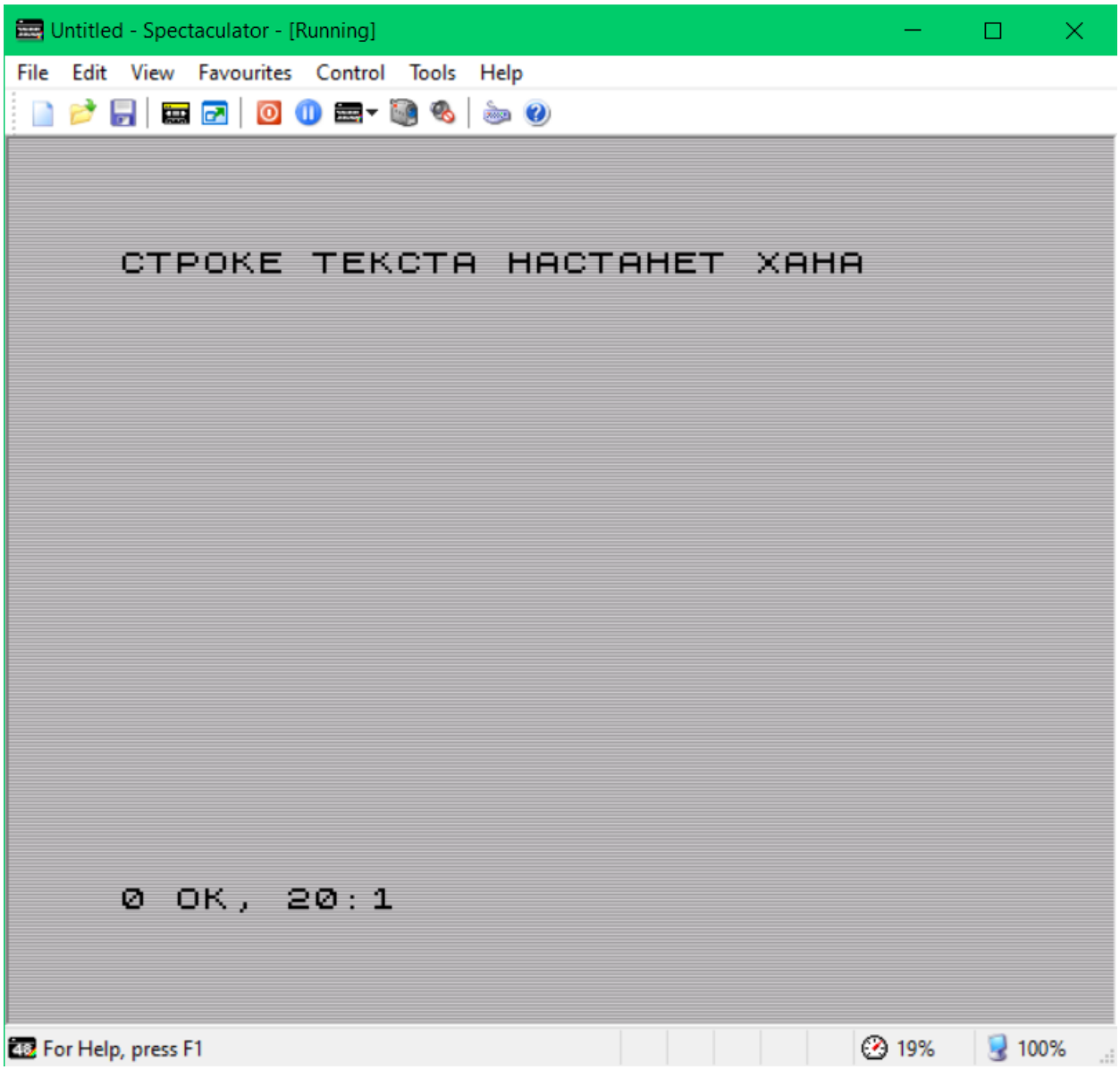


Рис. 265.

И зависания как не бывало. Но радоваться рано. Зависания бывают разной степени сложности. В большинстве ситуаций без поллитры Колы не разобраться, куда нужно выходить. Например, в запущенных играх, где не предназначен возврат в BASIC или в загружаемых играх с «защитой», где явно слышен звук BASIC блока, а разбираться, как этот хлам действует, лениво. Для того, чтобы не тратить время на распутывание этой хрени, предлагаю универсальный алгоритм на языке «*God Mode*» и он даже знаком вам по предыдущим главам.

Снова запустите программу, чтобы она зависла, набрав RUN и ENTER натуральным методом, а следом введите алгоритм:

```
Debugger
Dec
Go To 23613
23613 ← 65364
SP ← 65366
PC ← 4770
Trace
```

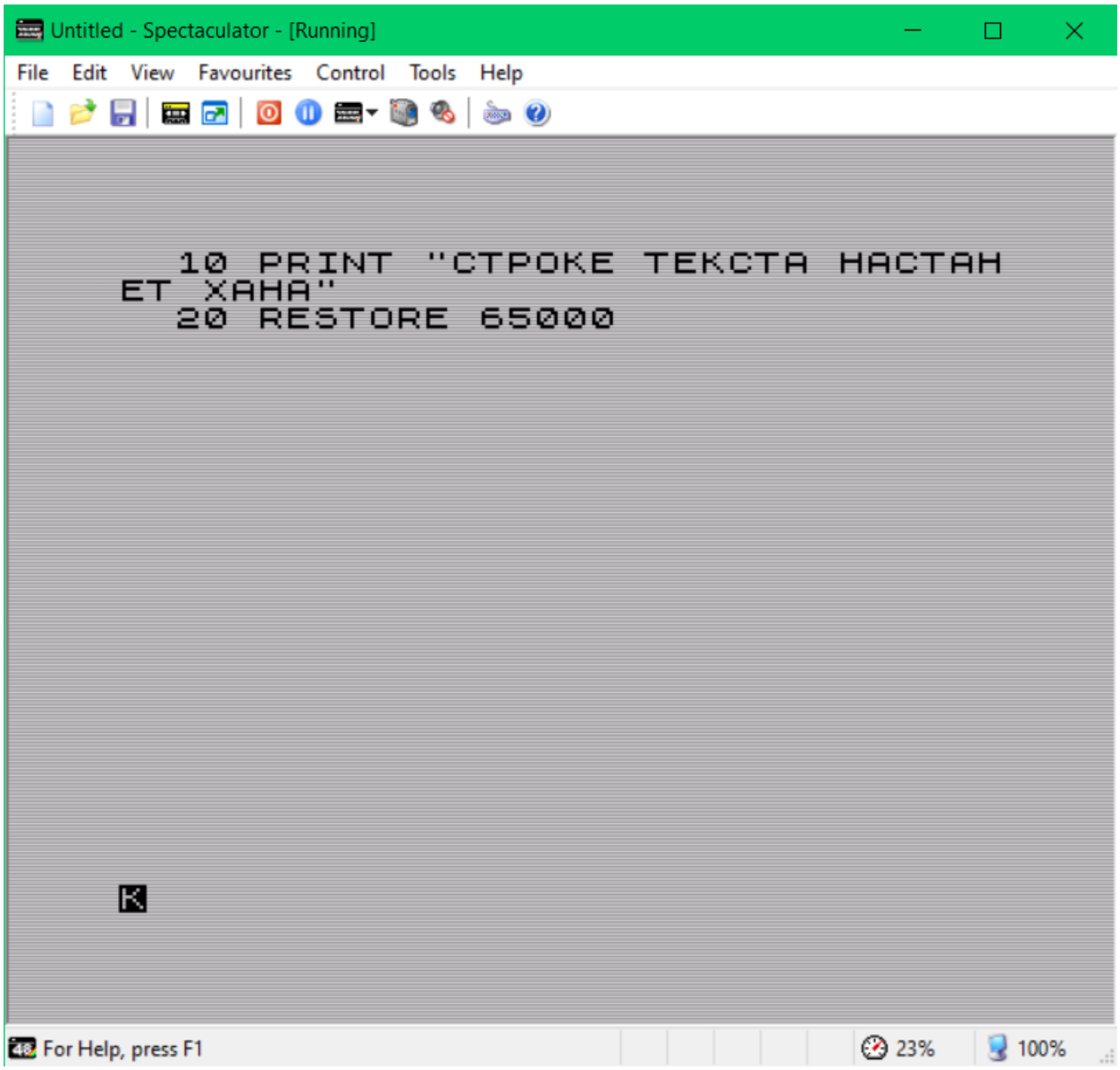


Рис. 266.

На этот раз программа не только остановилась, но и вывелась на экран. И снова всё сработало. Ну и еще одна разновидность метода вывода из зависания без обновления изображения на экране:

```
Debugger
Dec
Go To 23613
23613 ← 65364
SP ← 65366
PC ← 4764
Trace
```

Глава 27
Советы Неэкспертов: синтез BASIC области для остановки игр

Краткое содержание: быстрая остановка защищённых игр, возврат в BASIC

А теперь, после продуктивной работы, предлагаю сделать передышку и поговорить об играх. Да именно о них! И не только поговорить, но и применить полученные практические знания предыдущей главы. Распечатаю-ка я рубрику «Советы Неэкспертов».

Первой в центре внимания будет виновница торжества – игра «Animated Strip Poker». Именно благодаря ней и родилась данная книга. Именно из-за желания вспомнить, отчаянные попытки обойти неприступную защиту этой игры в начале 1993 года. На этот раз ничего не набирайте, а просто смотрите. Глава теоретическая, в которой важно уловить принцип.

После ввода `LOAD ""ENTER` загружается BASIC блок, в котором невидимая строка, куча управляющих символов и прочей хрени. После короткого блока «`Bytes:`» начинается загрузка блока без заголовка на всю имеющуюся память. Вот на экран вывелась картинка. Следом, с характерным звуком, загружаются системные переменные, а затем начинается длинная и сочная BASIC программа. Звук загрузки BASIC строк трудно перепутать с чем-либо:

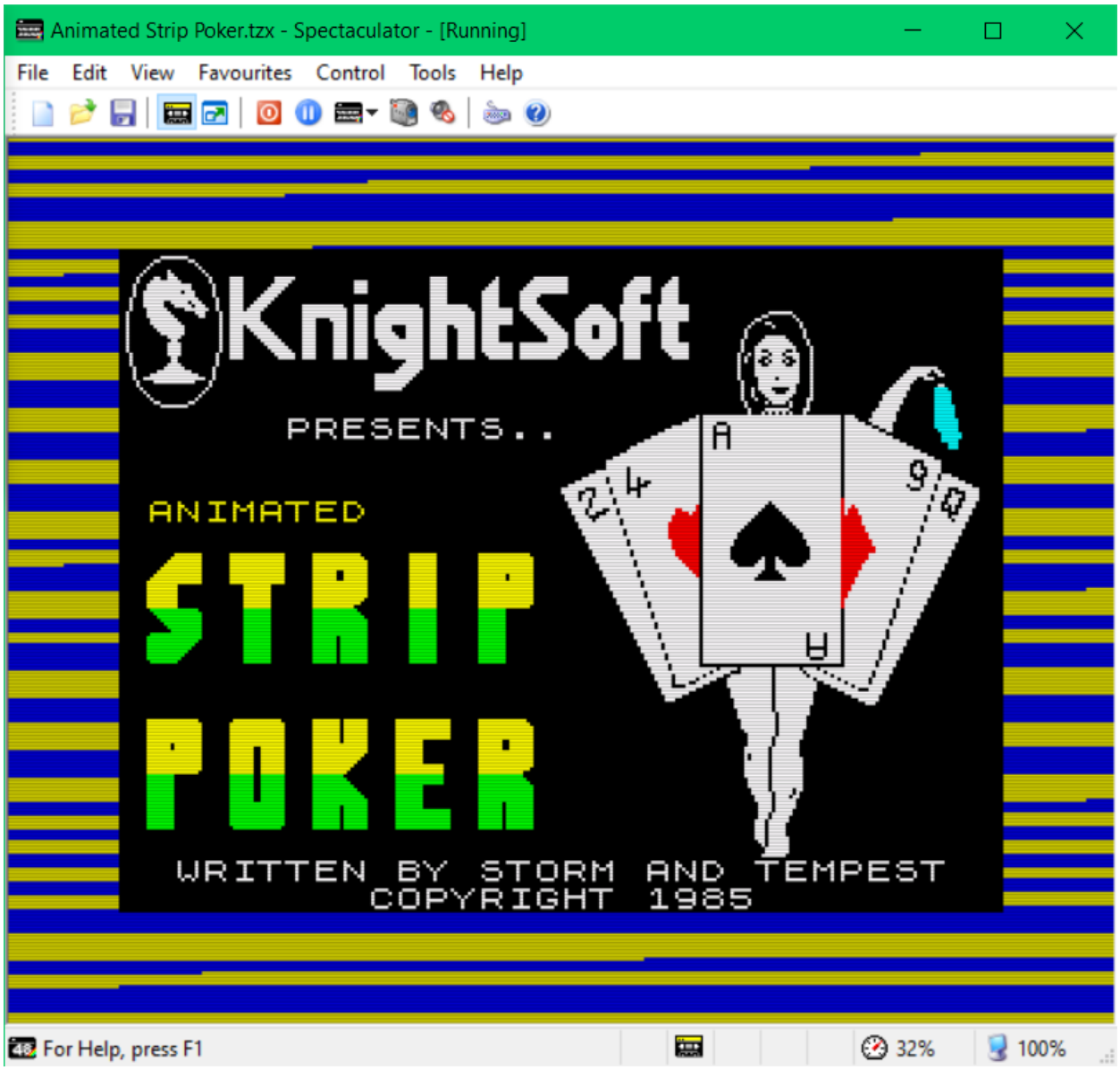


Рис. 267. Animated Strip Poker. Процесс загрузки оригинальной игры.

После BASIC блока с какими-то скрипами загружаются интересные данные. Настал момент, когда программа загрузилась и запустилась. После небольшого музыкального вступления началась игра:

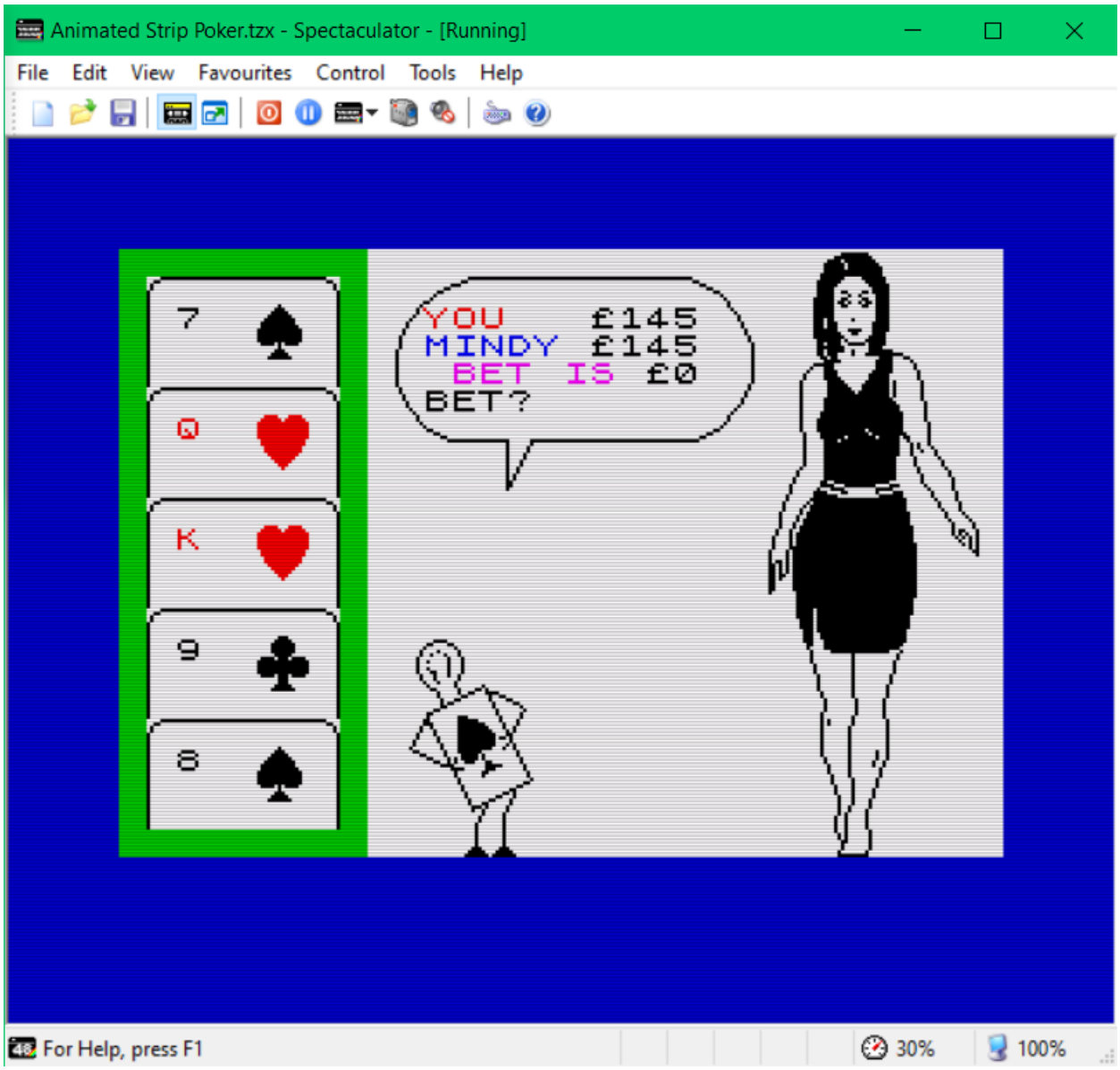


Рис. 268. Animated Strip Poker. Игра в процессе.

Время открыть «*Debugger*» и ознакомиться с происходящей ситуацией. Первым делом бросается в глаза SP-указатель чуть ниже сороковника. Ага! В BASIC блоке стопудово был «`CLEAR`». А вот и фундамент, с характерной последовательностью «0, 62». Так и есть, при загрузке выполнился `CLEAR 42744`:

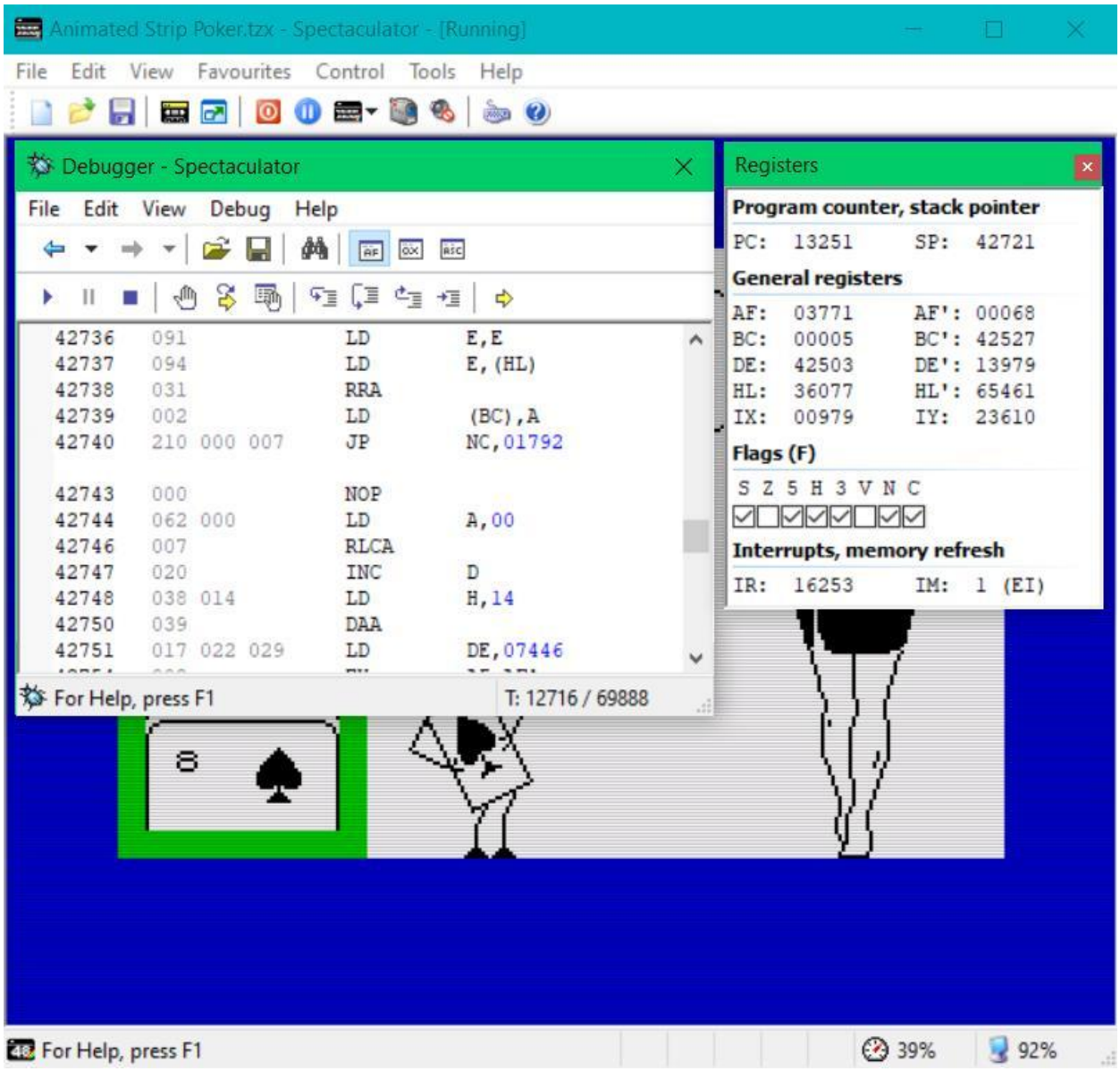


Рис. 269. Animated Strip Poker. Поиск SP столбика.

Пользуясь волшебной программой из прошлой главы, создаётся алгоритм остановки игры с запуском BASIC. Указатель «SP» нужно сделать «CLEAR-1», подкладку «CLEAR-3» и вперёд в `MAIN EXECUTION` по адресу 4770.

Для конкретной игры, алгоритм принудительного выхода в BASIC, будет таким:

```
Debugger
Dec
Go To 23613
23613 ← 42741
SP ← 42743
PC ← 4770
Trace
```

Набираю алгоритм:

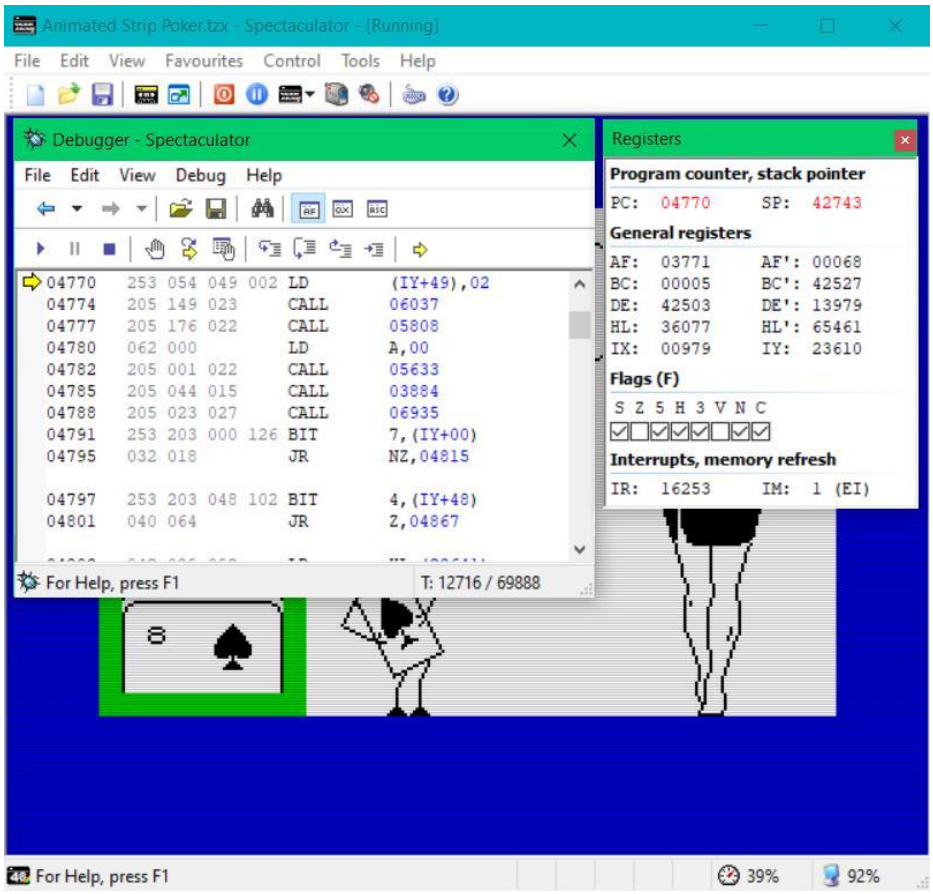


Рис. 270.

После запуска через «*Trace*»…

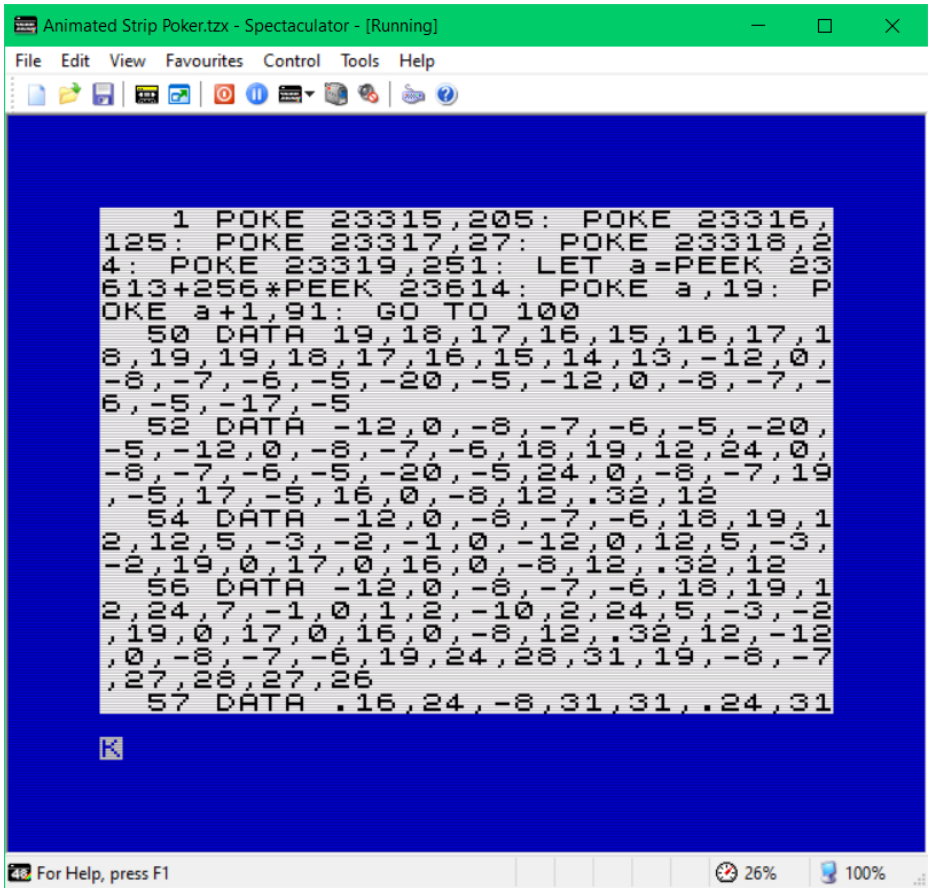


Рис. 271. Animated Strip Poker. Вскрывшаяся BASIC программа.

Игра со всеми защитами мгновенно прервалась, запустился BASIC, и на монитор выдалось 27,5 экранов текста программы. Ну что, можно лететь на машине времени в начало 1993 года и дарить себе секрет этой игры. Уверен, счастью не будет предела.

Таким образом можно не только реанимировать программы, но и принудительно выводить игры в BASIC, даже те, которые загружались нестандартными загрузчиками со множеством «защит». В большинстве случаев, достаточно найти фундамент SP-столбика.

Это был самый лёгкий тип программ. Следующий тип игр, которые сбивают чуть больше параметров, для нормального функционирования BASIC системы, но при этом самые важные системные переменные остаются неповреждёнными.

Итак, еще одна моя любимая игра «TETЯIS-2». Известно аж пять версий данной игры. Все они зашифрованы, и часть нестандартно загружается:

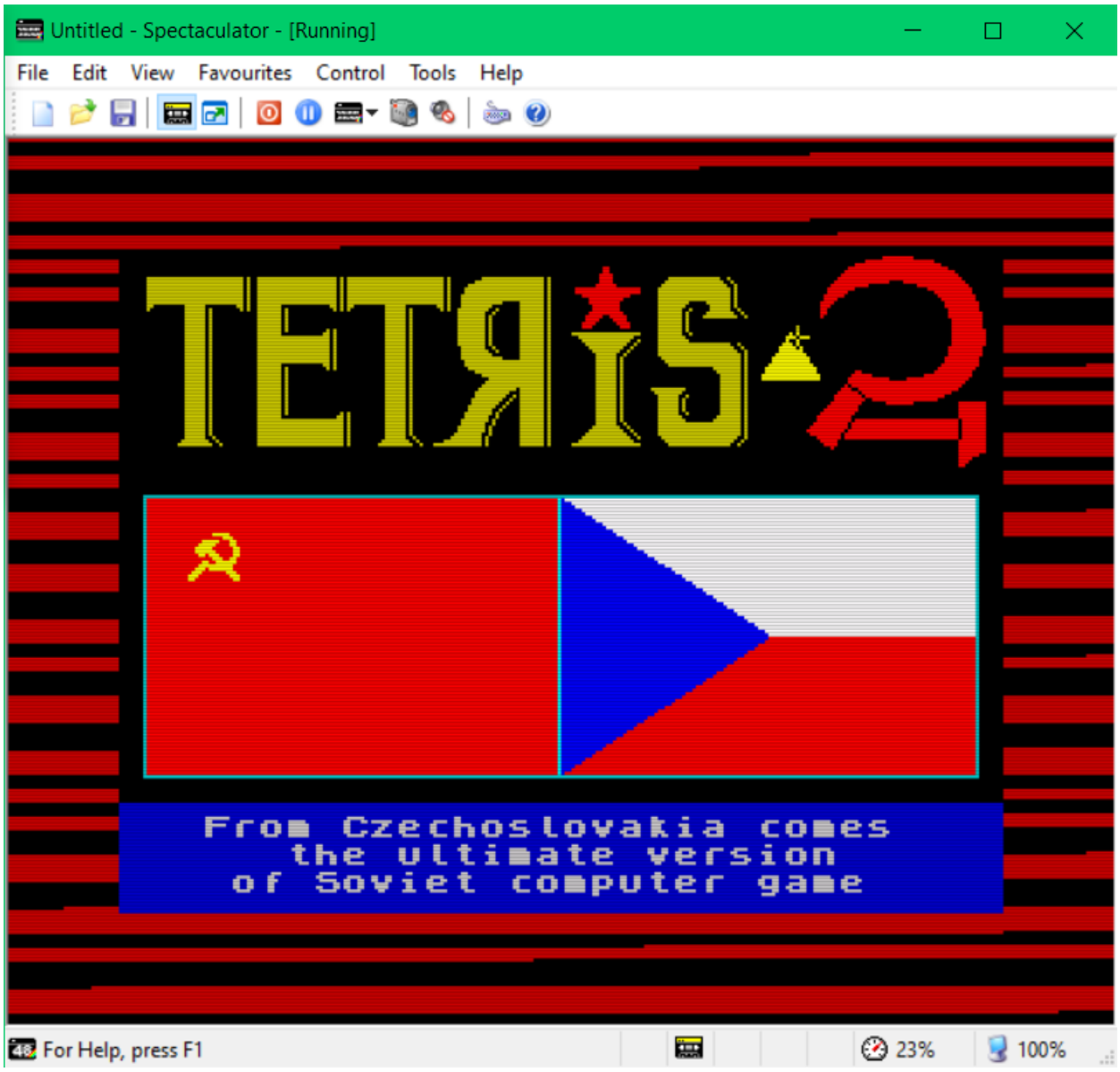


Рис. 272. Tetris-2. Процесс нестандартной загрузки.

Как говорится, да хоть обшифруйтесь и загружайте симфонией Моцарта. Я просто подожду окончание загрузки распаковки алгоритма игры, а потом сделаю всё, что пожелаю:

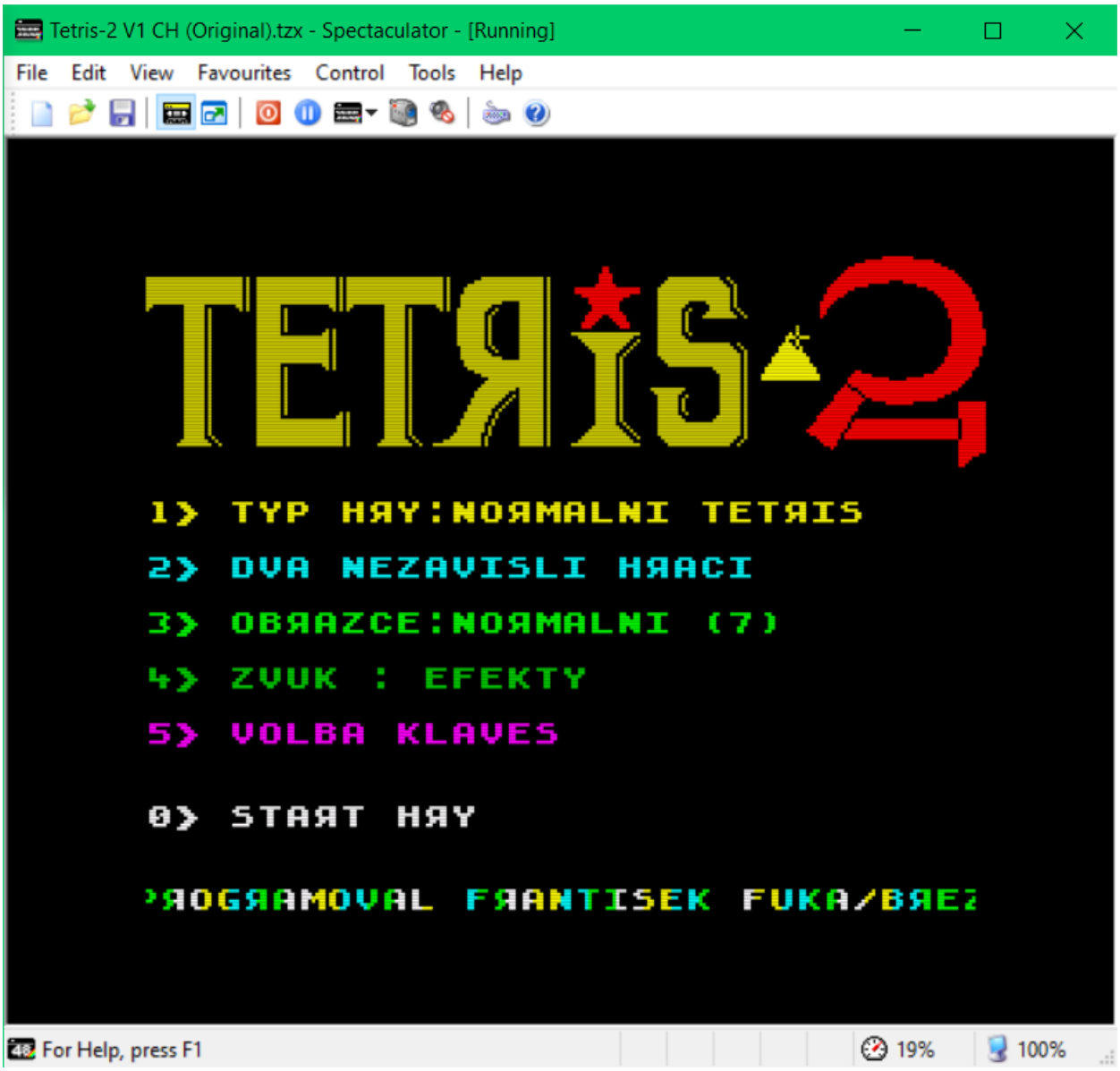


Рис. 273.

Игра запустилась, и на экране появилось меню. Можно открывать отладчик:

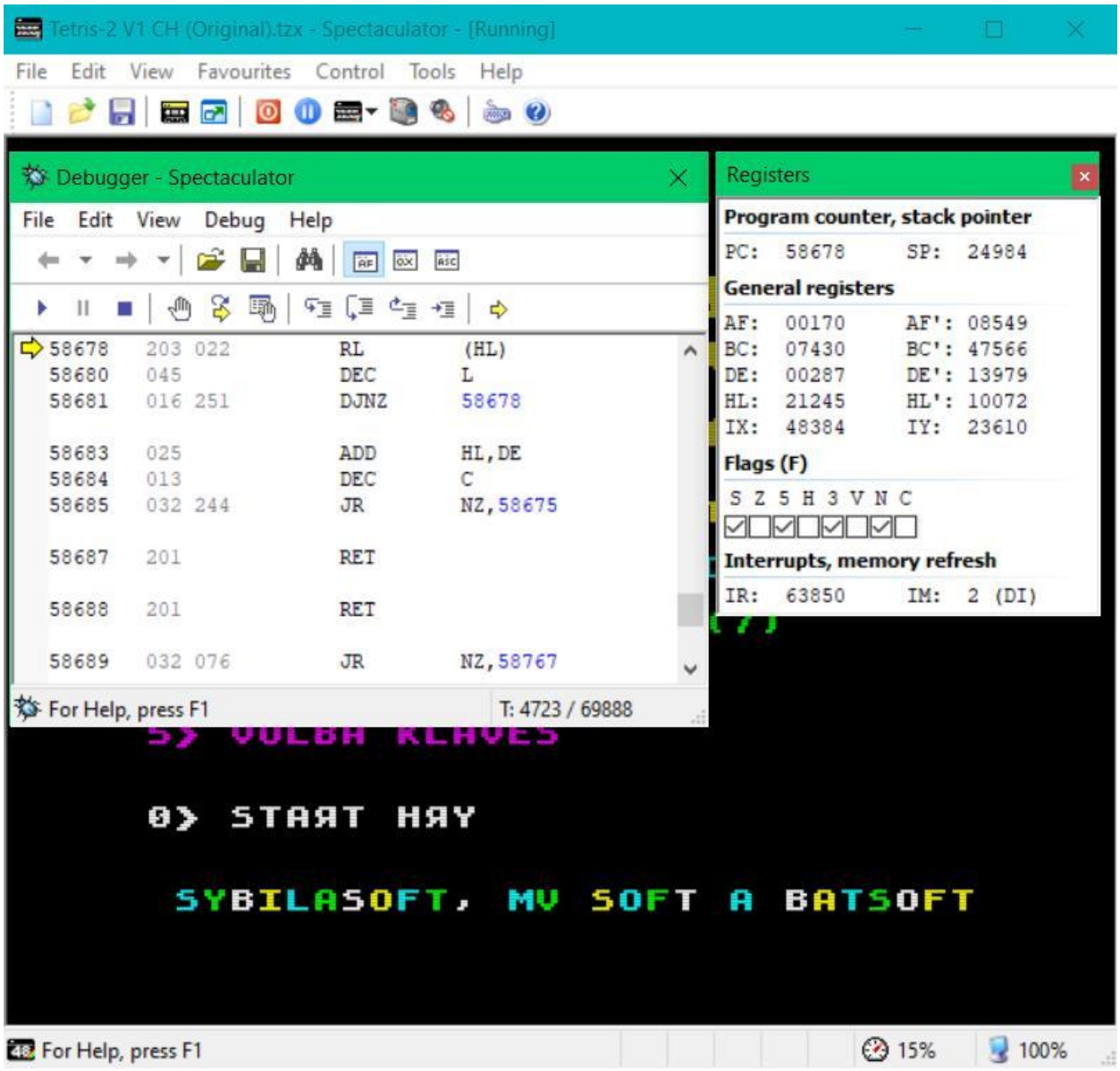


Рис. 274.

А тут «SP» задран аж на 24984. Рождённый самопальными защитными страшилками, он не имеет основания «62» для `GO SUB` и это логично. Разработчики не думали, что в далёком будущем, внешними методами, кто-то сможет выйти в BASIC прямо из игры. На тот момент это была научная фантастика.

Можно обойтись без фундамента. Для предварительного ознакомления это и не нужно. Как и в прошлом случае, установив «SP» на выбранное место, поверх кладётся «покрывало», путём установки в `ERR_SP (23613)` значение SP-2. Но как видно, «SP» это мелочи. Игра запустилась, и Spectaculator находится режиме «`IM 2`» да еще «`DI`». При таком раскладе клавиатура работать не будет.

Проблема тут только одна: разработчики Spectaculator не предусмотрели замену режимов `IM` с `DI/EI` напрямую в окошечках. Так что придётся либо добавить пару строчек в машинном коде, либо действовать методом точек прерывания. Более подходящим по теме книги будет второй вариант.

Тогда алгоритм остановки для всех версий игры «ТЕТЯIS-2» будет таким:

```
Debugger
Dec
PC ← 4626
Add Breakpoint 4661
```

```
HL ← 25000
Trace
Remove Breakpoint 4661
PC ← 4770
Trace
```

Осталось выполнить эту нехитрую универсальную программу:

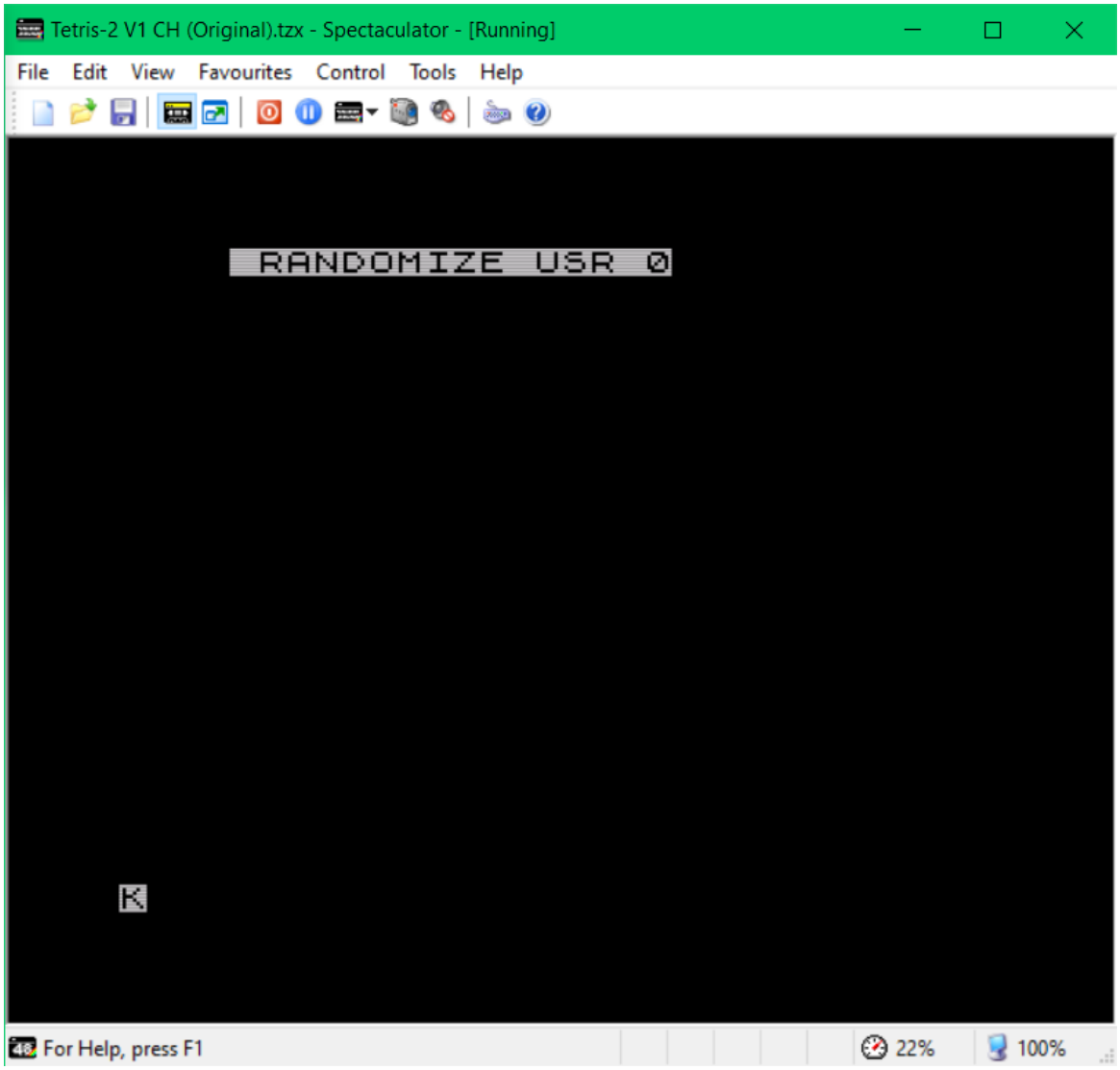


Рис. 275.

Вот и всё. 35-40 секунд работы и все эти потуги XOR-шифровки программы с компрессией, да спецзагрузчиком идут лесом. Игра встала, а вместо нее запустился BASIC. Ну и попутно угадывается точка запуска игры – 57472. Вернув экрану белый цвет (23693 ← 56) цвет, сверху появляется строка `703 RANDOMIZE USR 0`.

Как вы понимаете, это тоже не 100% способ «запуска BASIC». Существуют игры самой жестяной категории, где переписывается все возможные 49152 ячейки вместе с системными переменными. На такие хитропопые приёмы у меня заготовлено ещё более простое решение, которое вообще останавливает 99,9% защищенных и многократно зашифрованных игр с любого момента времени:

```
23617 ← 0
23665 ← 0
PC ← 4626
HL ← [адрес для размещения SP]
```

Как оно работает: при разрушении области системных переменных Стрелочку надо направить на отметку 4626 в самую середину подпрограммы сброса, но после очистки всей памяти. Таким образом, восстанавливается почти все критические системные переменные, но только те, которые должны иметь числовые значения, отличные от нуля. Понадеявшись на «сброс-перфоманс» важные ячейки, в которых для нормальной работы должен быть «0», не обновляются. Где-то это не так критично, а вот если мусор попадает в переменную `FLAGX (23665)` и активирует бит №5, строке ввода настанет полная задница. Именно поэтому к базовому восстановлению массива, необходимо добавить обнуление этой ячейки. Дополнительно не помешает выйти из расширенного и графического режимов, если они вдруг самоустановились, и на экране появляются всякие курсоры «RUN». Переменную `MODE (23617)` и её свойства изучали в главе «Удивительный режим MODE».

В качестве примера, рассмотрю еще одну игру детства - «Deflektor». Я не помню, какой именно был вариант в 1992 году, поэтому взял фирменный оригинал, который лежит на сайтах. Он со спецзагрузчиком, серийным номером, несколькими слоями шифра, самомодификациями, перетаскиваниями данных, и прочими наворотами:

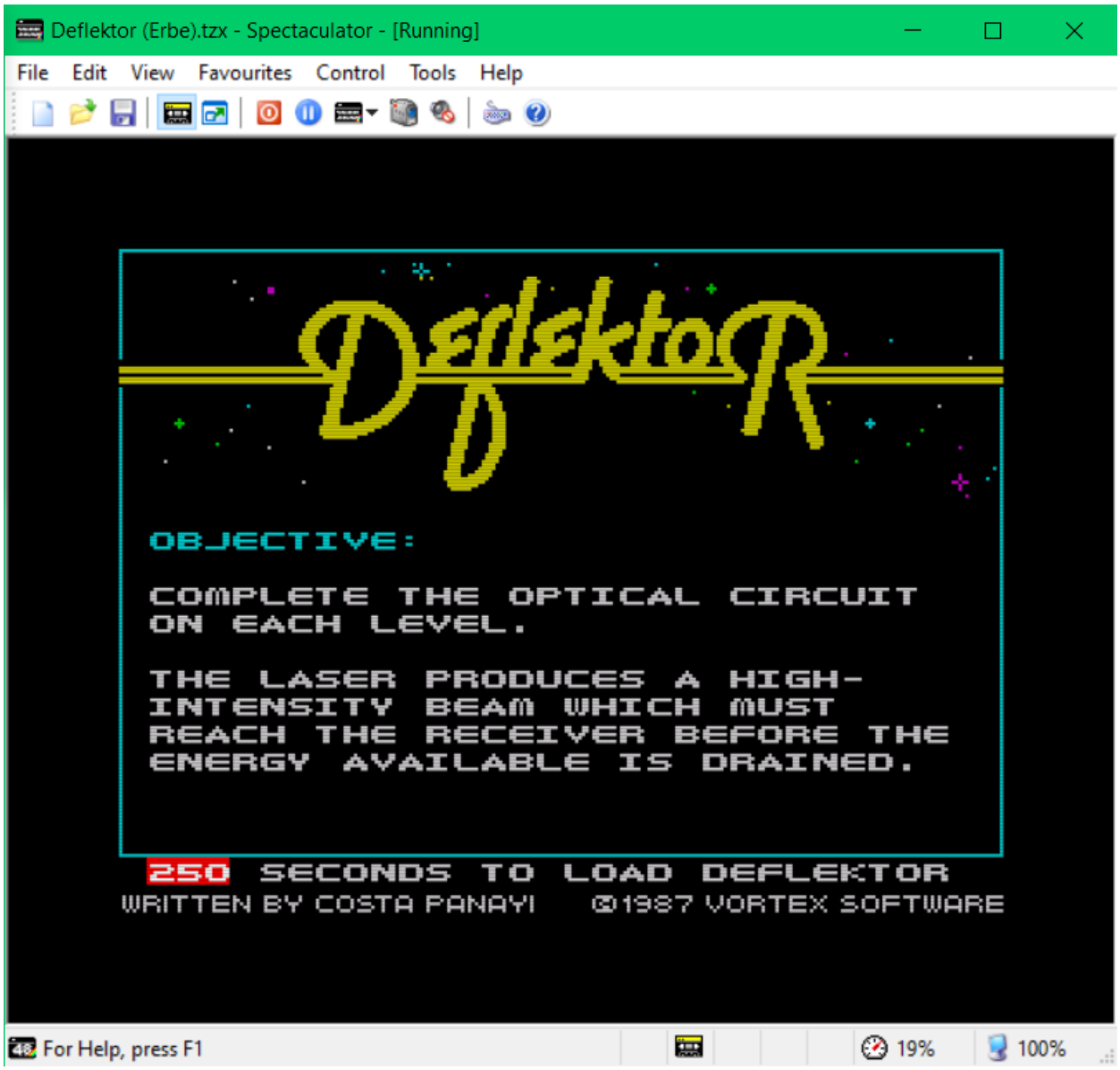


Рис. 276. Процесс загрузки фирменной версии игры «Deflektor».

Игра загрузилась, расшифровалась, распаковалась и заиграла скрипящая музыка:

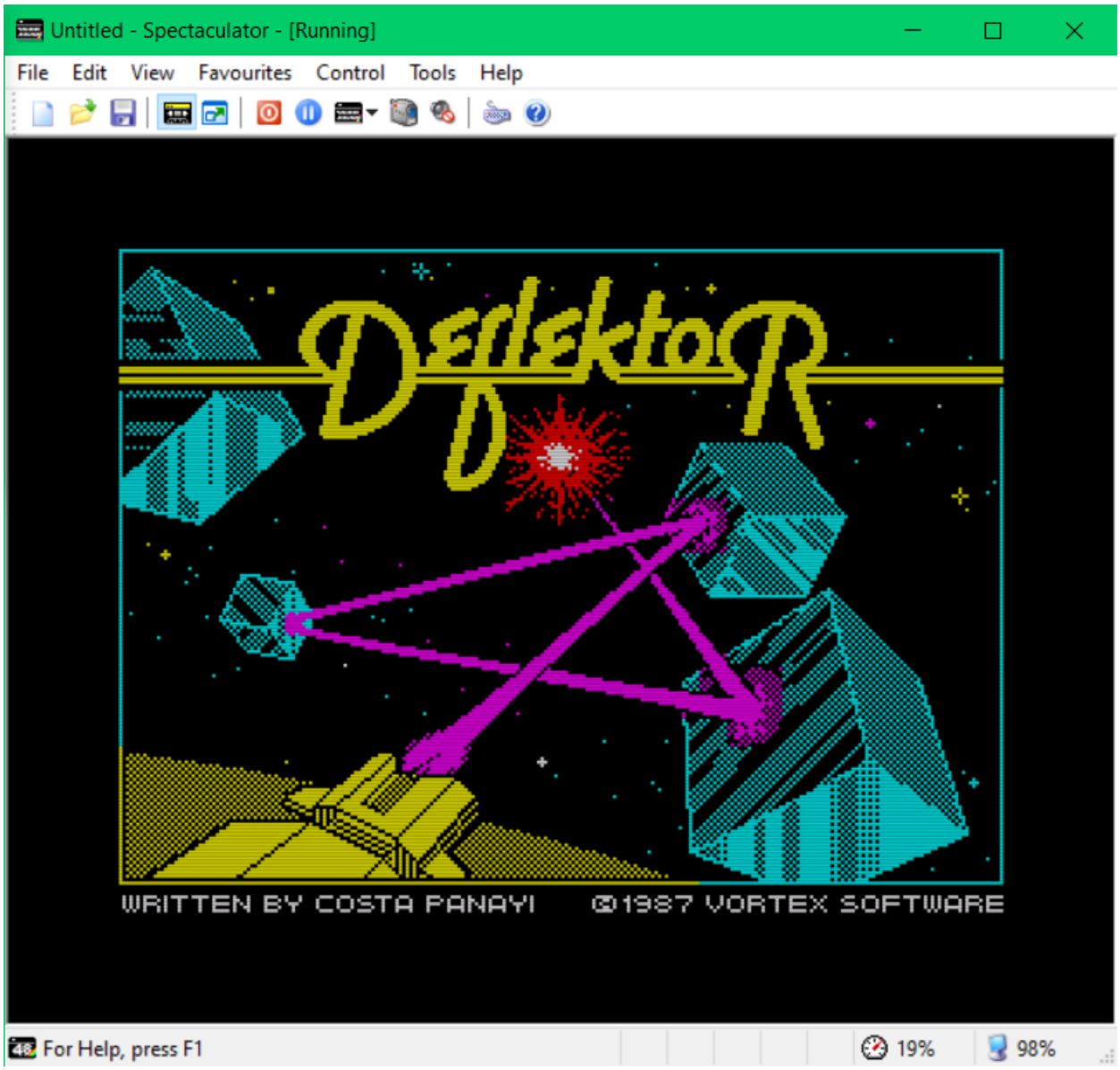


Рис. 277.

Осталось открыть отладчик и выполнить следующий алгоритм:

```
Debugger
Dec
Go To 23617
23617 ← 0
23665 ← 0
PC ← 4626
HL ← 65367
Trace
```

На экране появилась заставка;

© 1982 Sinclair Research Ltd.

После нажатия любой клавиши замигал курсор. Не надейтесь найти BASIC, его там и не было, а вот всем защитам мгновенно настал писец. Удивительно, правда? Чем сложнее загрузчики и защита, тем проще алгоритм принудительной остановки.

Есть у этого метода и недостаток. Данные игры, которые находились в области (23552-23758), естественно будут затёрты. Если это графика, то не так страшно. В такой ситуации, повреждённый системными переменными кусок данных, можно легко вклеить с помощью HexEditor или модифицировать игру, упихав фрагмент в другое свободное

место. А вот если особо одарённый автор засандалит в эти адреса логику, то будет в разы печальнее. Делать реверс-инжиниринг и пересобирать большую фирменную игру дело не из приятных.

В данной игре, ничего не повредилось, потому что в системных переменных остался мусор от работы нестандартного загрузчика и шифровальщика данных, поэтому я и взял её в качестве примера. Точка входа в программу 33800. Ввод `RANDOMIZE USR 33800` покажет полностью работоспособность программы.

Я понимаю, что такой подход в нынешней ситуации выглядит так, если бы я бульдозером сносил старый деревенский толчок, но мне очень хотелось «отомстить» за то, что в детстве все эти «супер защиты» не позволяли вскрыть любимые игры и попробовать создать собственные уровни.

На этом игровую передышку заканчиваю.

Глава 28
Дополнительные сведения о PRINT'осодержащих подпрограммах

Краткое содержание: RST 16, PR-STRING (8252), PO-MSG (3082), вывод символов

Есть такая глава в самоучителях по BASIC. Если честно, то в этой книге она и не планировалась, но в дело вмешалась случайность. Представьте себе такую ситуацию. Набираете вы программу и ошибаетесь в одной циферке – хренак и всё повисло. Это как кусок булки, который норовит упасть маслом вниз. При разработке примера для будущих глав я случайно промахнулся при копировании и вкатил число в соседнюю ячейку области системных переменных. После запуска случилось чудо. Произошла полезная ошибка и на экране появились непредсказуемые эффекты, которые позволили оптимизировать код примеров, которые очень понадобятся в следующих планируемых главах. Так родилась эта спонтанная пародийная глава.

По аналогии с ассемблером, известно как минимум три метода вывода текста на экран алгоритмами «*God Mode*» без использования точек ● прерывания.

Самым простейший будет метод побуквенного вывода с помощью подпрограммы 16, а точнее 5618, который упоминают в любом самоучителе по ассемблеру или машинным кодам

Введите и запустите следующую программу.

```
Debugger
Dec
Go To 23613
23613 ← 65364
23612 ← 32
65364 ← 4777
AF ← 22528
SP ← 65364
PC ← 16 или 5618
Trace
```

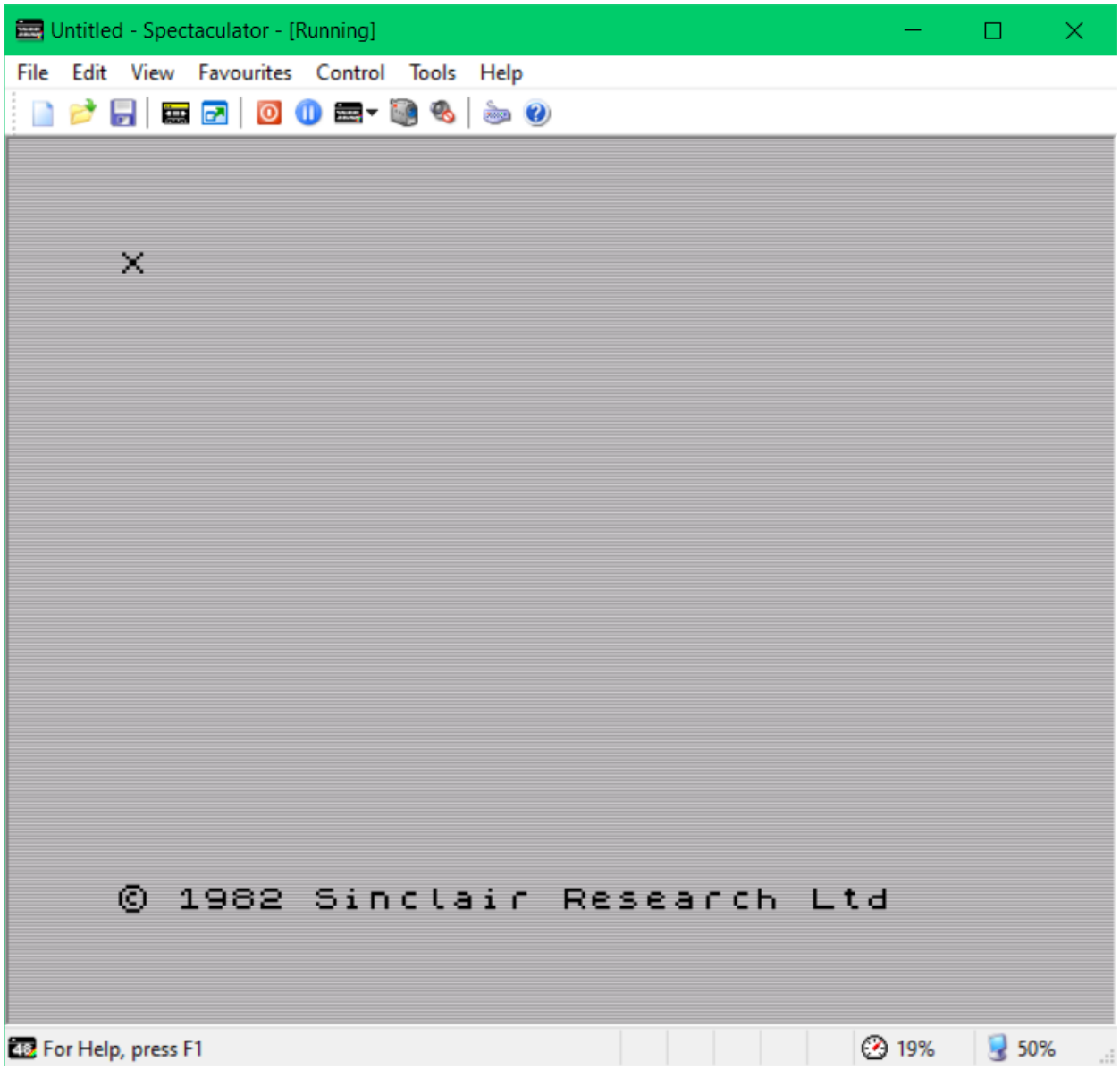


Рис. 278. Вывод символа X на экран с помощью подпрограммы 5618.

Программа является аналогом `PRINT "X"` на BASIC или на «*EmuZWin*» ассемблере:

```
LD (IY+2), 32
LD A, 88
JP 5618
```

У этого метода есть недостаток. За раз можно напечатать только один символ. Именно поэтому я и выбрал символ «X», потому как выводить большие тексты таким методом **X**РЕНОВО.

Не могу найти, но попадалась забавная статья в каком-то из «ZX-Ревю». Один из читателей прислал программу, кажется, для взлома игры. Весь текст выводился этим методом с помощью `RST 16` (`RST 10`). Там автора еще «Инфорком» потроллил немножко. Естественно, этого слова еще не существовало, но смысл именно такой.

Следующий алгоритм имитирует строку одноцветного текста с помощью подпрограммы вывода служебных сообщений 3082:

```
Debugger
Dec
Go To 23296
```

```
23296 ← 128
23297 ←ВЕТЕР ВОЕТ В СТАРОМ ОКН
23320 ← 197
23612 ← 32
23613 ← 65364
65364 ← 4777
AF ← 0
DE ← 23296
SP ← 65364
PC ← 3082
Trace
```

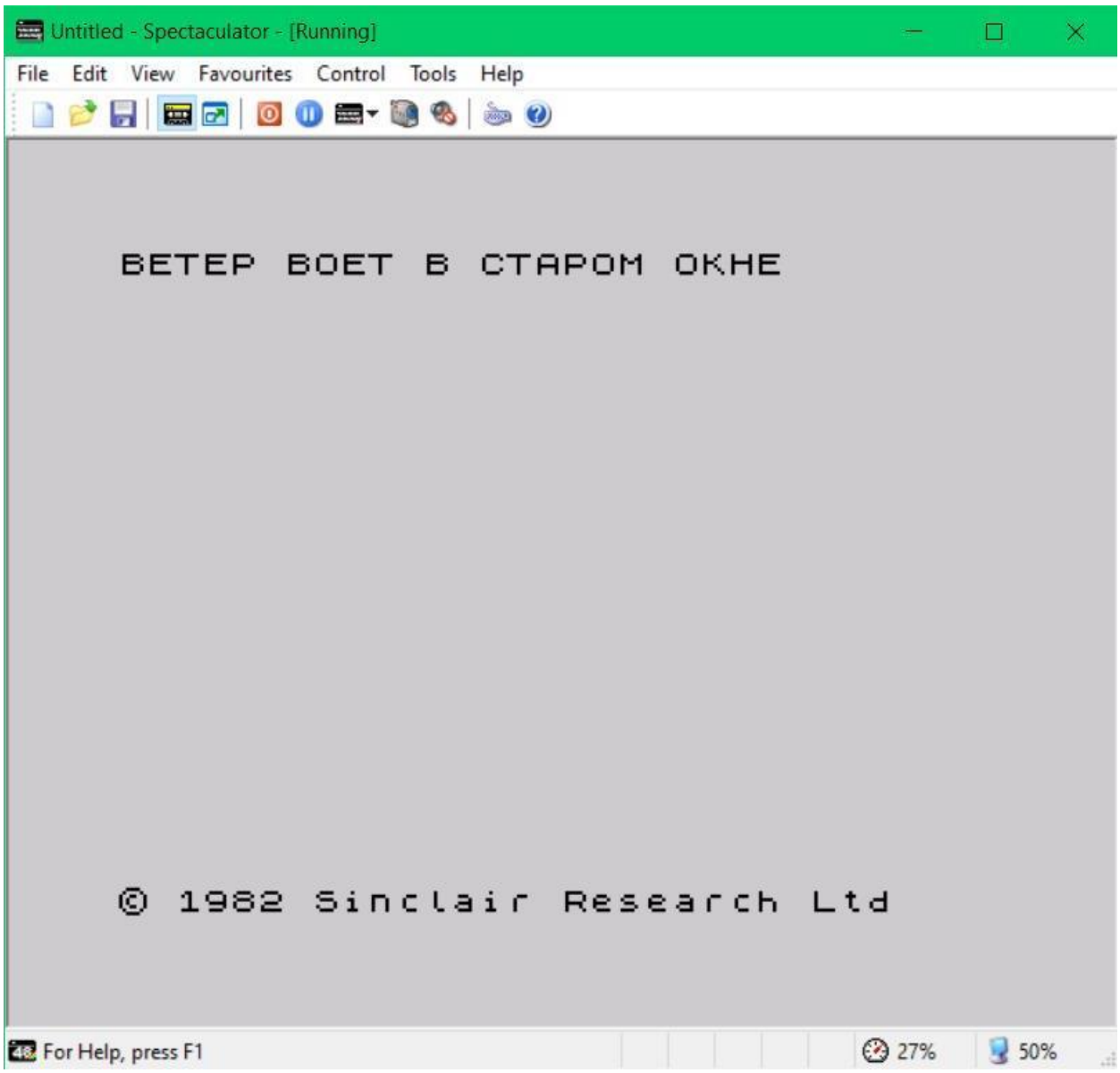


Рис. 279. Вывод монохромного текста с помощью подпрограммы 3082.

Осень. За окном приятный серый полумрак. Моросит дождик. Всё правильно, текст на эту тематику и должен быть в черно-белых тонах.

И, наконец, полноценный вывод цветного текста по координатам:

```
Debugger
Dec
Go To 23296
23296 ← 22 11 6 16 3
```

```
23301 ←B numepckux napagkax
23321 ← 22 12 10 16 2
23326 ←mycum ronoma
23612 ← 32
23613 ← 65364
65364 ← 4777
BC ← 42
DE ← 23296
SP ← 65364
PC ← 8252
Trace
```

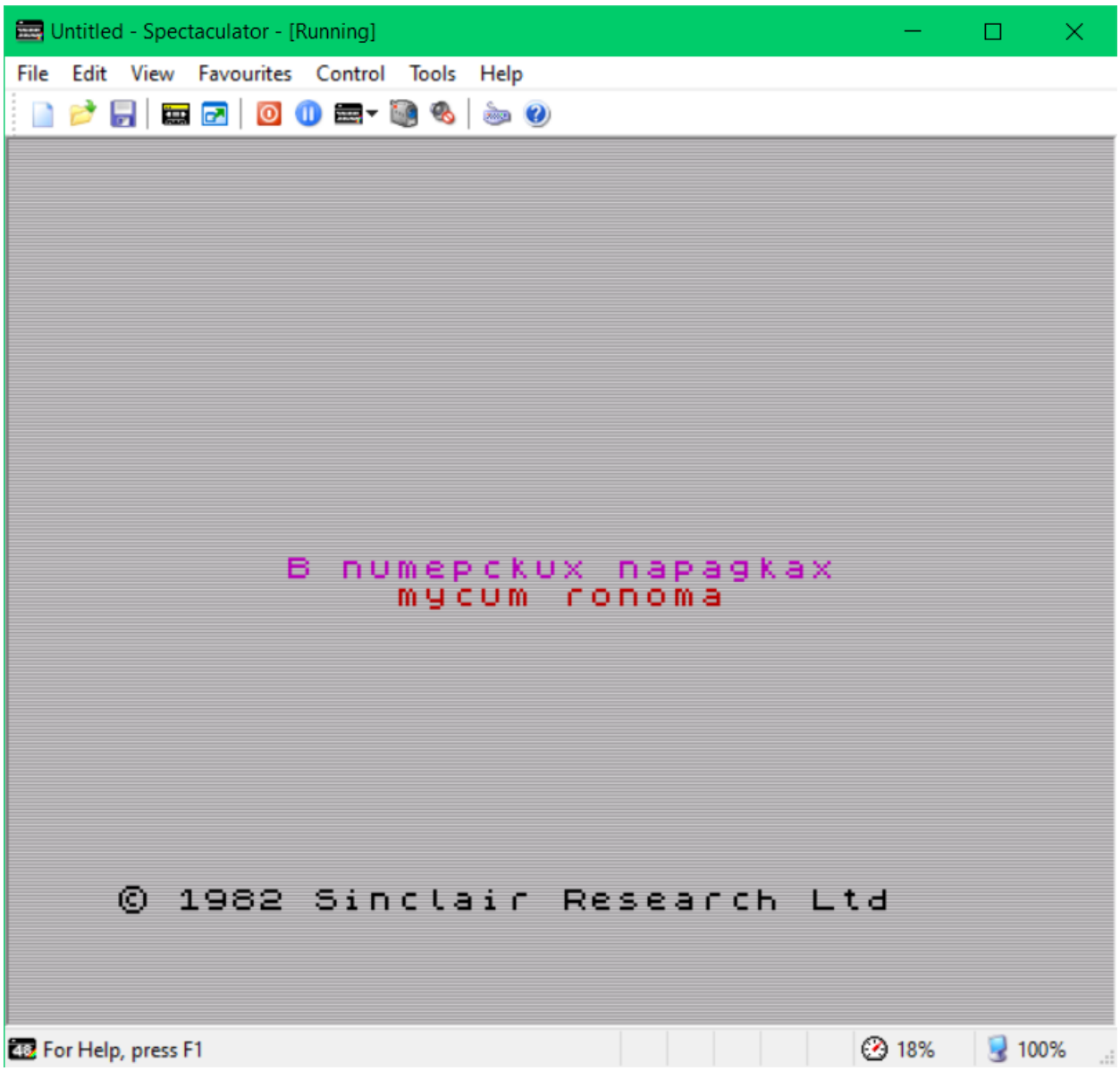


Рис. 280. Вывод цветного текста с помощью подпрограммы 8252.

Вот это уже жизненная фишка. Мир гопников яркий и цветной… но недолгий.

Глава 29
Апокалиптичная графика

Краткое содержание: экранная область, графика, цвет, 16384, 22528

Прежде, чем приступать к изучению графики, неплохо вспомнить строение экрана. Именно вспомнить, потому что эта информация давно баянная и есть в каждой книжке:

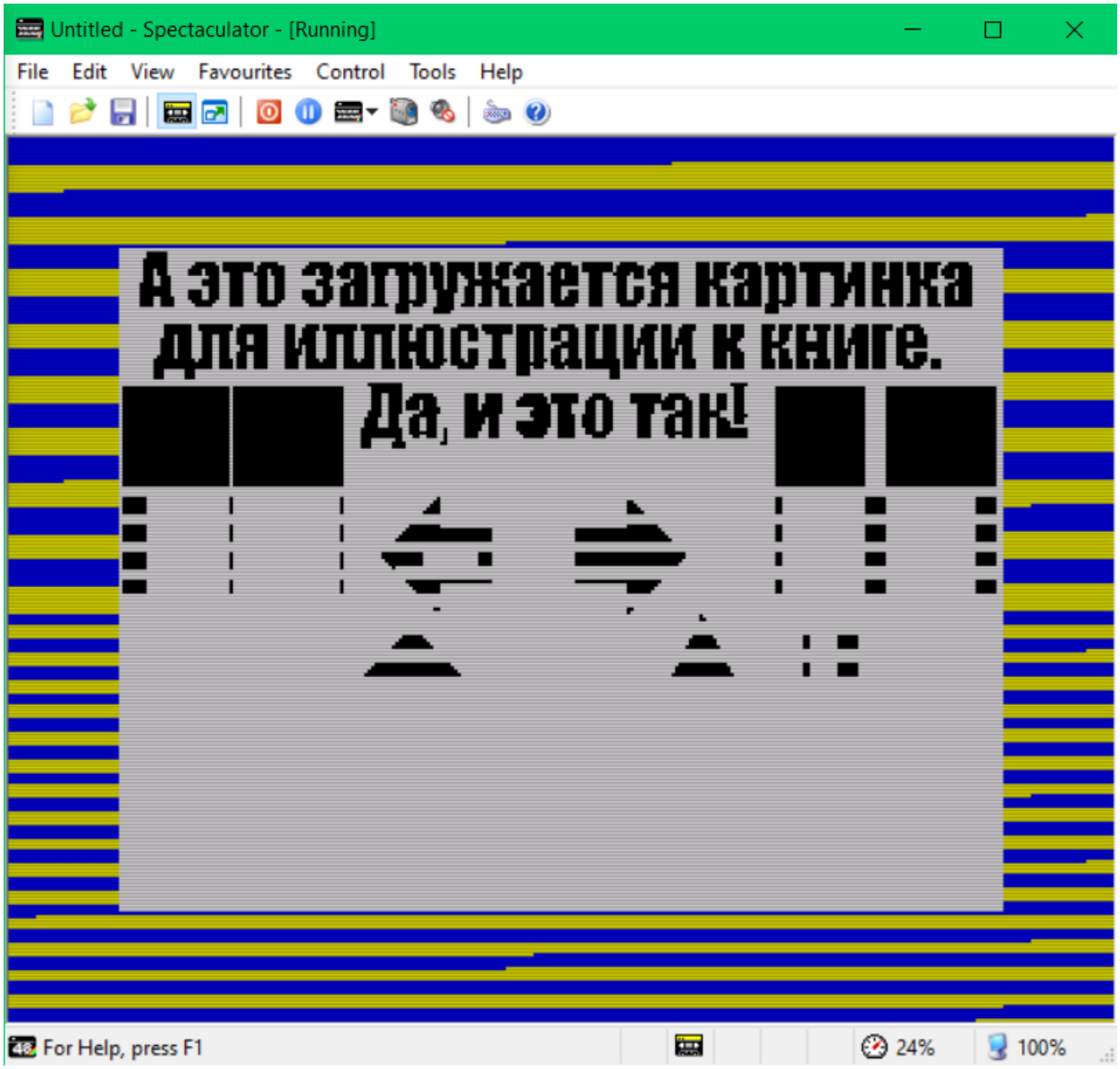


Рис. 281. Процесс загрузки картинки в область экрана.

Суть экранной области в том, что для вывода изображения не существует каких-либо видимых специальных подпрограмм ПЗУ. Вся эта обработка делается на процессорном уровне и скрыта от глаз. Поэтому достаточно просто поместить в отведённый адрес памяти (16384…22527) точку, и она сразу высветится на экране.

В девяностые, один широко известный в узких кругах человек, на примере переключения канала телека, подобный процесс описал так: «хуяк-пиздык». Ну а чего? Лаконично и понятно, особенно если при этом сделать соответствующую жестикуляцию руками.

Если так, то нужно попробовать что-нибудь нарисовать:

```
Debugger
Dec
18794 ← 30972 33852 31864 37376 64632 120
19050 ← 132 132 68 132 132 132 0 84 132 132 132
19306 ← 132 132 68 252 132 132 0 56 132 132 132
19562 ← 132 132 68 132 252 124 0 56 132 132 252
19818 ← 132 132 68 132 132 68 0 84 132 132 132
20074 ← 30852 33924 33924 37376 33912 132
Trace
```

Когда вы это введёте и запустите, то не услышите щемящих звуков гоблинов с рыцарями, зато посередине экрана увидите фразу, которая красноречивее любой музыки расскажет о непростой жизни в нынешних 2020-х годах:

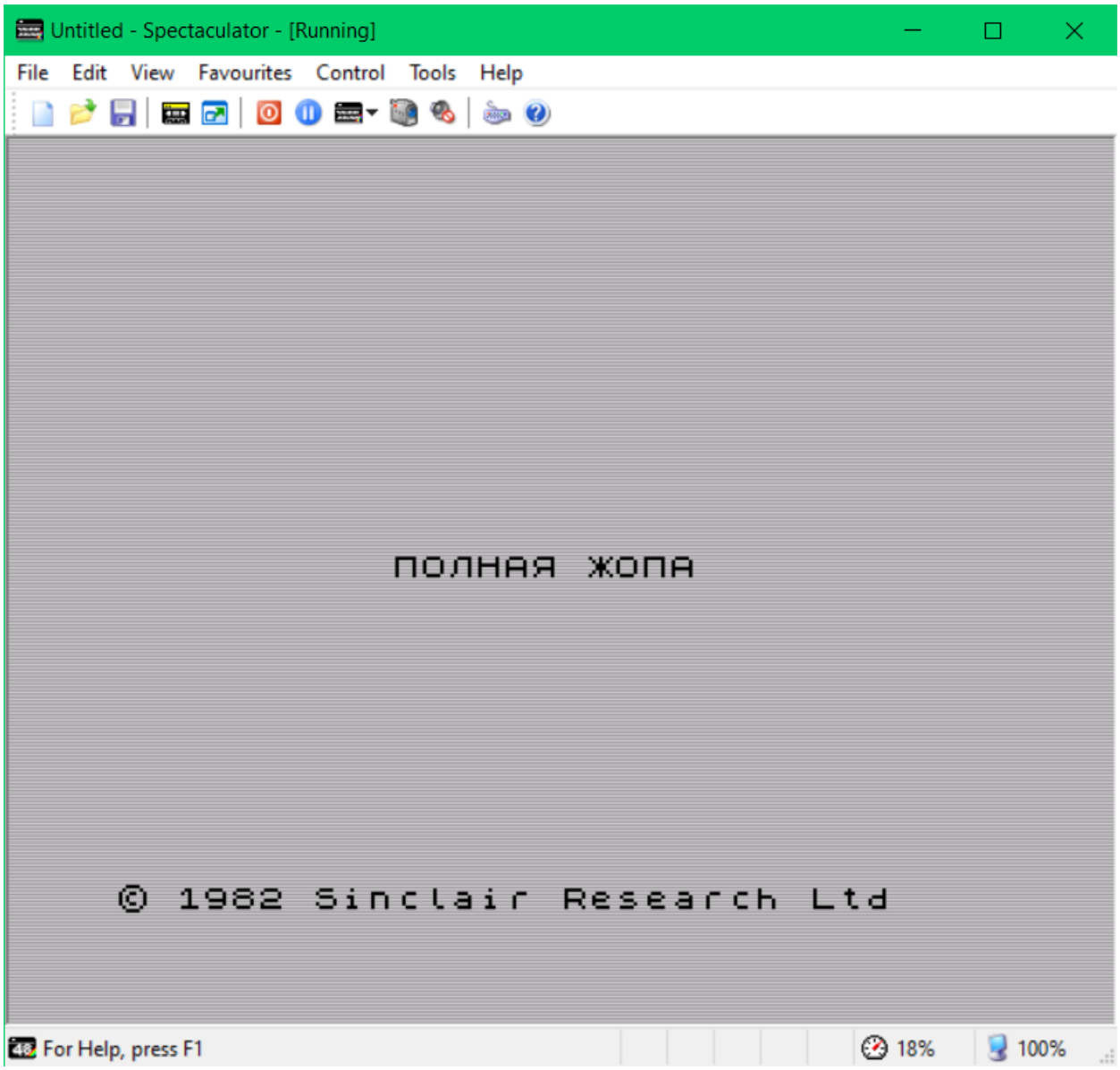


Рис. 282. Вывод графики прямой записью в область экрана.

Область атрибутов экрана простирается с 22528 по 23295. Кодировку цветов кратко можно изобразить следующей табличкой:

INK	0	1	2	3	4	5	6	7
PAPER	0	8	16	24	32	40	48	56
BRIGHT	0	64	-	-	-	-	-	-
FLASH	0	128	-	-	-	-	-	-

Например, чтобы задать выбранной клетке цвета `INK 1 PAPER 6 BRIGHT 1` нужно сложить значения 1+48+64=113.

А теперь для усиления эффекта обязательно нужно покрасить своё творение. Зайдите в «*Debugger*» и добавьте:

```
Debugger
22890 ← 64250 64250 64250 64250 64250 14586
```

Trace

Вводите:

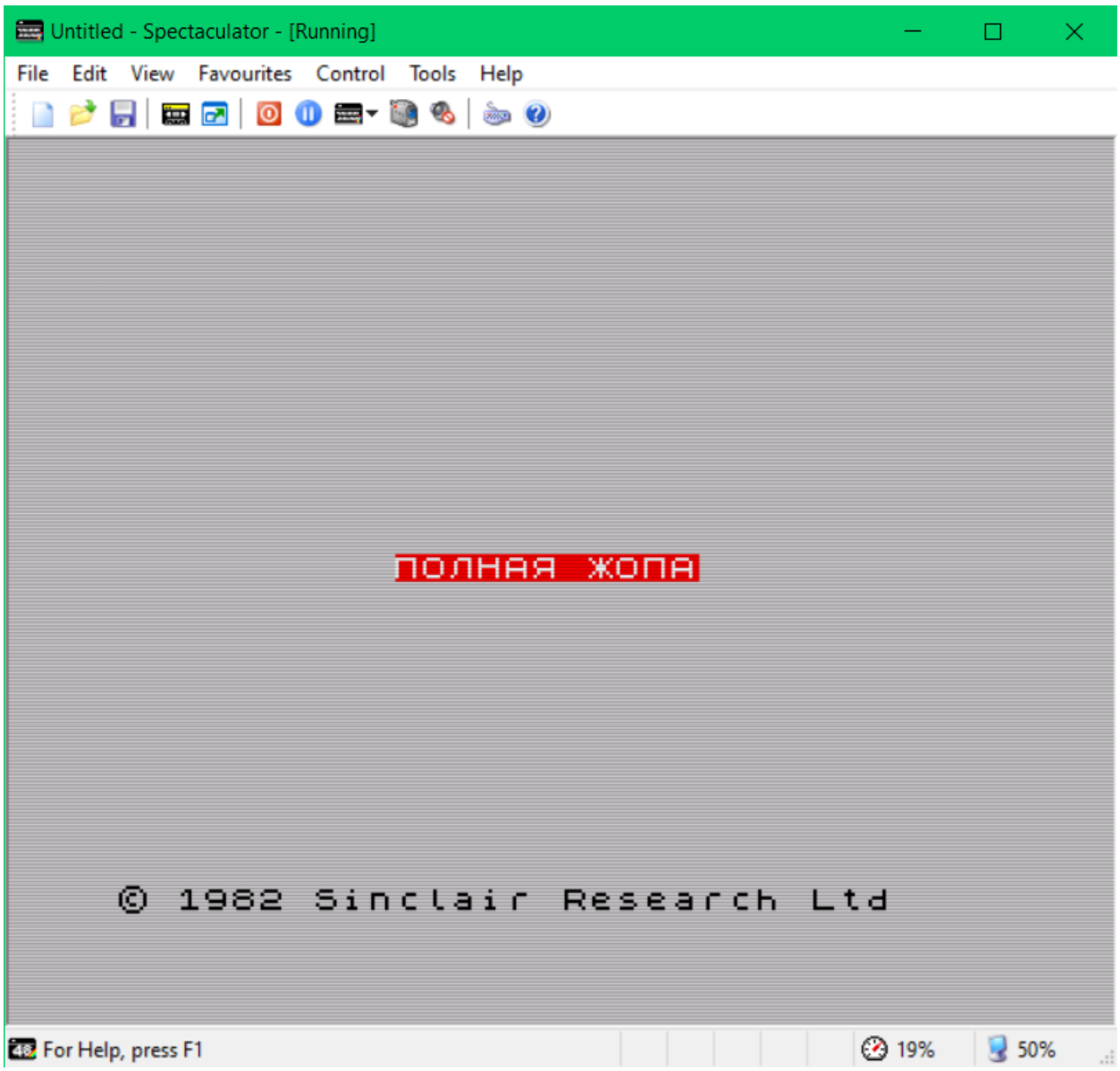


Рис. 283. Закрашивание текста прямым обращением в область атрибутов экрана.

Во-о-о-т! Теперь это не просто ПОЛНАЯ ЖОПА, а ПОЛНАЯ ЖОПА.

Пытаться нарисовать таким образом картинку… даже не подобрать слово, как это назвать. Из режима «*God Mode*» имеется возможность моментально заполнить все точки экрана. Причём, вариантов несколько.

Самым удобным способом моментально закрасить все 6912 байт, является загрузка картинки формата *.scr*. Загрузка это сильно сказано. Достаточно просто из Windows перетащить мышкой картинку на экран и рисунок готов.

Сложнее вопрос, где ее раздобыть. Можно нарисовать в программе ZX-Paintbrush, а можно не мучится, и скачать картинку с сайта, например, из игры «*Rockfall*» с Worldofspectrum.org:

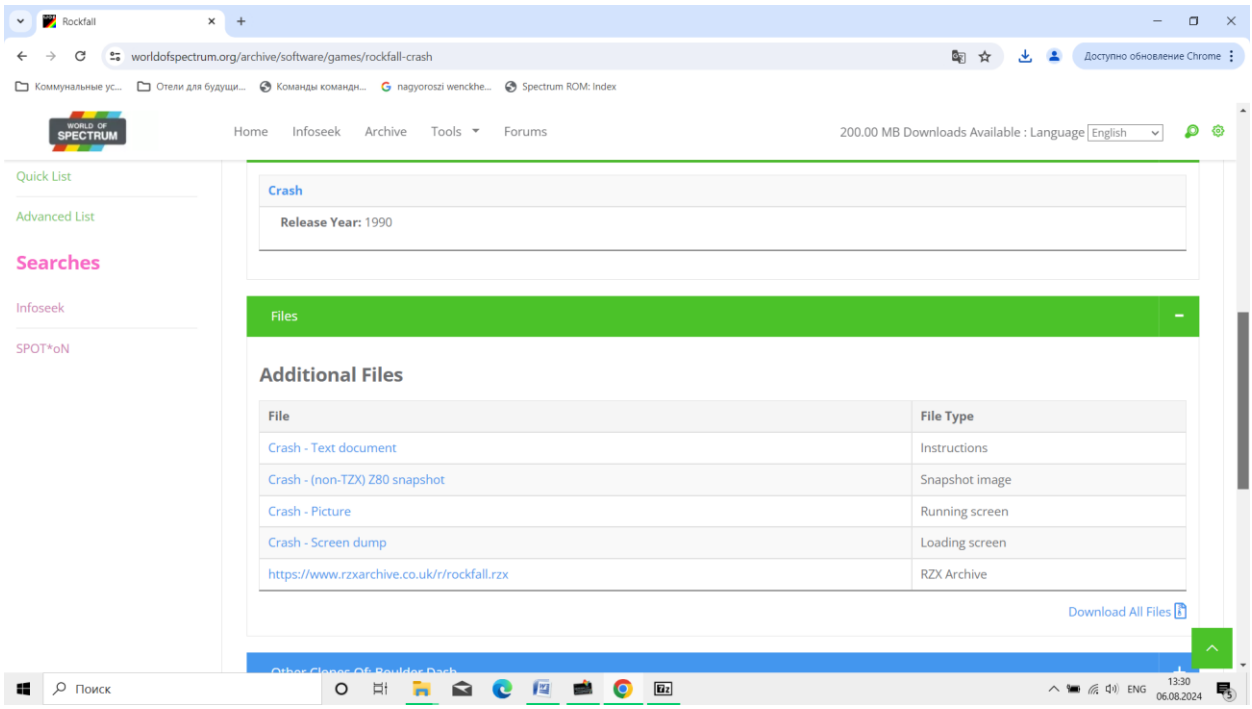


Рис. 284.

Скачав архив, распакуйте и найдите в его комплекте «*Rockfall.scr*»:

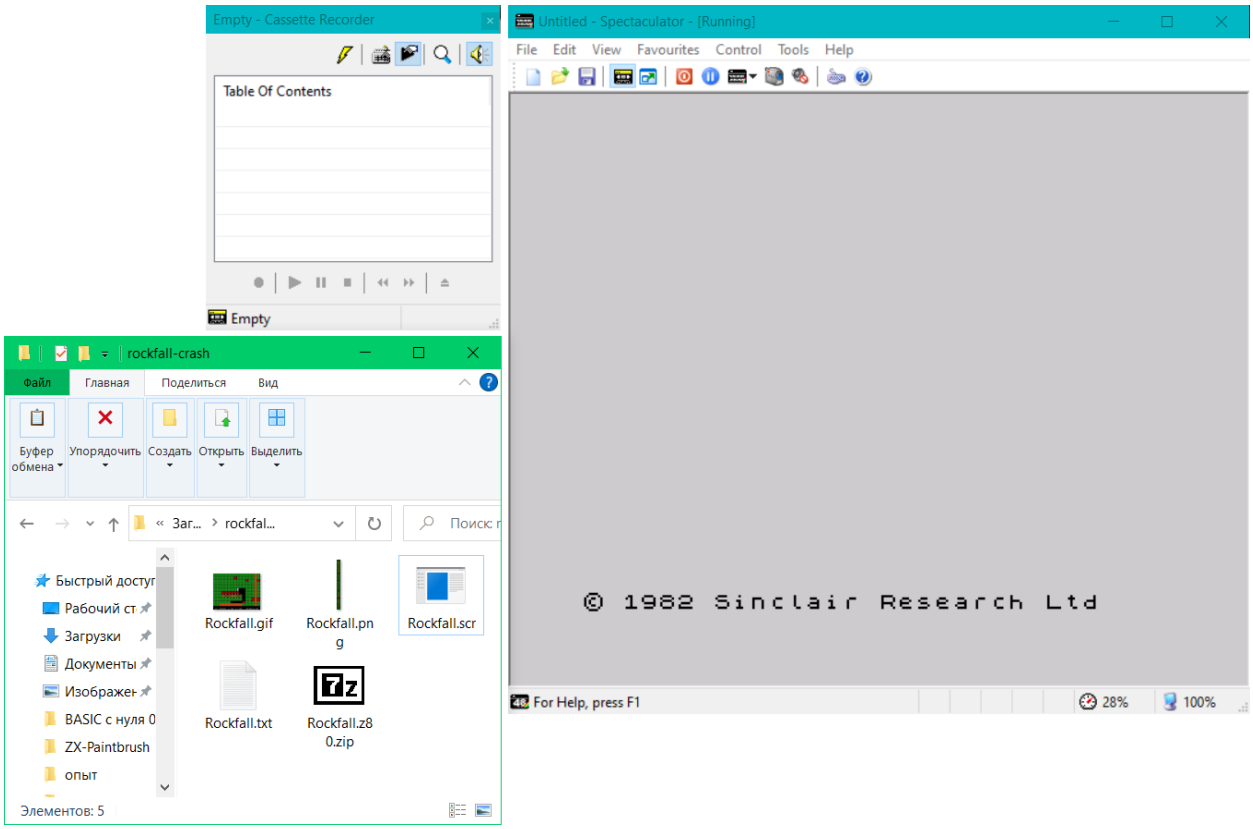


Рис. 285.

Возьмите файл и просто перетащите в окно *Spectaculator*:

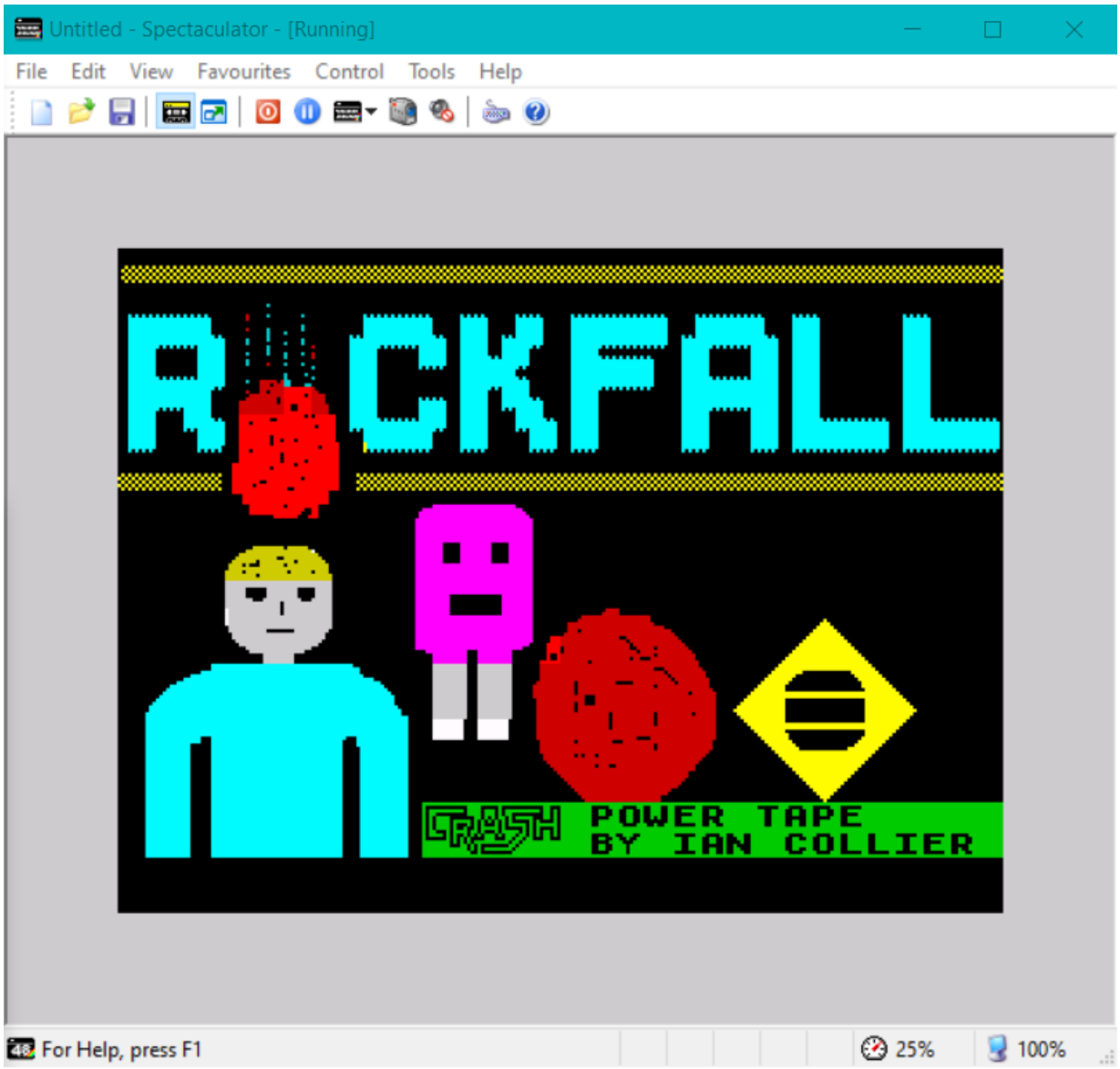


Рис. 286.

Картинка моментально отпечаталась в область экрана, причем с нижними строками.

А теперь немного серьёзнее. Все операторы BASIC, имеющие отношения к графике и выводу на экран (`PLOT`, `DRAW`, `CIRCLE`, `PAPER`, `INK`, `PRINT`…) основаны именно на работе с этой областью. Их задачей является вычислить смежные позиции, чтобы вывести осмысленное изображение на экран с такой жестяной чересстрочной разметкой.

Есть смысл рассматривать эти операторы в данной главе? Возможно, нет, потому что тут идёт чистый ассемблер, с использованием вычислительных подпрограмм и методом алгоритмов тут не сильно напрограммируешь. К тому же их применение можно найти в любой книжке по ассемблеру.

Глава 30
Набор жизненных UDG-символов

Краткое содержание: ПИ, ПЦ, ПЗДЦ, ХУ, ЁБ, подпрограммы PO-GR-X

По традиции главу начну с ретроспективы. Нужно вспомнить, что думали о графическом режиме 30-40 лет назад:

Следующие символы не входят в ASCII, но используются в ZX SPECTRUM, первые из них это 15 черно-белых значков, называемых графическими символами и используемых для изображения рисунков. Их можно ввести с клавиатуры используя, так называемый, графический режим. Если вы нажмете GRAPHICS (CAPS SHIFT 9), то курсор изменится на <G>. Теперь цифровые клавиши с 1 по 8 выдают графические символы, обозначенные на клавишах, а если при этом удерживать SHIFT, то они будут выдавать инверсные символы, т.е. черное становится белым, а белое черным.

Независимо от SHIFT, клавиша с цифрой 9 обеспечивает вам возврат к обычному (< L>-курсор) режиму, а клавиша 0 функцию DELETE.

После графических символов на клавиатуре располагаются символы алфавита от a до U. Графические значения этих клавиш могут определяться самим пользователем, а затем использоваться в графическом режиме.

Рис. 287.

А кстати, почему от A до U то? И какого хрена по нажатию на клавиши V, W, X, Y и Z выдаются полноценные команды? Вообще, откуда на числовых клавишах в режиме G берутся разновидности «▪ ▪ ▪ ▪▪▪▪▪ ▪▌ ▪ ▪ ▪ ▪■» вот этих квадратиков? Почему никто об этом не пишет, и принимает как должное?

Правильным ответом на все эти вопросы будет коротким: «Нужно разбираться!». Первым делом предлагаю узнать, откуда берутся квадратики.

Допустим, в режиме курсора G нажата клавиша «6»:

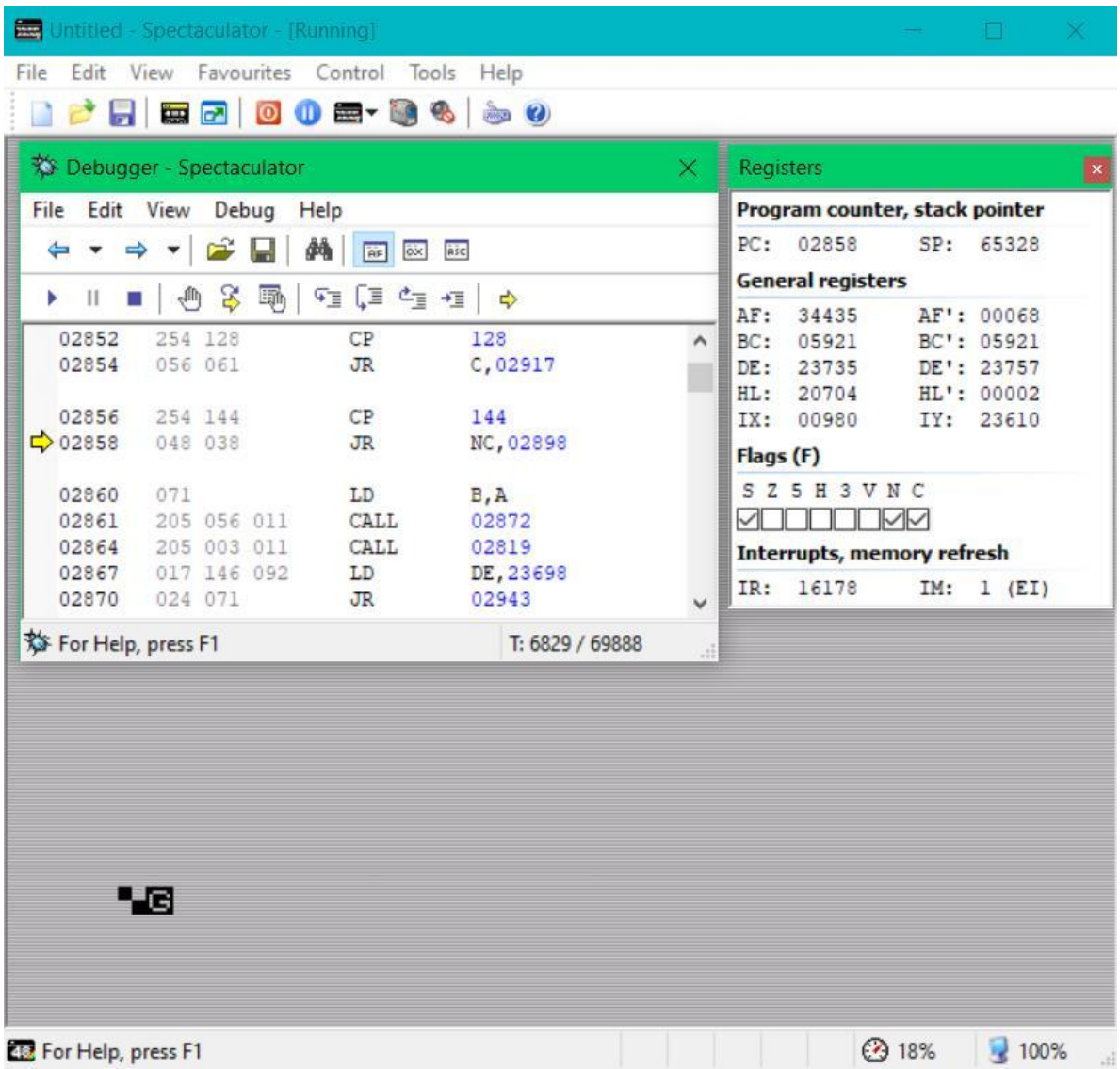


Рис. 288.

После нажатия «6» Стрелочка попадает на программу PRINT ANY CHARACTERS по адресу 2852. В процессе распознавания символов первым делом отбрасываются коды менее 128, то есть всё, что меньше графики UDG. Следом в 2856 просматривается граница числовых и буквенных графических символов. Если символ с кодом больше «144» (буквы UDG и команды BASIC) то нужно идти на дальнейшую сортировку по адресу 2898.

Если символ попал в интервал от 128 до 143 , то это числовые графические символы и можно приступать к их печати:

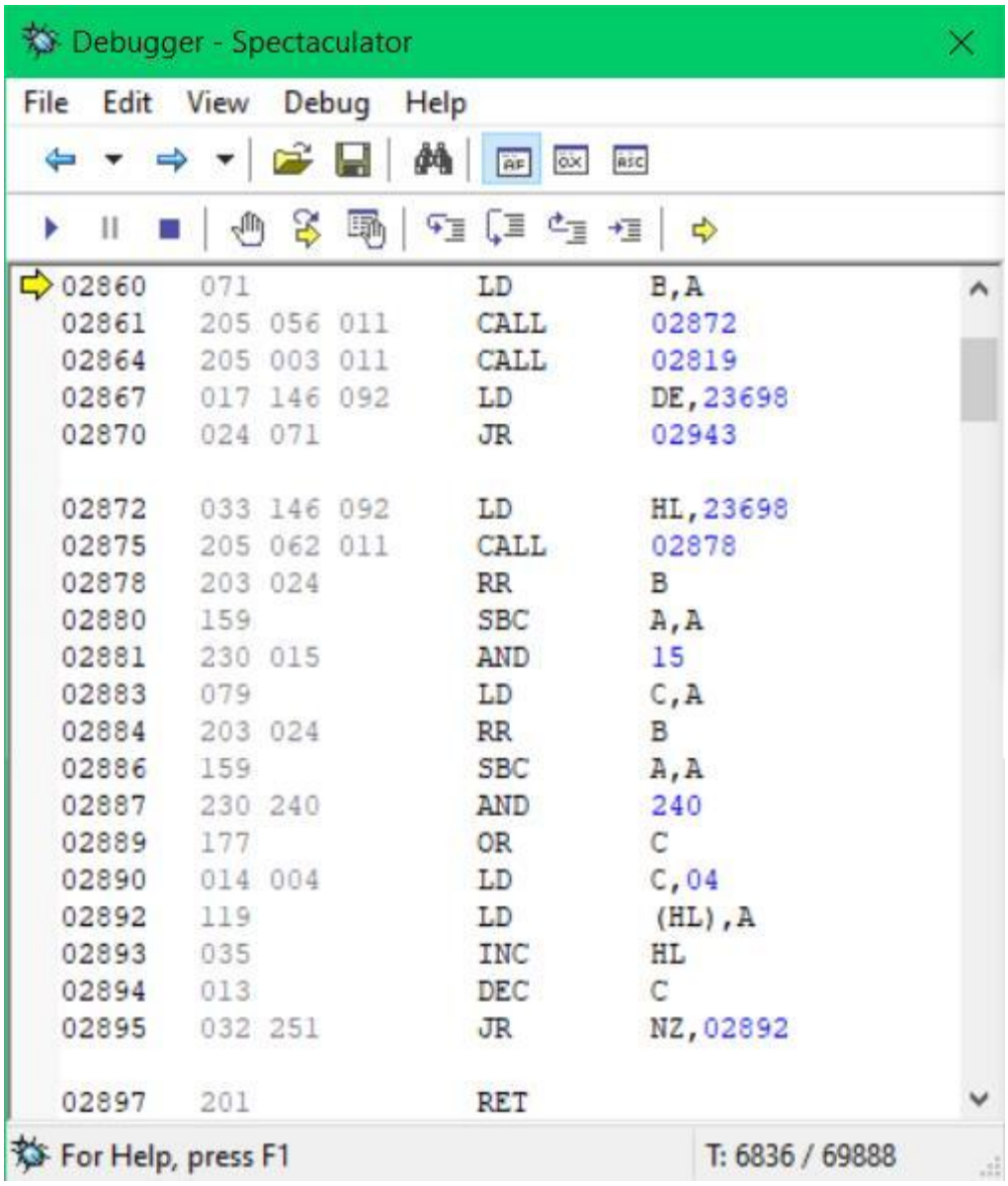


Рис. 289.

Скопировав код символа в «B», Стрелочка бежит в комплексы подпрограмм PO-GR-1 и PO GR-2. Только вот беда. Таких символов в наборе CHARS нет. Сперва их нужно сгенерировать. Прежде всего, готовится место, где будет создаваться рисунок символа. Под это дело выбирается часть пустыря области MEMBOT (23698-23727) в системных переменных, для чего в «HL» записывается значение 23698.

С адреса 2878 начинается алгоритм создание рисунка, в зависимости от кода символа. Сгенерировав полоску 1x8, размером в 1 байт, по адресу 2892 происходит её запись в отведённую область. Таким образом, за два захода, в куске массива MEMBOT формируется полноценный 8-ми байтный рисунок:

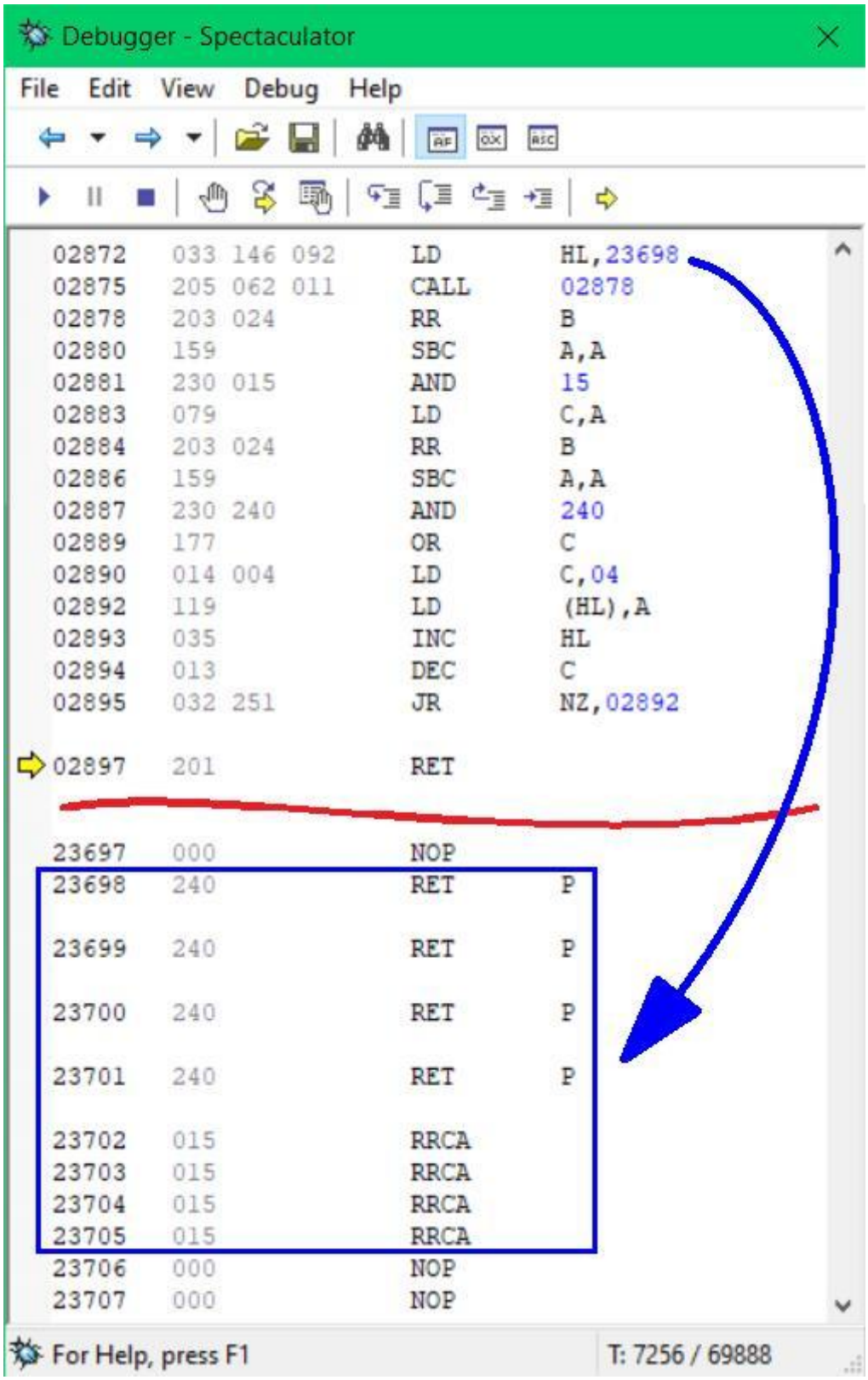


Рис. 290.

После формирования Стрелочка приступает к выводу символа на экран.

Теперь по поводу буковок в режиме UDG. А что будет, если нажать букву «V» в графическом режиме?

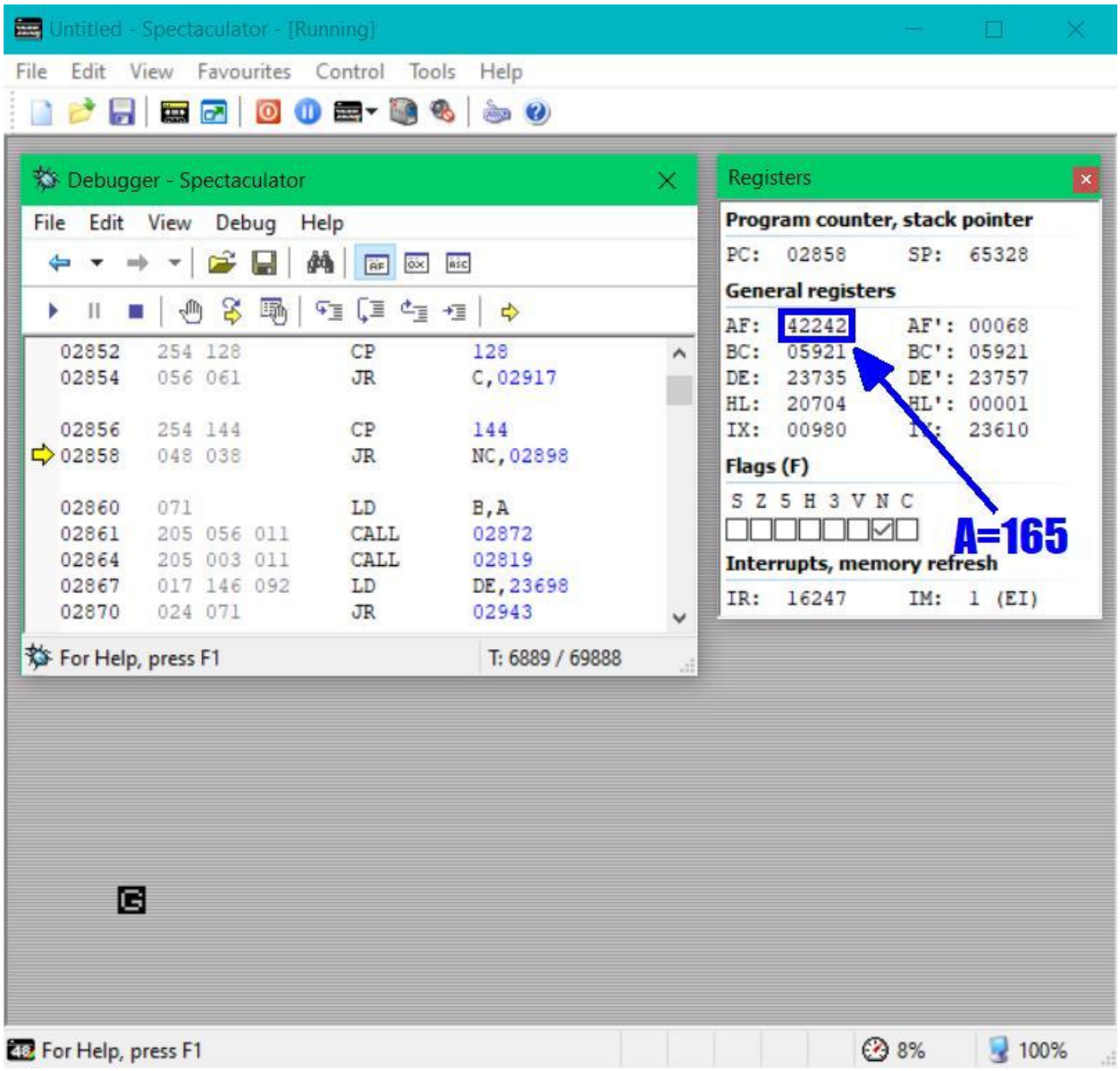


Рис. 291.

Обратите внимание, вопреки здравому смыслу, код буквы «V» в графическом режиме к этому времени распознаётся, как «165» и является как бы продолжением версии буквенного алфавита режима UDG. Что-то пошло не так.

Далее эта буква с неправильной кодировкой попадает в сортировочный центр PO-T&UDG (2898):

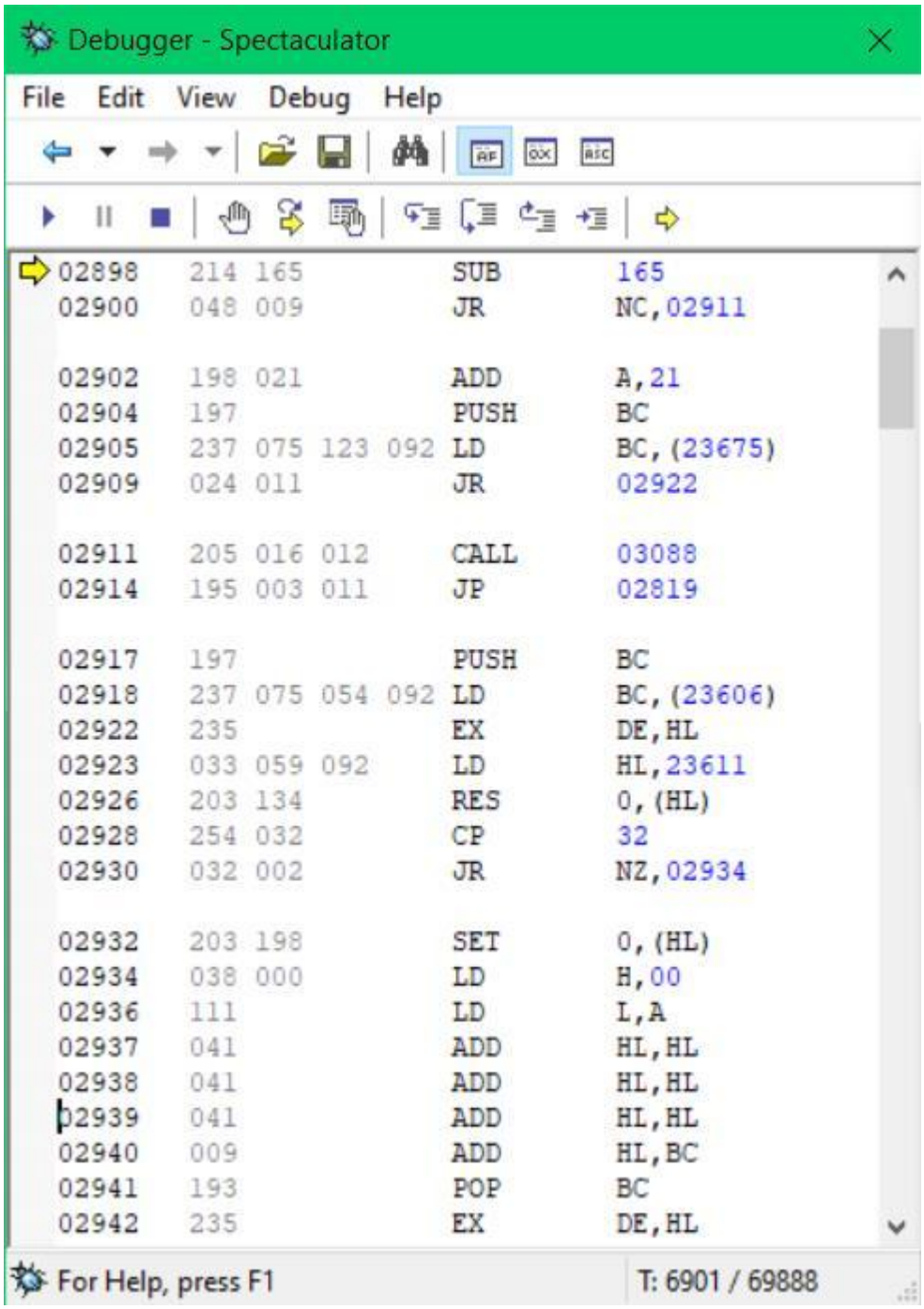


Рис. 292.

Тут отделяется диапазон графических символов 144-164 от BASIC команд, которые имеют код от 165 и выше. Если символ оказывается в рамках букв A-U, то Стрелочка продолжит выполнение программы со следующего адреса 2902, где приведет букву к нумерации от 0 до 20 (`2902 ADD A, 21`). Из переменной `UDG (23675)` берётся начало размещения рисунков букв. Прибавляя к адресу код сформированной буквы, умножением на 8 вычисляется местонахождение рисунка для неё.

В текущем случае, буква «V», напечатанная в графическом режиме имеет ошибочный код «165». Вполне естественно, что в 2900 по команде `JR NC 2911` стрелочка пойдёт печатать команду `RND` куском знакомой подпрограммы `MESSAGE PRINTING` по адресу 3088. Вывод сообщения с её помощью, только с точкой входа 3082, рассматривался в позапрошлой главе.

Тут всё логично и правильно, интересно другое: где происходит ошибочная формировка кодов символов. Обратите внимание на уникальность ситуации. Коды с

номерами 165, 166, 167, 168 и 169 имеют двойное значение. Кроме команд (RND, INKEY$, PI, FN, POINT), в графическом режиме они обозначают буквы от V до Z.

Вариантов немного, особенно если вспомнить, где формируется код нажатой клавиши. Да, это снова пачкает мозги, программа K-DECODE по адресу 819. В главе 7 «*Удивительный режим MODE*» разбиралась недоработка по адресу 825. Сейчас в коллекцию к ней добавился недочёт по адресу 830:

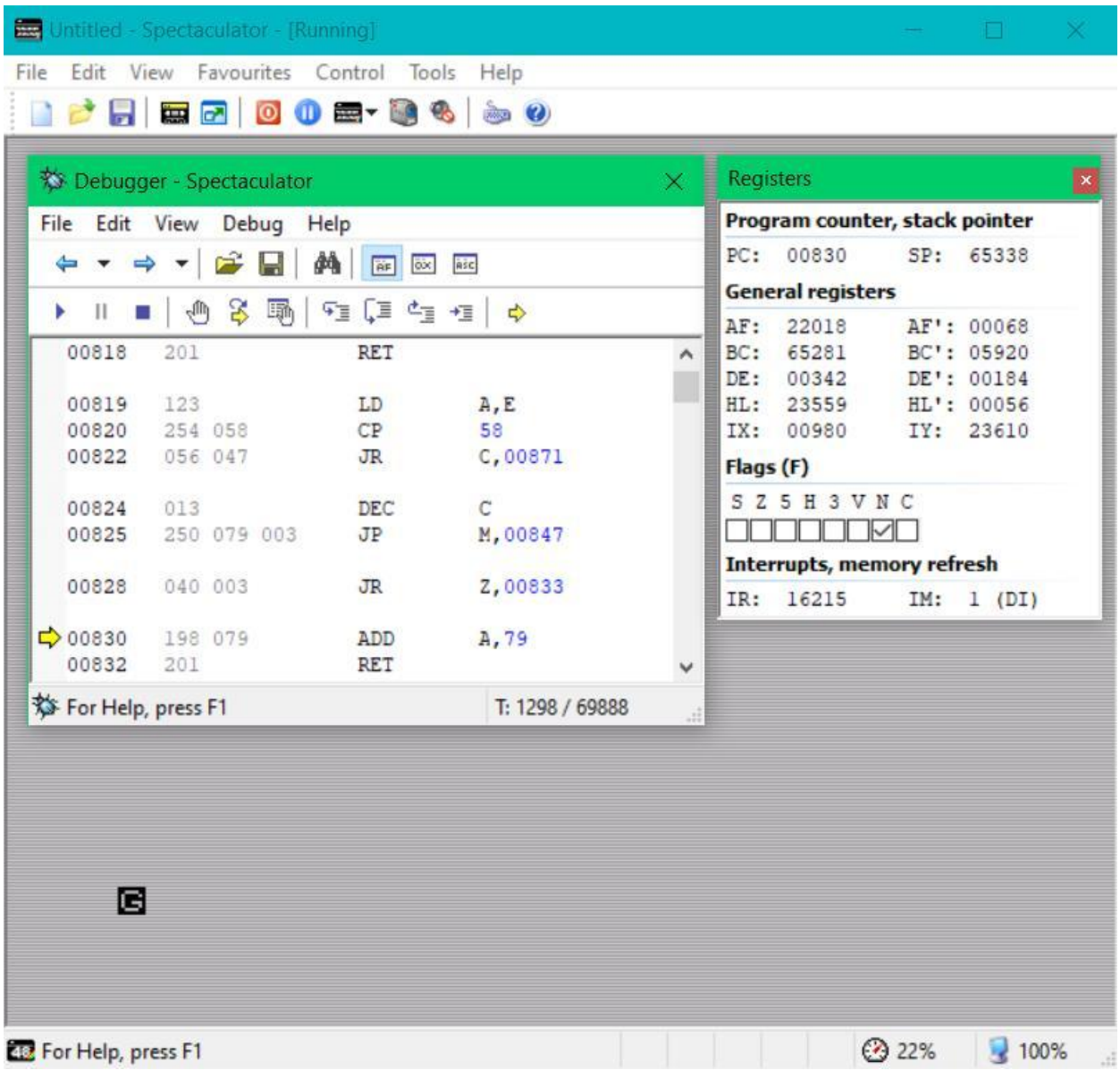


Рис. 293.

Вот тут это и происходит. Поняв, что буква нажата в режиме курсора G, к коду обычной буквы добавляется +79, чтобы получить код UDG. И если с клавишами от A до U это прокатит, то прибавив к коду V число 79, получается именно 165, что наглядно видно на картинке выше.

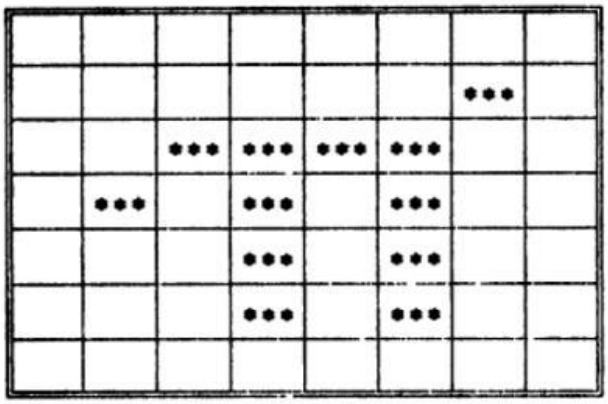
Теперь следующий извечный вопрос: «Что делать?» Очевидно, что нужно перенести набор UDG куда-то выше. Запрограммировать рисунки для недостающих клавиш, потом нажать V, W, X, Y или Z и с помощью точки прерывания на 2900 поставить C ☑ галочку «C» и направить Стрелочку на нужный путь.

Можно приступить к практике и начать с самого простого. Узнать, какие символы предлагали создать в самоучителе:

Определение графики этих клавиш проиллюстрируем на примере определения символа буквы греческого алфавита "ПИ".

1. Каждый символ представляется точками в матрице 8 х 8, поэтому мы в начале начертим диаграмму, приведенную на рисунке. Мы оставим по одной клетке по периметру символа для отделения его от соседних знаков.

2. Закрепим данный символ за клавишей "Р", так, чтобы при нажатии клавиши в графическом режиме выдавался символ "ПИ".



3. Запрограммируем это изображение. Каждый определяемый пользователем символ запоминается в памяти восемью знаками, по одному на каждый ряд.

Рис. 294. Фрагмент главы «Набор символов» самоучителя по BASIC.

Отчасти соглашусь с авторами. Вместо буквы «Р» нарисовать «ПИ», решение вполне логичное, но слишком поверхностное. Что такое «ПИ» без «ЗДЕЦ»? Это как туалет без унитаза или программист без ЭВМ.

Копнув чуть глубже, вспоминается старая поговорка: «Всё есть как в Греции». И действительно, в греческом алфавите, помимо «ПИ» есть куча других хороших букв, таких как «Ӿ», «Ё» и прочие:

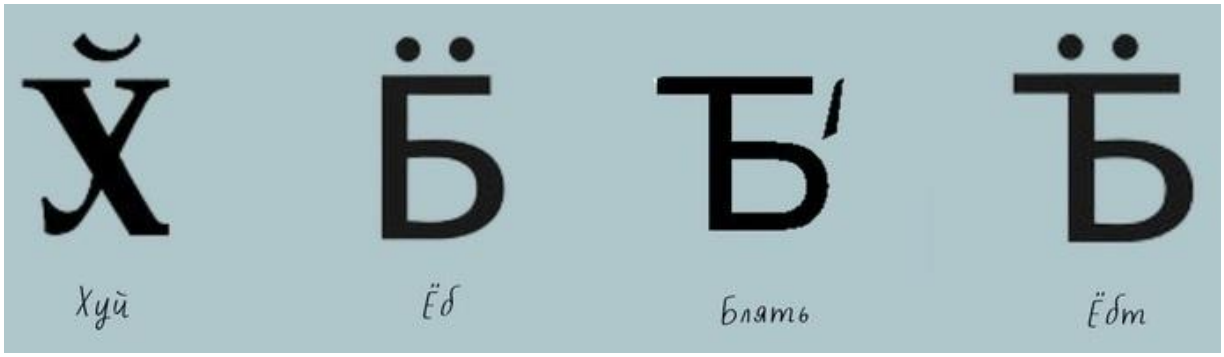


Рис. 295. Некоторые разновидности специальных букв. Фото из интернета.

Мало того, у самой «ПИ» существует несколько вариаций, которые более ярко отражают это состояние души. Самыми известными будут «ПЦ» и «ПЗДЦ», которые тоже отсутствуют в английском алфавите. Последнее является самым приоритетным, но, к сожалению, на Спектруме, в знакоместе 8х8 эмоции такого плана уместить проблематично. Ну, в английском алфавите много чего отсутствует. В нём даже букв всего 26, а в графическом режиме вообще 21. Поэтому, предлагаю ограничиться созданием буквы «ПЦ»:

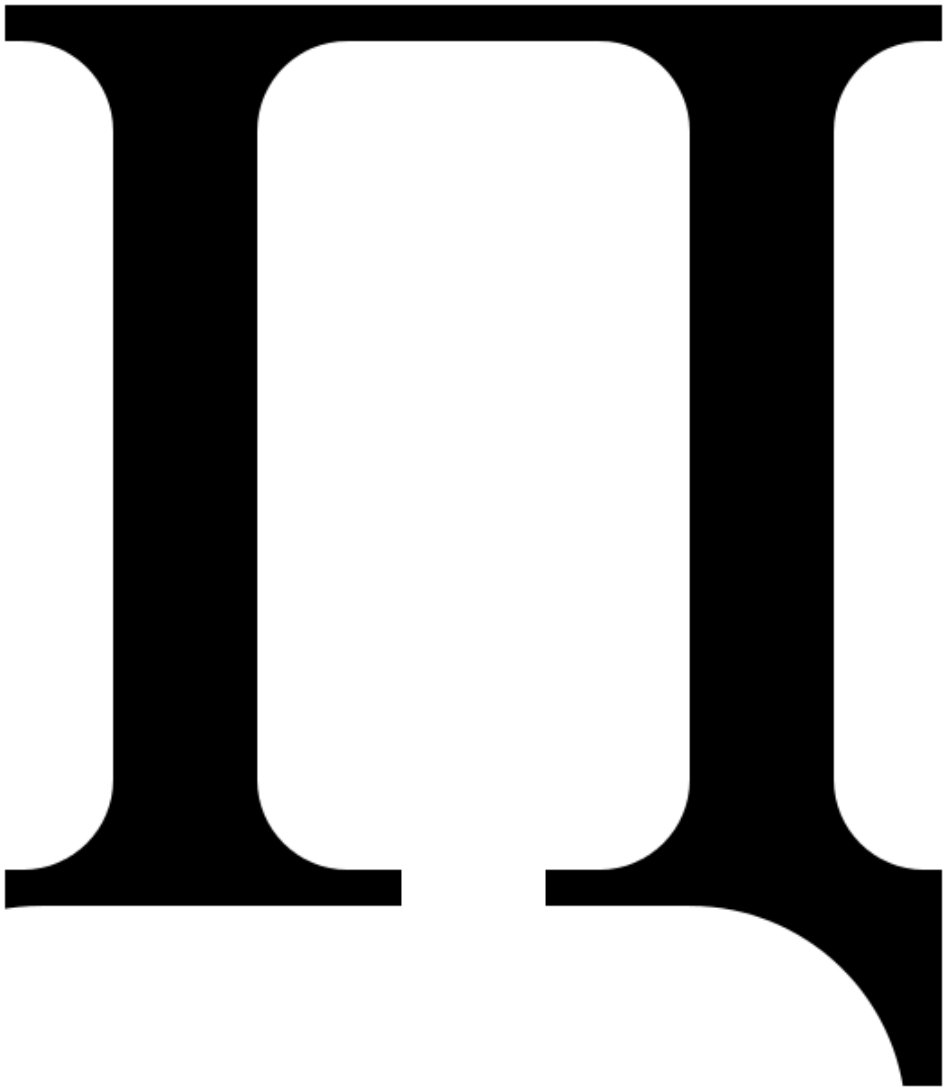


Рис. 296. Буква «ПЦ». Фото из интернета.

Сделать её также легко. Достаточно набрать следующий алгоритм:

```
Debugger
Dec
Go To 65489
65489 ← 124 68 68 68 68 70 2
Trace
```

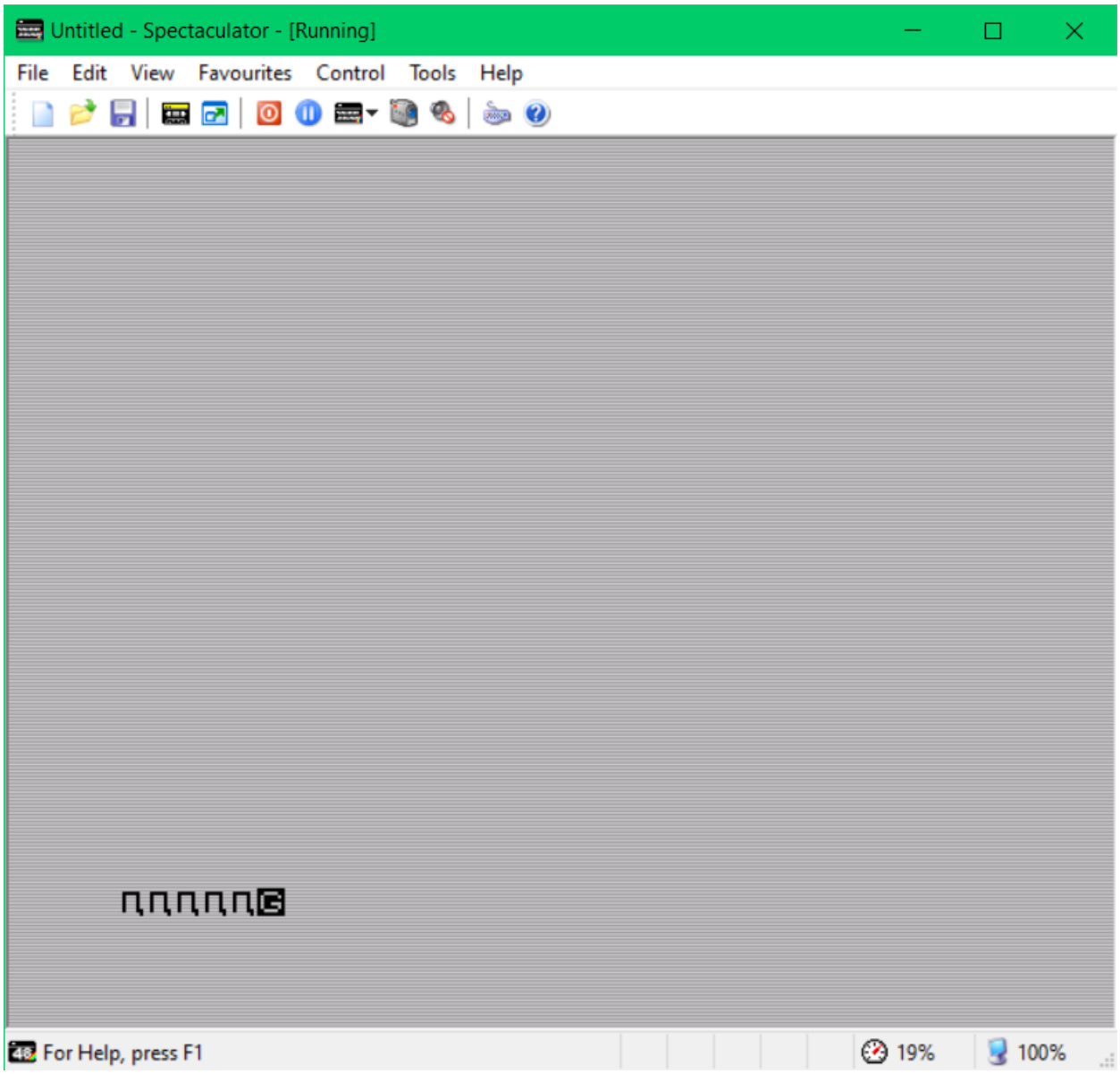


Рис. 297. Буква ПЦ в версии ZX-Spectrum.

Даааа… Выглядит не слишком эстетично, но от такого разрешении ожидать лучшего бессмысленно. Остальные спецбуквы «греческого» алфавита можете сконструировать самостоятельно.

Напоследок вывод несуществующего 22-го графического символа UDG путём нажатия на букву «V». Введите и запустите этот трёхступенчатый алгоритм:

```
Debugger
Dec
Go To 23611
23611 ← 32
23617 ← 2
23675 ← 60000
60168 ← 0 146 84 56 56 84 146 0
Add Breakpoint 2900
Trace
BASIC ← V ENTER
Trace
Remove Breakpoint 2900
AF ← AF+1
Trace
```

После выполнения алгоритма от нажатия «V» на экране появляется:

Рис. 298.Вывод символа буквой V в графическом режиме. Фрагмент нижней части экрана.

Буква «Ж», а не привычная команда RND.

ЧАСТЬ II. СИНЯЯ КУЛЬМИНАЦИОННАЯ
ВНЕШНЯЯ ПАМЯТЬ НА МАГНИТНОЙ ЛЕНТЕ

Пролог

В начале июня 2024 года, когда книга еще не приобрела свои нынешние очертания, с таким названием появилась глава-пародия на 3-4 странички. Сырой черновик я поместил на задворки будущей книги.

Жизнь внесла свои коррективы. В процессе дальнейшего изучения структуры BASIC в ПЗУ лавиной посыпались эксклюзивные открытия, которые я не успевал записывать в черновик. Глава начала разбухать. 10, 20, 30... страниц. Поняв, что для главы это перебор, она была разбита на несколько небольших глав. Шквал открытий не утихал. В процессе написания выявлялись всё новые и новые аспекты. Вскоре и разделившиеся главы начали разрастаться на десятки страниц. Настал момент, когда от разделившихся глав начали отпочковываться новые. Количество глав только на тему записи/считывания превысило 150 страниц. Назрело отделение темы в отдельный раздел.

А почему раздел назван синим, тут ответ простой. В словосочетании «красный-синий-зелёный», слово «синий» идёт вторым. А сама цветная философия родилась от любимейших аудио кассет «*DJ Цветкоff – In the mix*», которые окончательно перевернули все музыкальные вкусы. Почти 28 лет подряд, я слушаю эти сборники, и с годами всё сильнее влюбляюсь в волшебную «пицикатную» музыку. Первая кассета в красном дизайне вышла в октябре 1997 года, синяя кассета в декабре 1997-го, а зелёная в конце марта 1998-го. С развитием философии ярких цветов родился и  логотип.

Глава 31
Приобретение и настройка мафона

Краткое содержание: окно «Cassette Recorder», настройка виртуального магнитофона

Кстати о магнитофоне и кассетах. «Внешняя память на магнитной ленте» – таким многообещающим заголовком начинается глава, которая должна знакомить с записью и загрузкой с магнитофонной ленты, а на деле…:

Глава 20

Внешняя память на магнитной ленте

Краткое содержание: LOAD, SAVE, VERIFY, MERGE

Основные команды работы с магнитофоном SAVE, LOAD и VERIFY уже рассматривались во вводном описании. Вы могли видеть, что LOAD затирает старую программу в памяти компьютера при загрузке новой программы с ленты. Есть другая команда MERGE, не делающая этого. Эта команда стирает лишь те строки старой программы или переменные, которые совпадают с номерами строк новой программы или именами новых переменных. Программу " DICE " (" игральная кость ") из главы 11 запишем на ленту под именем "DICE".

А теперь введем и выполним следующую программу:

```
1 PRINT 1
2 PRINT 2
10 PRINT 10
20 LET X=20
```

Затем осуществим ее проверку, заменив команду VERIFY "DICE" на команду MERGE "DICE". Вы увидите, что строки 1 и 2 сохраняются, а строки 10 и 20 заменяются на строки с этими номерами из программы "DICE", переменная X тоже сохраняется (проверьте PRINT X)

Теперь вы знаете четыре оператора для работы с кассетным магнитофоном:

SAVE - записывает программу и переменные на магнитофон;

VERIFY - проверяет программу и переменные в памяти компьютера по их копии на ленте;

LOAD - очищает память компьютера от всех программ и загружает в нее новые данные, считанные с магнитофона;

MERGE - подобно LOAD, только не очищает всю память, а лишь заменяет те строки программы или переменные, у которых совпадают номера или имена с такими же на магнитной ленте;

Рис. 299. Отрывок типовой главы «Внешняя память на магнитной ленте» из самоучителя по BASIC.

Совершенно верно, и не только видеть, но даже слышать! Набираю команду `LOAD ""CODE` после сброса на чистый компьютер. Голубая рамка сменяет красную, и вскоре: «Пи-и-и-и-и-и-и-и-и-и-и Тщю-у-у-ю. Пи-и-и-и Шю-ю-у-у-ю-у-и-и…». Замените команду `VERIFY`, о которой даже не упоминалось на `MERGE`. А в целом всё уже рассматривали, всё всем понятно и можно заканчивать главу. Ну не знаю, какую туалетную бумагу нужно курить, чтобы потом так торкнуло. Точно не 54-х метровый «Светогорский Стандарт». Подозреваю, что «Сясьскую», вот она ядрёная, пробирает до глубины души. А впрочем… даже комментировать эту красотищу уже лень. Давайте, и правда, сразу к делу.

Чтобы что-то считать, нужно это что-то записать. Для начала нужно раздобыть кассетный магнитофон. *Spectaculator* через микрофонный вход, позволяет подключить реальный магнитофон, но я предлагаю ограничиться виртуальным. Он будет удобнее.

Итак, откройте *Spectaculator* и первым делом нажмите на кнопочку с кассетой  «*Cassette Recorder*» или комбинацию клавиш «*Ctrl+K*».

Откроется следующее окно:

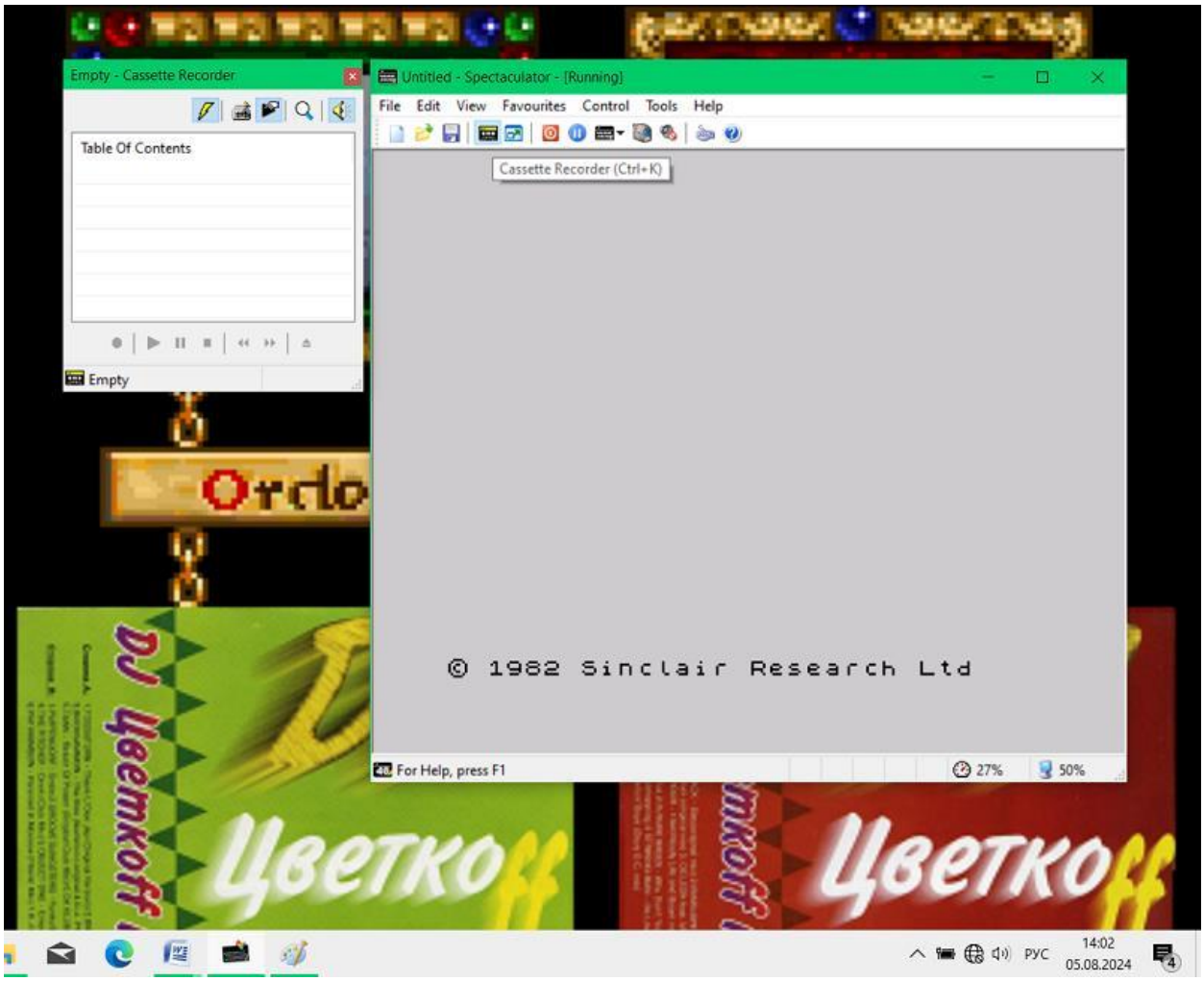


Рис. 300. Появление окна виртуального магнитофона.

Это виртуальный кассетный магнитофон и пока он пустой, о чём сигнализируют посеревшие кнопки управления и надпись «*Empty*».

В большинстве случаев, при установке программы, настройки магнитофона выставляются так, что самая важная функция загрузки с пищанием в реальном времени отключена. Поэтому вверху окна найдите пункт «*Tools*», откройте его и найдите там «*Options (Ctrl+Y)*»:

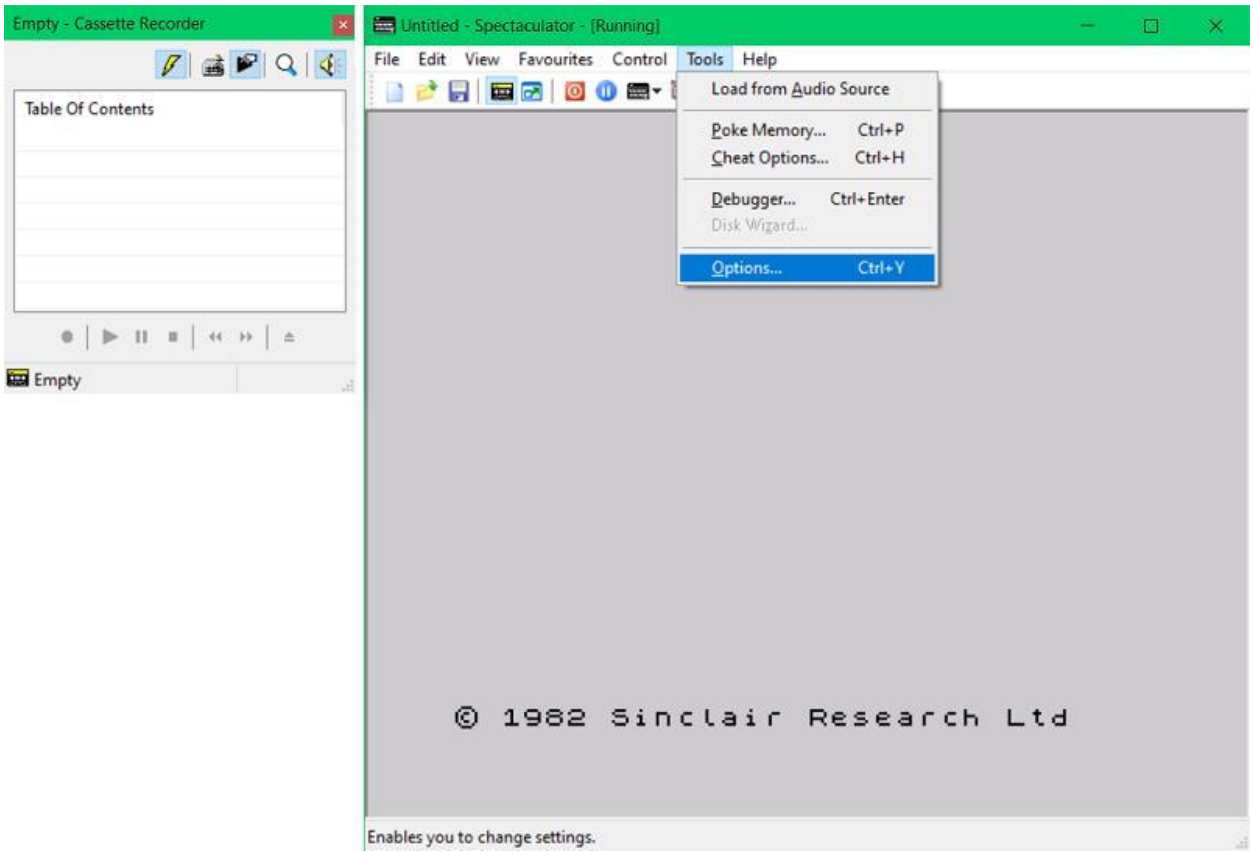


Рис. 301. Выбор меню «Options».

Нажав на пункт «*Options* (*Ctrl+Y*)» откроется окошко настроек. Ещё раз убедитесь, что все галочки периферийных устройств «*Additional Hardware*» выключены:

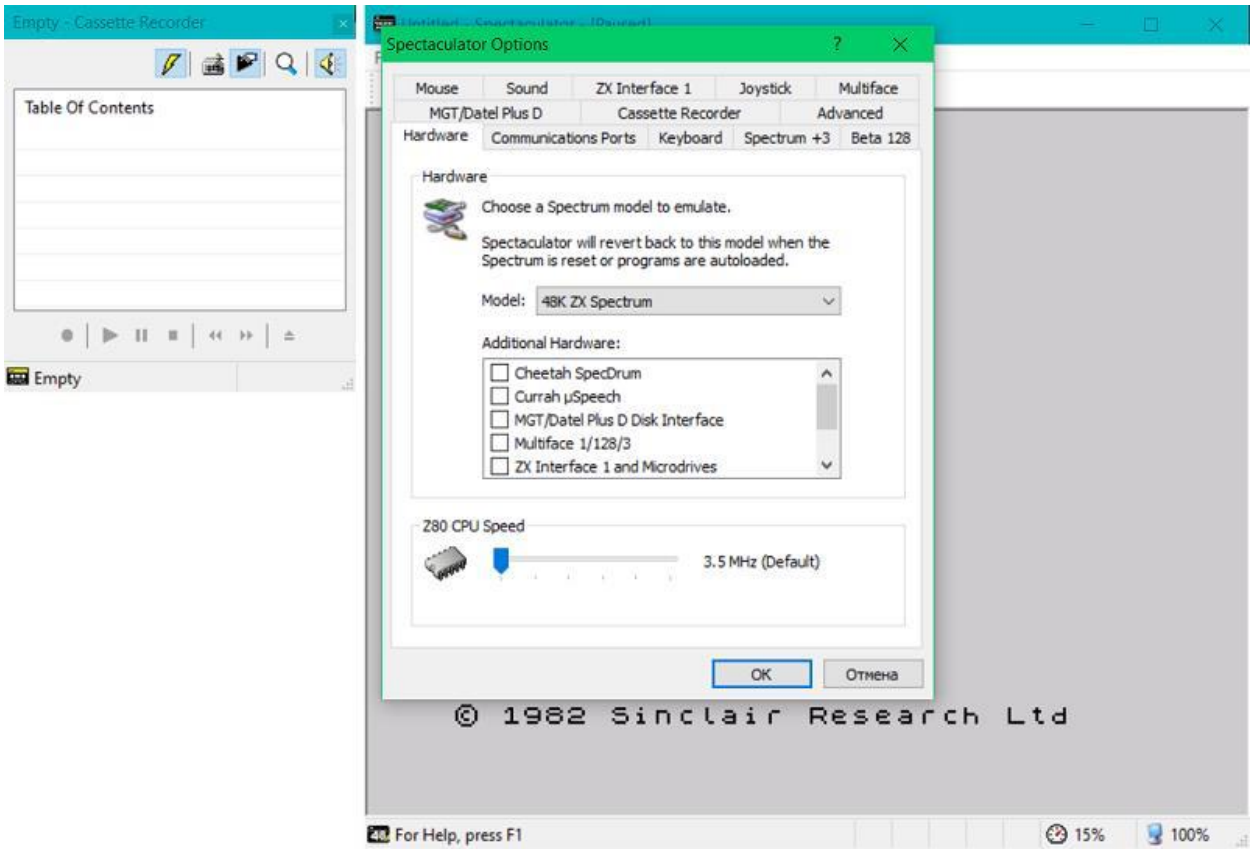


Рис. 302. Отключенные галочки на вкладке «Hardware» в окне «Spectaculator Options».

Сверху в три ряда расположены кнопки вкладок. Найдите «*Cassette Recorder*» и нажмите на неё левой кнопкой мыши:

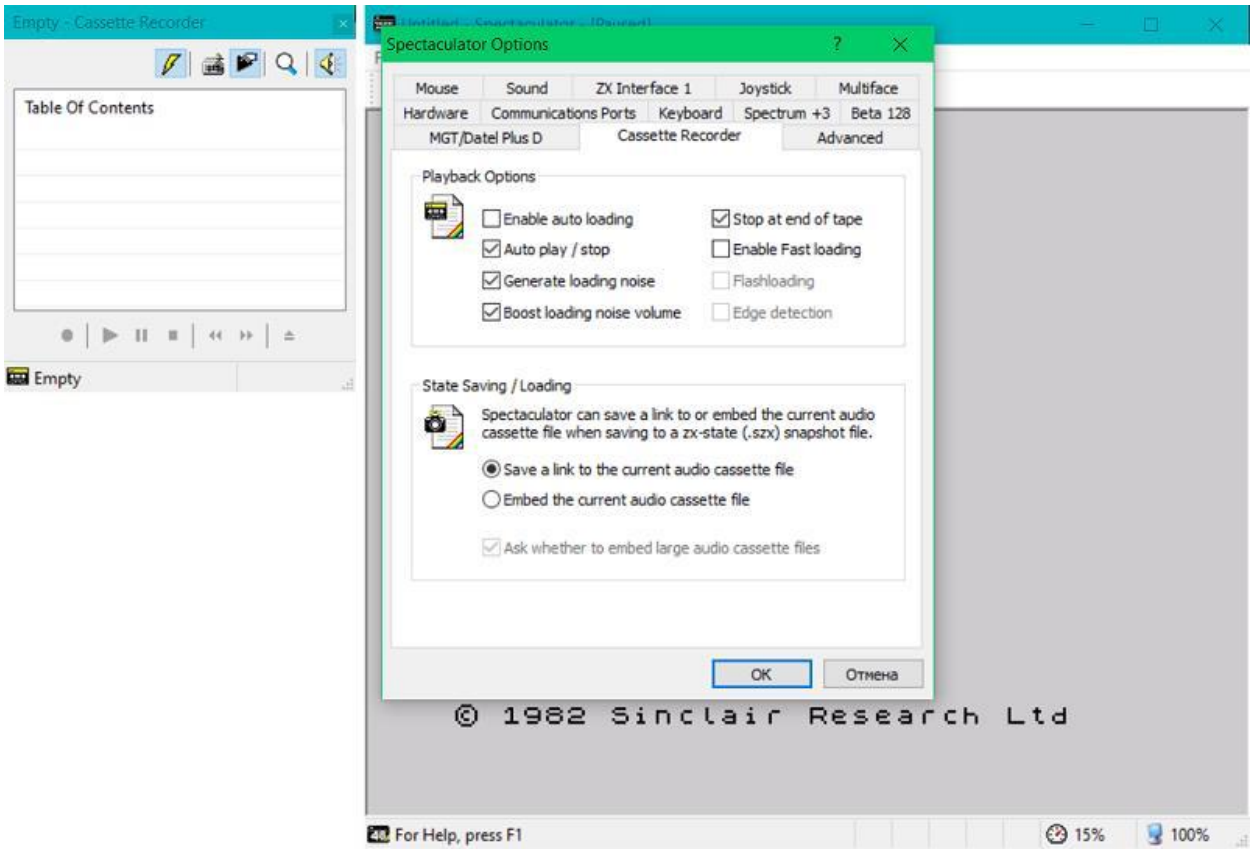


Рис. 303. Вкладка «Cassette Recorder» в окне «Spectaculator Options».

В «животе» окна появились настройки виртуального магнитофона. В разделе «*Playback Options*» в два ряда расположено 8 галочек. Чтобы окно магнитофона функционировало и пищало как положено, в правом ряду снимите галочку «*Enable Fast Loading*» и установите «*Generate loading noise*». Для усиления звукового эффекта также рекомендую включить «*Boost loading noise*».

Оставшиеся настройки можно подобрать по своему вкусу. Если желаете, чтобы магнитофон автоматически выключался после загрузки, то нужно установить «*Stop at end of tape*», а для автоматического включения кнопки «*Play*» магнитофона, после набора LOAD ""ENTER необходимо зажечь «*Auto play / stop*»:

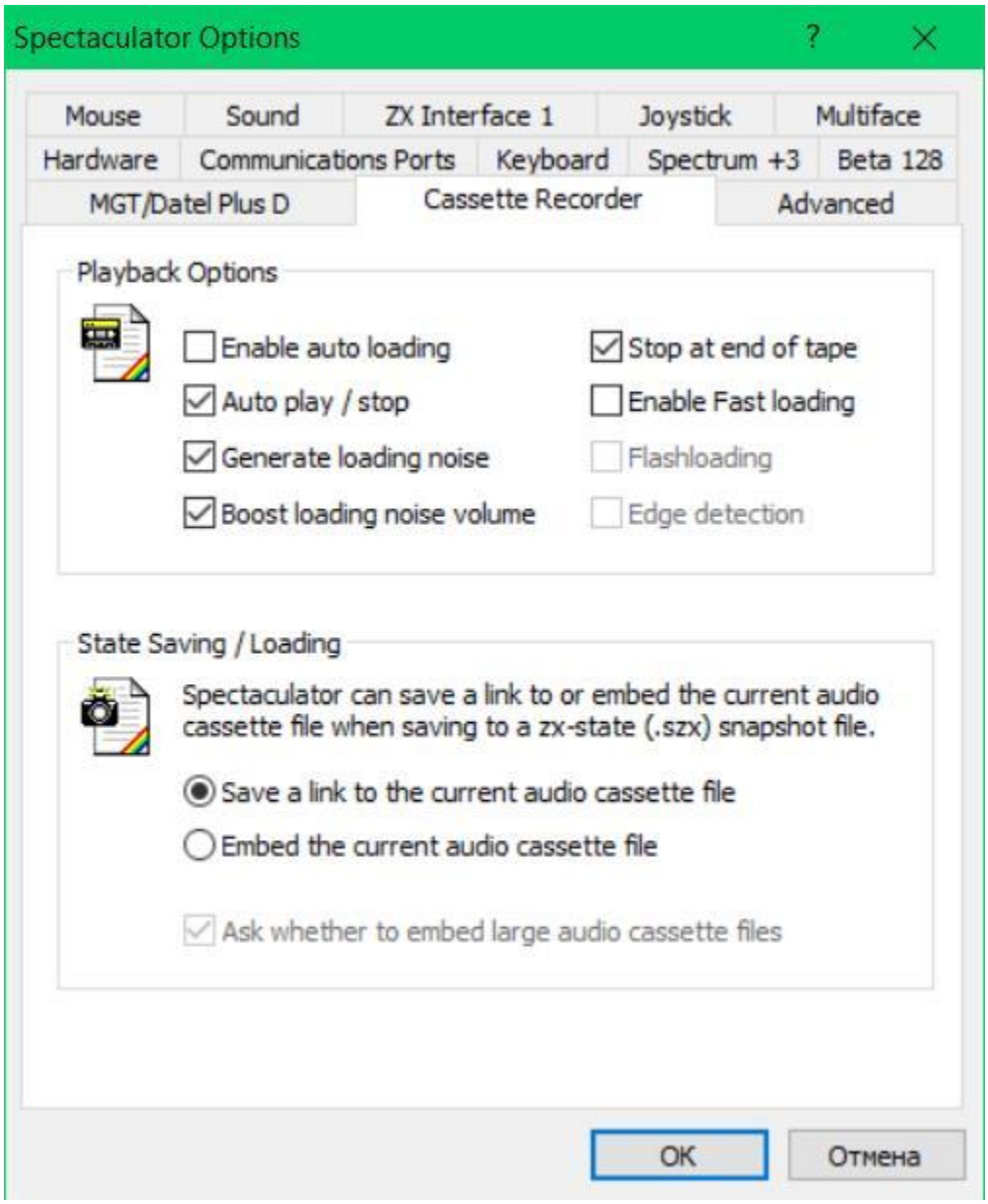


Рис. 304. Рекомендуемые настройки вкладки «Cassette Recorder».

Выставив желаемые настройки, нажмите «*OK*» внизу окна. Окно закроется и магнитофон готов к работе. Это единоразовая настройка, которую в дальнейших примерах трогать не потребуется.

Магнитофон добыт и настроен, можно начинать знакомиться с основами записи.

Глава 32
Знакомство с SAVE

Краткое содержание: SAVE-ETC

Что вообще известно о загрузке из прекрасной литературы прошлого? Учитывая качество подобных книг, о записи и считывании программ можно сделать следующий теоретический вывод.

Существует два типа данных, которые можно записать/считать с магнитной ленты. Самый первый – это заголовок «Пи-и-и-и-и-и-и-и-и-и-и Тщю-у-у-ю».

Науке известно о четырех подвидах таких блоков. В повседневной жизни наиболее часто используются заголовки «`Program:`» и «`Bytes:`». Блок «Program:» хорошо знают ~99,8% пользователей компьютера. С него начинается загрузка большинства игр.

«Bytes:» (*жарг. Бутэс или Бытес*) идёт следом за блоком «Program:» и является элементом продолжения загрузочного процесса подавляющего большинства игр.

Варианты «`Number array:`» и «`Character array:`» относят к категории экзотических видов животных, которые крайне редко встречаются в дикой природе. Они находятся на грани полного вымирания, поэтому занесены в красную книгу, дабы люди не забывали об их существовании и периодически прикладывали руку к поддержанию численности популяции.

Следом за заголовком идёт второй тип блока «Пи-и-и-и Шю-ю-у-у-ю-у-и-и…». Длина «ш-ю-у-у-у…» – традиционного метода загрузки может варьироваться от ~10 миллисекунд, до 4 минут 51,504 секунды. В зависимости от внутреннего наполнения блок имеет разную звуковую окраску и во время загрузки проигрывает занятные эксклюзивные мелодии.

Подвожу итог вышесказанного: стандартным способом, средствами подпрограмм BASIC, можно записать 5 типов работоспособных блоков:

0) Program:
1) Number array:
2) Character array:
3) Bytes:
4) Беззаголовочные произвольные данные

Вот и вся научно-популярная теория, которая известна до настоящего момента. Но за окном осень 2024 года, поэтому пора пересмотреть эти знания, а возможно, что-то опровергнуть. Разбор предлагаю начать с подпрограммы, которая формируют команду `SAVE`.

Предположим, вы хотите записать картинку. После ввода команды `SAVE "ХРЕНОТЕНЬ"CODE 16384,6912` Стрелочка попадает на распознавание строки, а оттуда на многоцелевой комплекс программ загрузки/записи «`SAVE-ETC`» по адресу 1541. Там выделяется место в рабочей области (1569) и сразу за маркером конца вводимой строки [`WORKSP+00`] формируется шаблон заголовка из 11-ти пробелов (1573-1579):

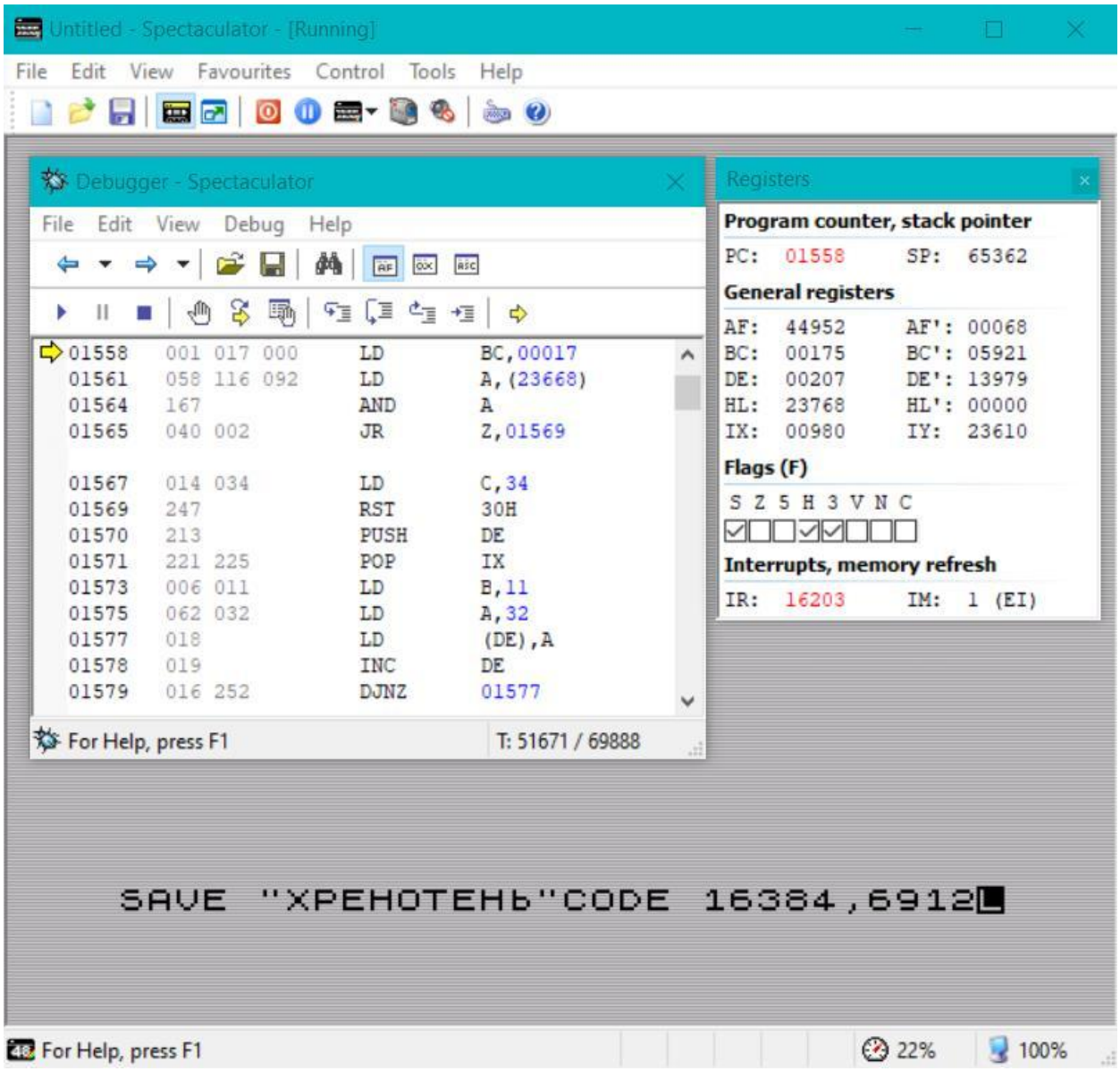


Рис. 305. Формирование шаблона заголовка в «SAVE-ETC».

Далее в него из кавычек добавляется имя:

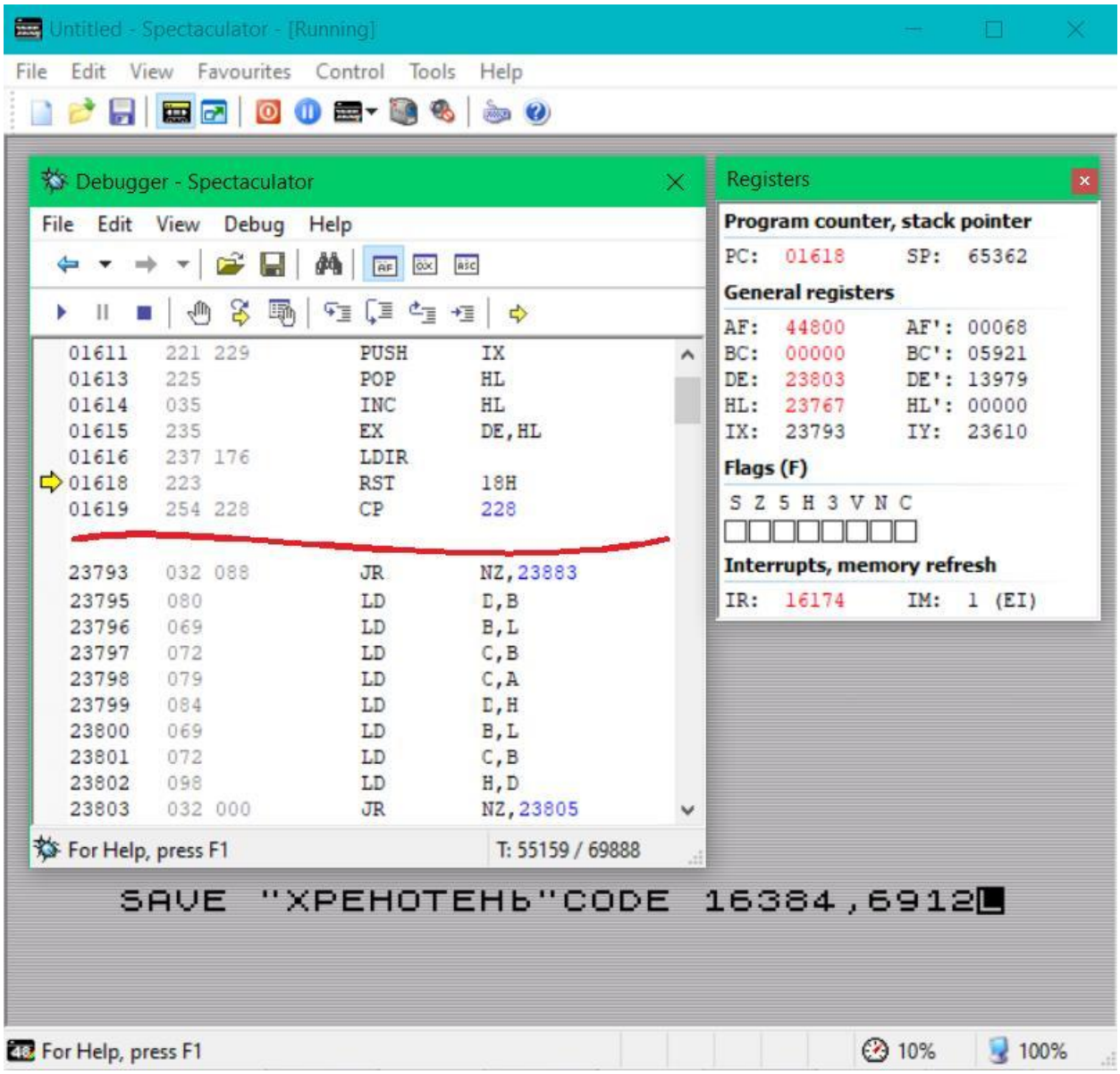


Рис. 306.

На заключительной стадии формирования параметров, из строки извлекается длина (1794) и стартовый адрес (1803), после чего он переписывается в «HL», а в заглавный байт заголовка помещается тип блока «3» (1818):

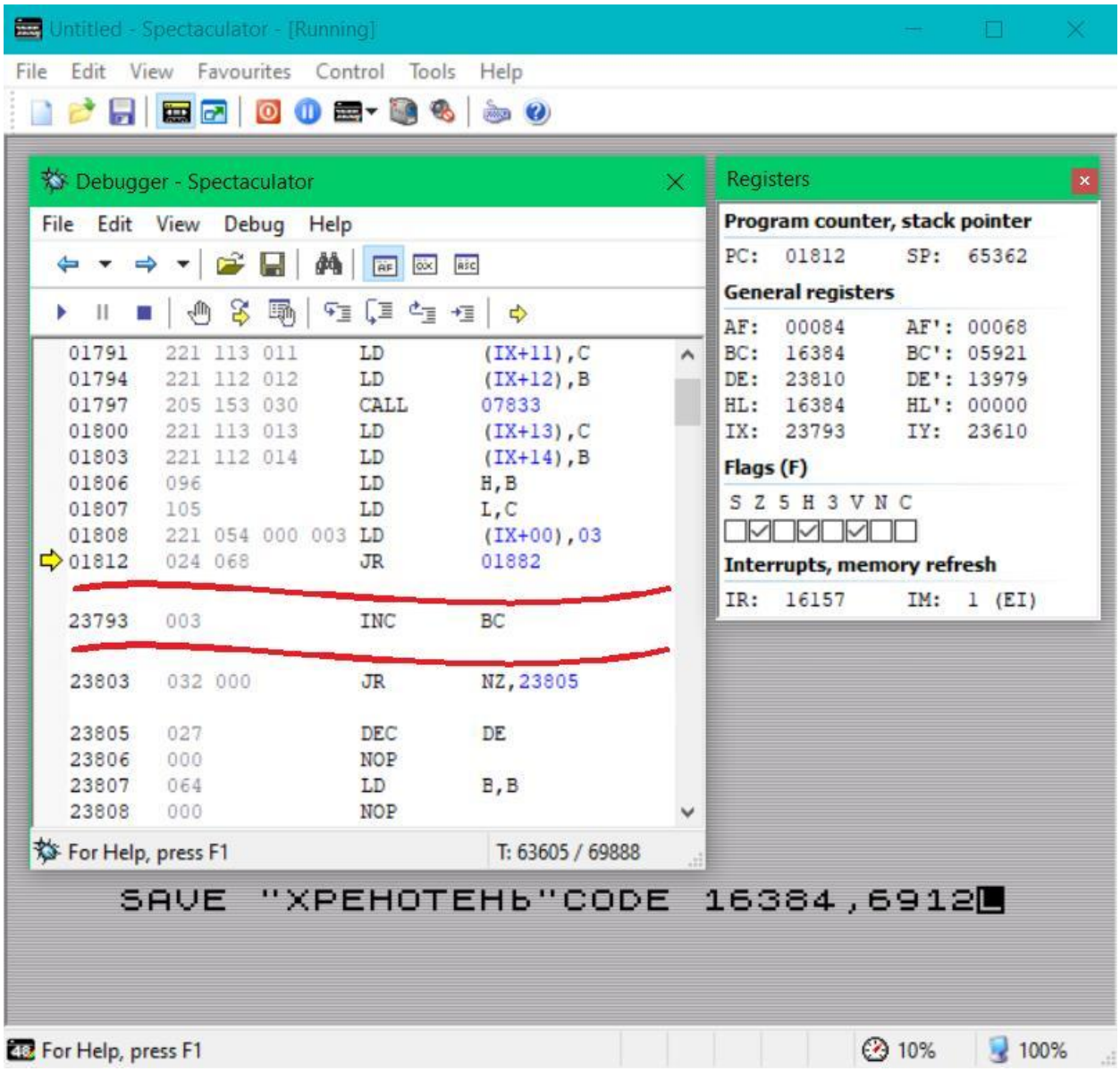


Рис. 307.

После всех перипетий Стрелочку приносит в комплекс программ `SAVE (2416)` на подпрограмму `SA-CONTR (2416)`. В ней вся суть:

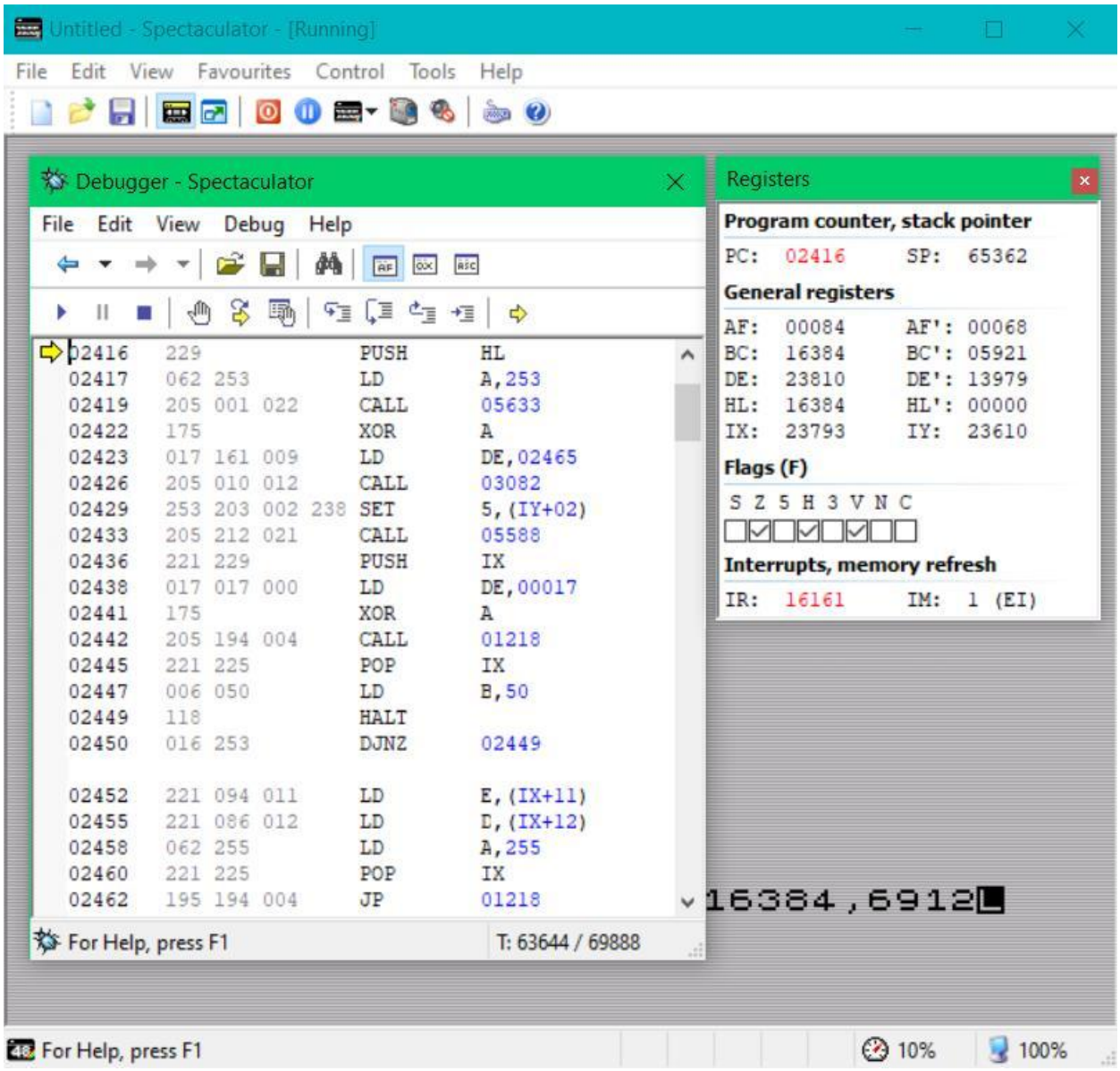


Рис. 308. Подпрограмма генерации записи «SA-CONTRL».

Первым делом происходит сохранение адреса начала записываемых данных, после чего подготавливаются нижние строки экрана для вывода сообщения. Выбрав из библиотеки сообщение «`Start tape, then press any key.`», методом `PO-MSG (3082)` оно печатается в двух нижних строках. Как правило, это часто вызывает негодование, потому что после сообщения запоганивается низ записываемой картинки.

Для подавления мигающего курсора включается 5-й бит `TV_FLAG (23612)` и вызывается обычный BASIC-редактор по адресу 5588, и все замирает.

После нажатия [почти] любой клавиши сохраняется адрес местонахождения шаблона для заголовка, а это самое начало области `WORKSP`. В «DE» выставляется его фиксированная длина 17 байт. Обнуляется «A», показывая, что для заголовка нужно выставить длинный сигнал «Пи-и-и-и-и», и отправляется на подпрограмму записи `SA-BYTES` по адресу 1218.

Возвратившись после записи заголовка, вспоминается адрес размещения шаблона, и перед записью данных организуется секундная пауза. (50 HALT'ов по 69888 волшебных несекунд + пара сотен сверху от выполнения команд).

Далее из 12-го и 13-го байта только что записанного шаблона берется длина блока. В «A» указывается 255, означая, что будет блок данных с коротким сигналом «Пи-и-и». Выставив в «IX» адрес точки, откуда нужно начать запись, происходит повторный заход

на программу записи `SA-BYTES (1218)`. На этот раз уже безвозвратно, потому что оттуда через адрес 7030 Стрелочка начнет возвращаться на поверхность.

Для разминки, этой теории вполне достаточно, и можно смело переходить к практике.

Глава 33
Синтезированная запись комплекта «Bytes:» на виртуальную кассету

Краткое содержание: файл .tzx, виртуальный магнитофон, форматы записи

Предлагаю начать с классики и синтезировать результат работы следующей строки:

```
SAVE "Capaumoneg"CODE 40000,46
```

Иными словами, записать комплекс «`Bytes:`» с содержимым в полной комплектации, включая сообщение: «`Start tape, then press any key.`»

Первым делом следует раздобыть виртуальную магнитофонную кассету, на которую можно будет записать программу. В данном случае всё просто. Не нужно бежать на Удельный рынок, «Юнону» или «Avito». Достаточно просто создать файл формата *.tap* или *.tzx* и открыть его для записи.

В левом верхнем углу окна нажмите на пункт меню «*File*»:

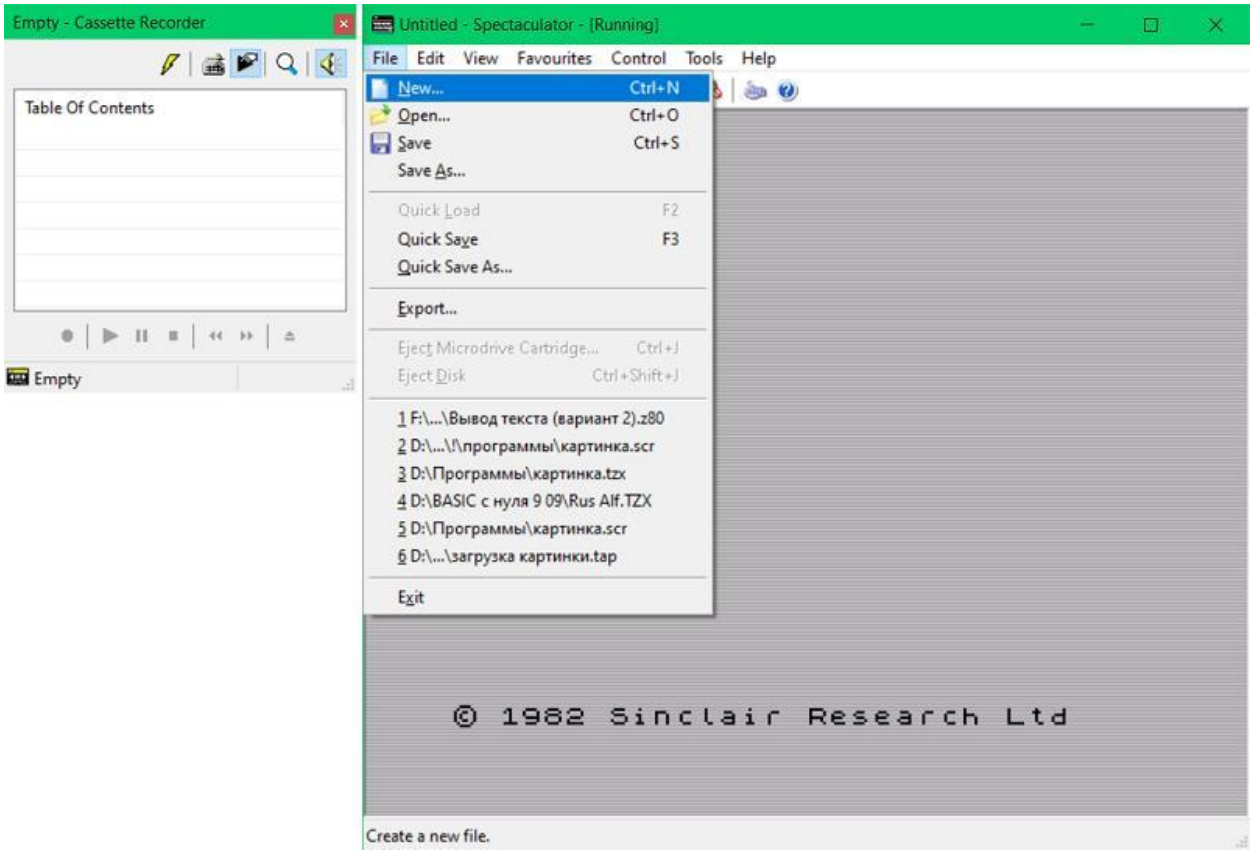


Рис. 309. Пункт «New» меню «File». Процесс создания виртуальной магнитофонной кассеты.

В нем откройте самый первый пункт «*New…*»:

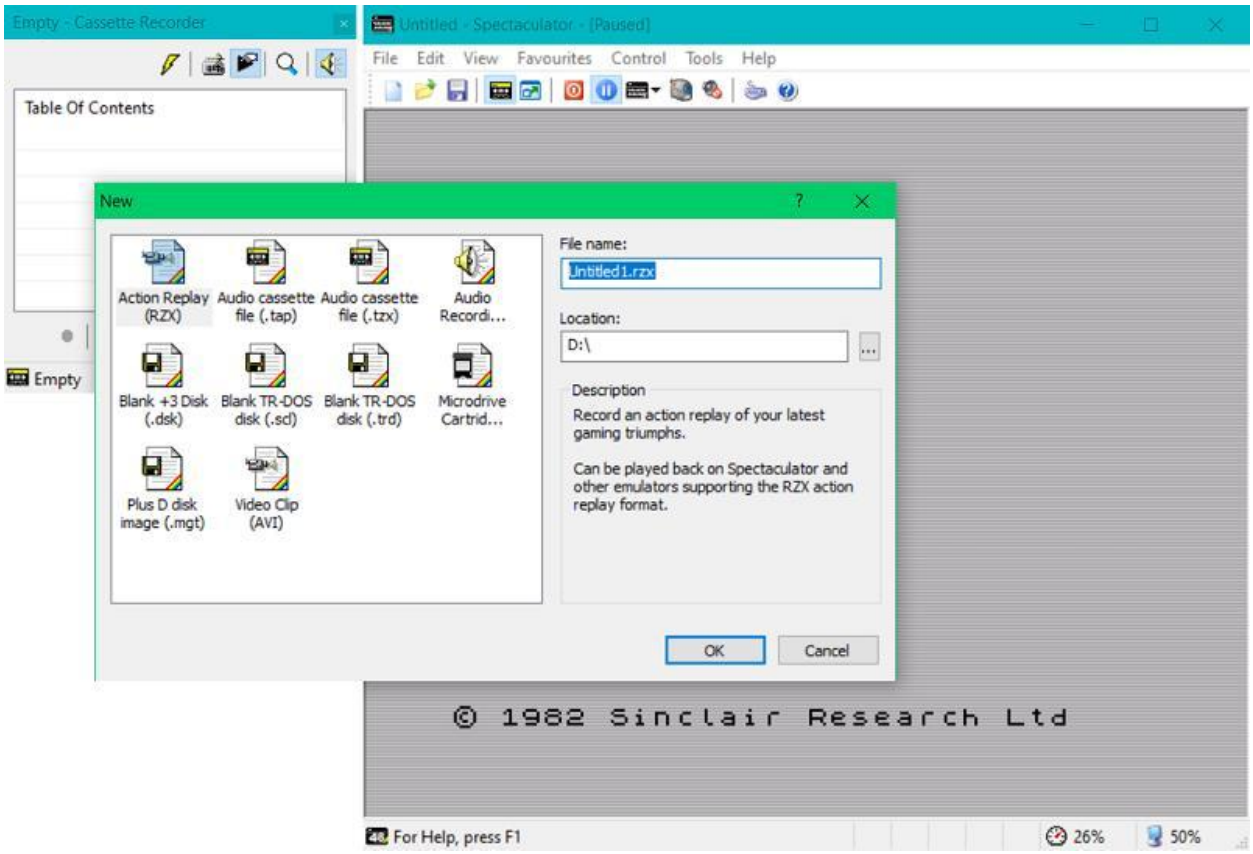


Рис. 310. Окно «New». Шаблоны форматов записи.

Появится одноимённое окно «*New*» с шаблонами форматов. Прежде всего, выберите путь, куда будут писаться файлы. Для этого на своём компьютере создайте папку «*Программы*». Слева от белой полоски «*Location*» нажмите кнопочку с тремя точками. В открывшемся окне выберите нужную папку или создайте новую:

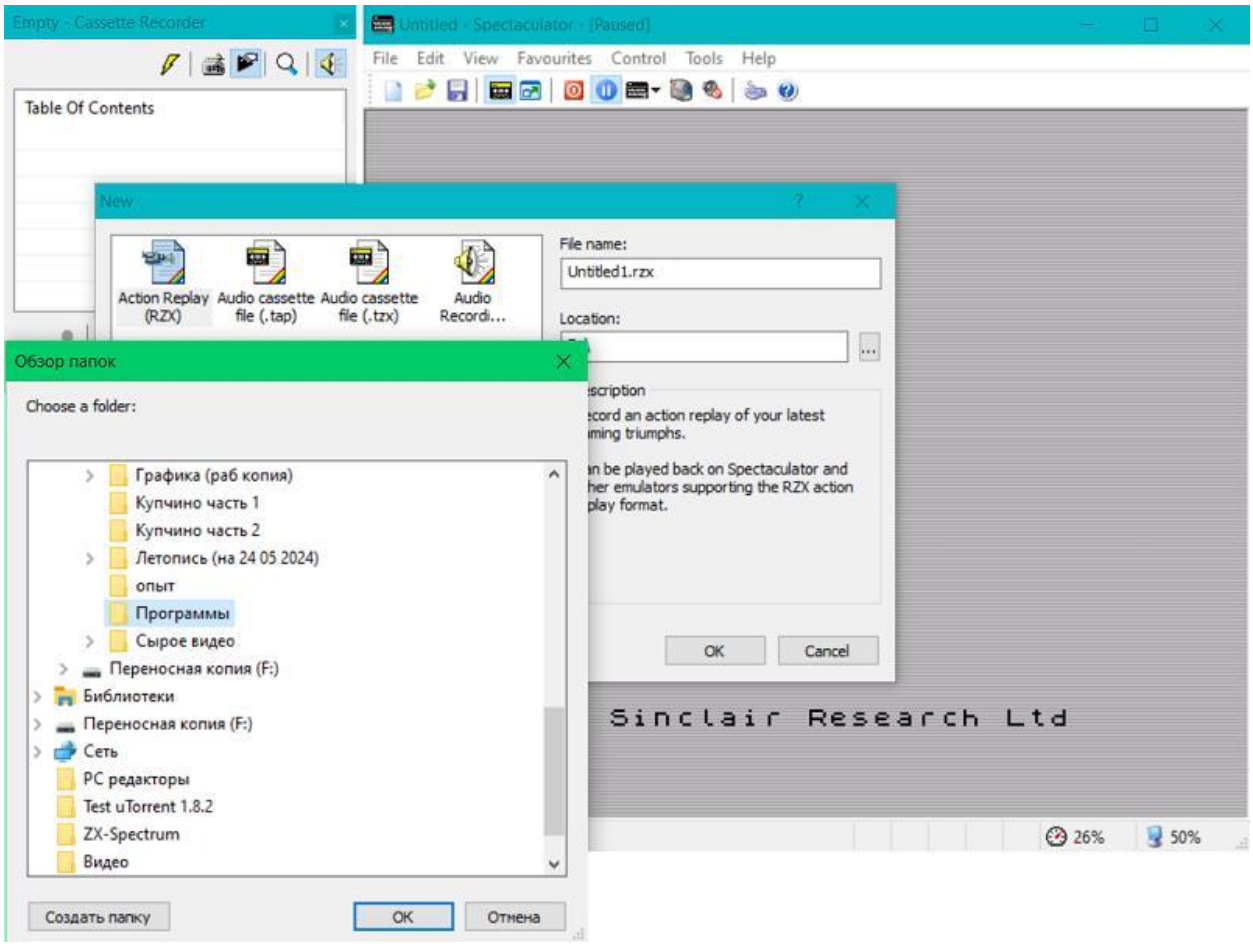


Рис. 311. Выбор пути для создаваемого файла виртуальной кассеты.

После выбора папки нажмите «*OK*». Окно закроется и в полоске «*Location*» появится путь с папкой. Нажмите на тип «*Audio cassette file (.tzx)*». В белой полосе «*File name*» введите желаемое имя, например «*Монед в сарае*». Должно получиться так:

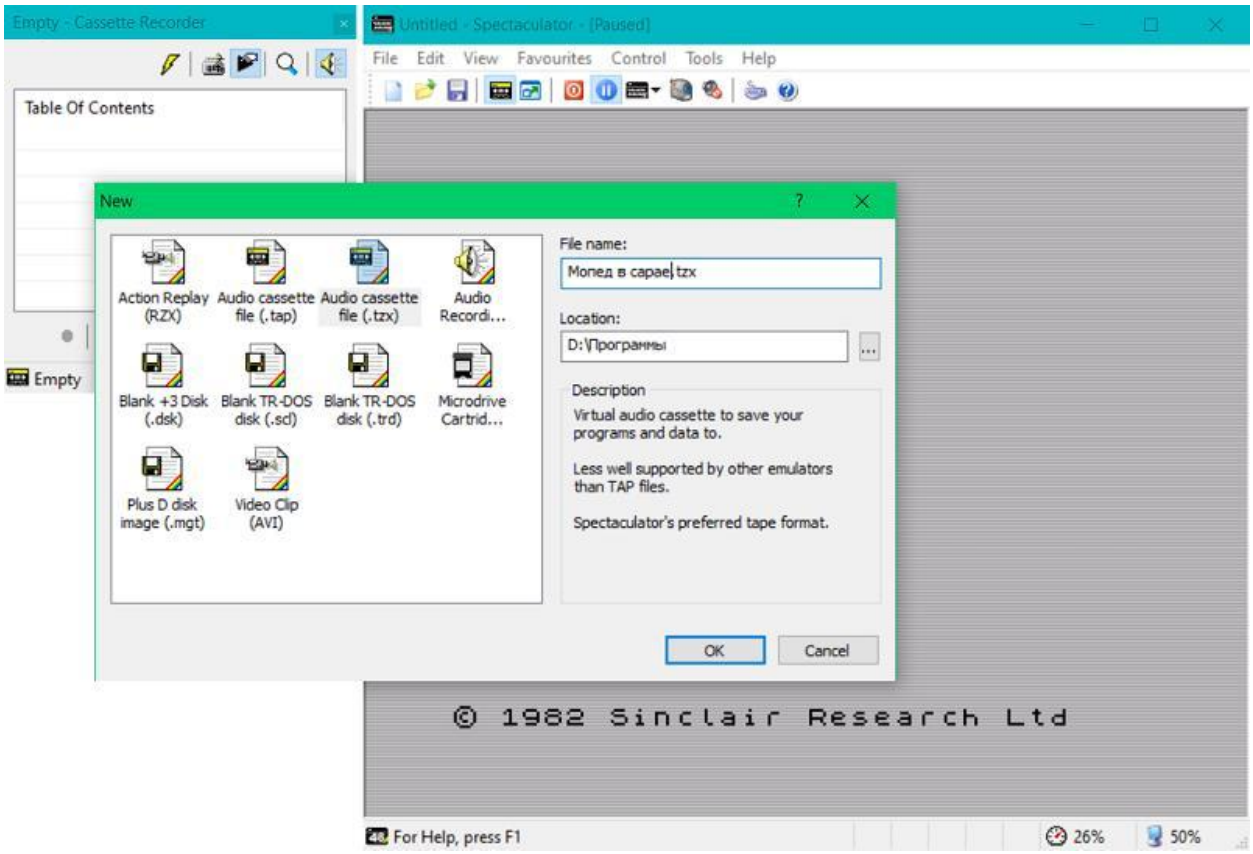


Рис. 312. Задание имени файлу виртуальной кассеты.

Нажимайте «*OK*». Окно закроется, а в виртуальном магнитофоне покраснела и активировалась кнопочка  «*REC*». Магнитофон заряжен новенькой пленочкой и готов к записи, но пока стоит в режиме ожидания «*Stopped*»:

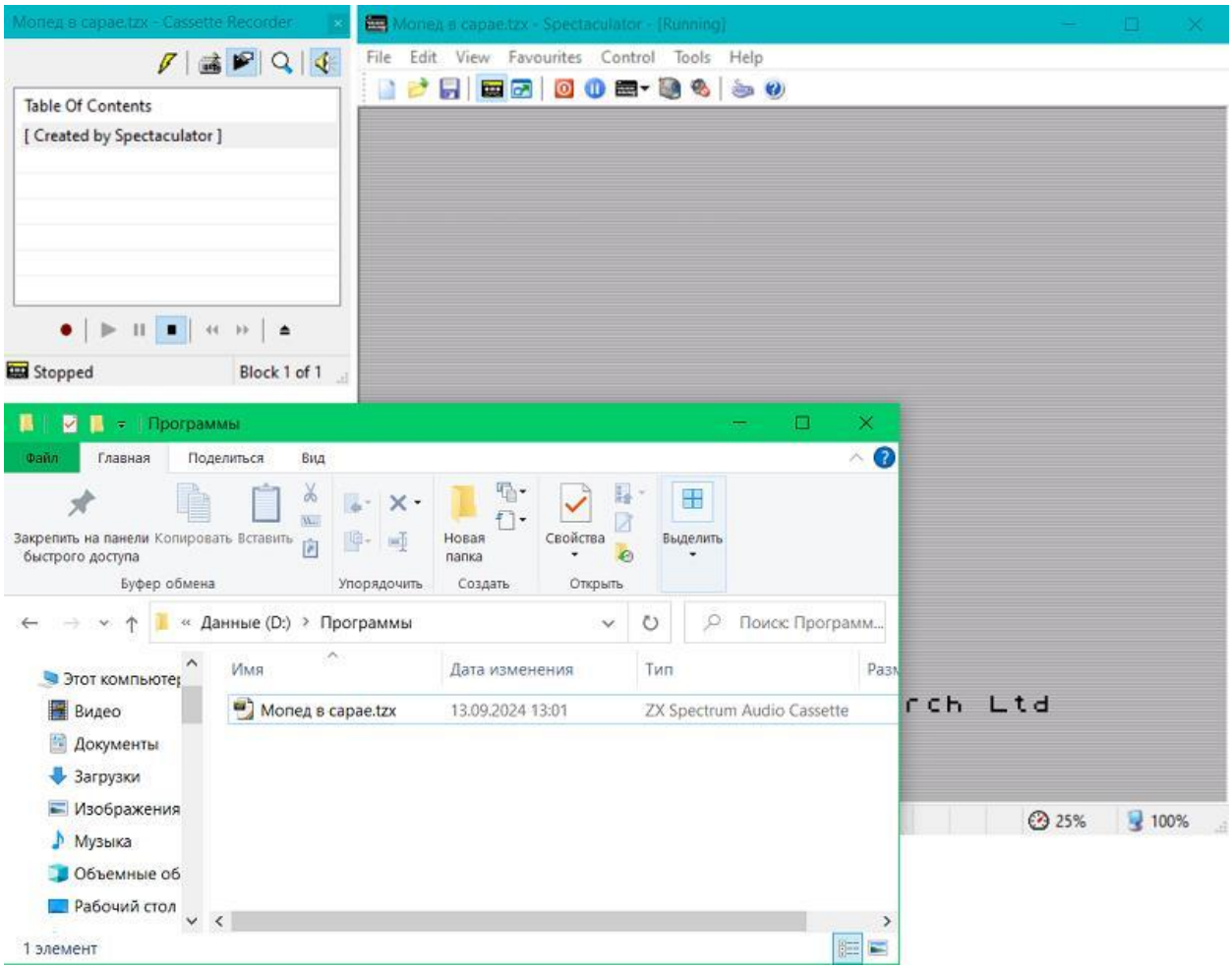


Рис. 313. Появление кассеты в окне виртуального магнитофона.

Нажмите кнопочку «*REC*», и под магнитофоном появится надпись «*Recording*». Не переживайте, без специальной команды вхолостую ничего записываться не будет:

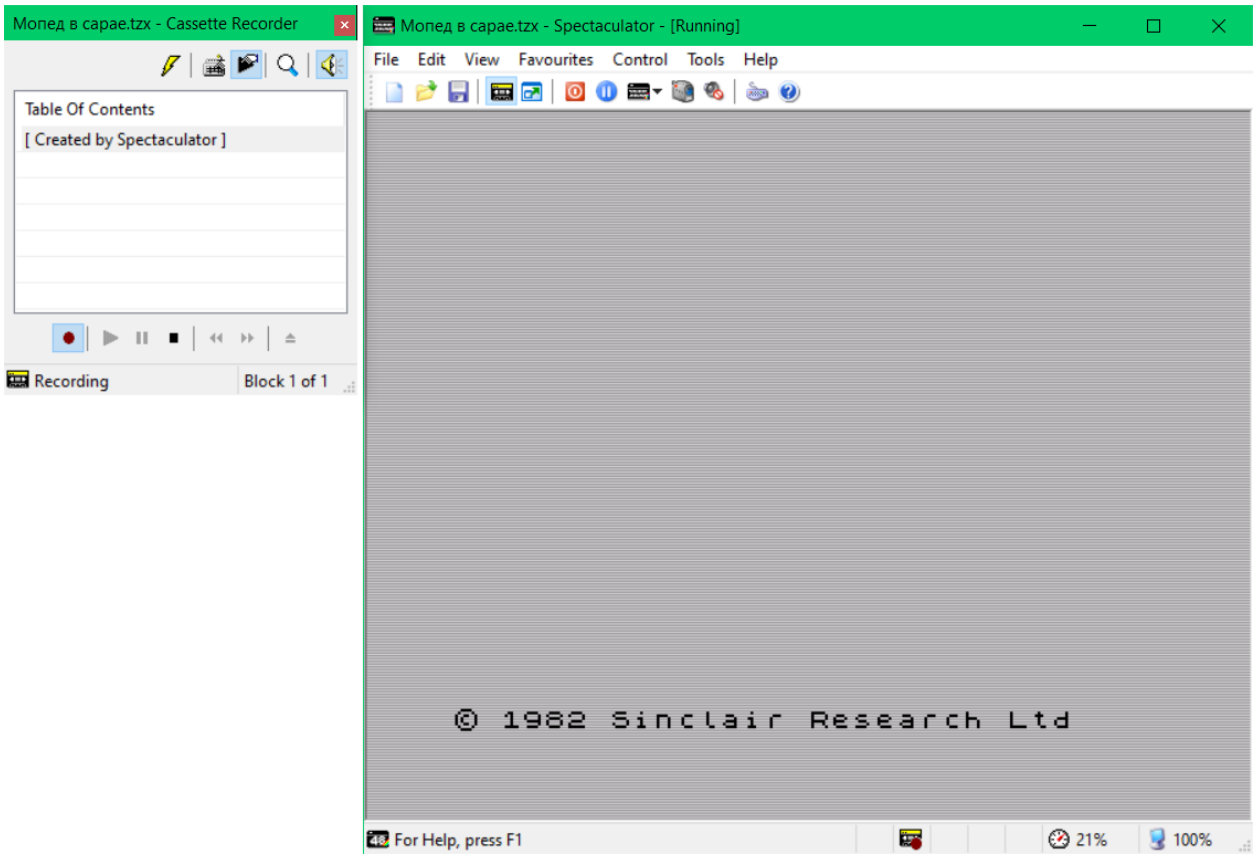


Рис. 314. Включение виртуального магнитофона в режим записи «REC».

Оставьте магнитофон включенным и заходите в «*Debugger*». Обновите экран, чтобы, замигал курсор K:

```
Debugger
23560 ← 13
23611 ← 32
Trace
```

Можно приступать к созданию заголовка. Что о нём известно? Любой заголовок имеет длину 17 байт, ну а чтобы его записать, нужно сначала где-то выложить. Пусть это будет адрес буфера принтера 23400, чтобы не засорять основные ячейки памяти. Сама программа будет располагаться с адреса 40000.

А теперь наберите и введите следующий алгоритм:

```
Debugger
Dec
Go To 23400
23400 ← 3
23401 ← CapauMoneg
23411 ← 46 40000
23415 ← 0 0

23610 ← 255
23613 ← 65364
65364 ← 4867

40000 ← 253 203 2 134 17 77 156 1 33 0 195 60 32
40013 ← 22 11 2 16 2
40018 ← B Capae y Cocega Cmoum Moneg

HL ← 40000
```

```
IX ← 23400
SP ← 65364
PC ← 2416
Trace
```

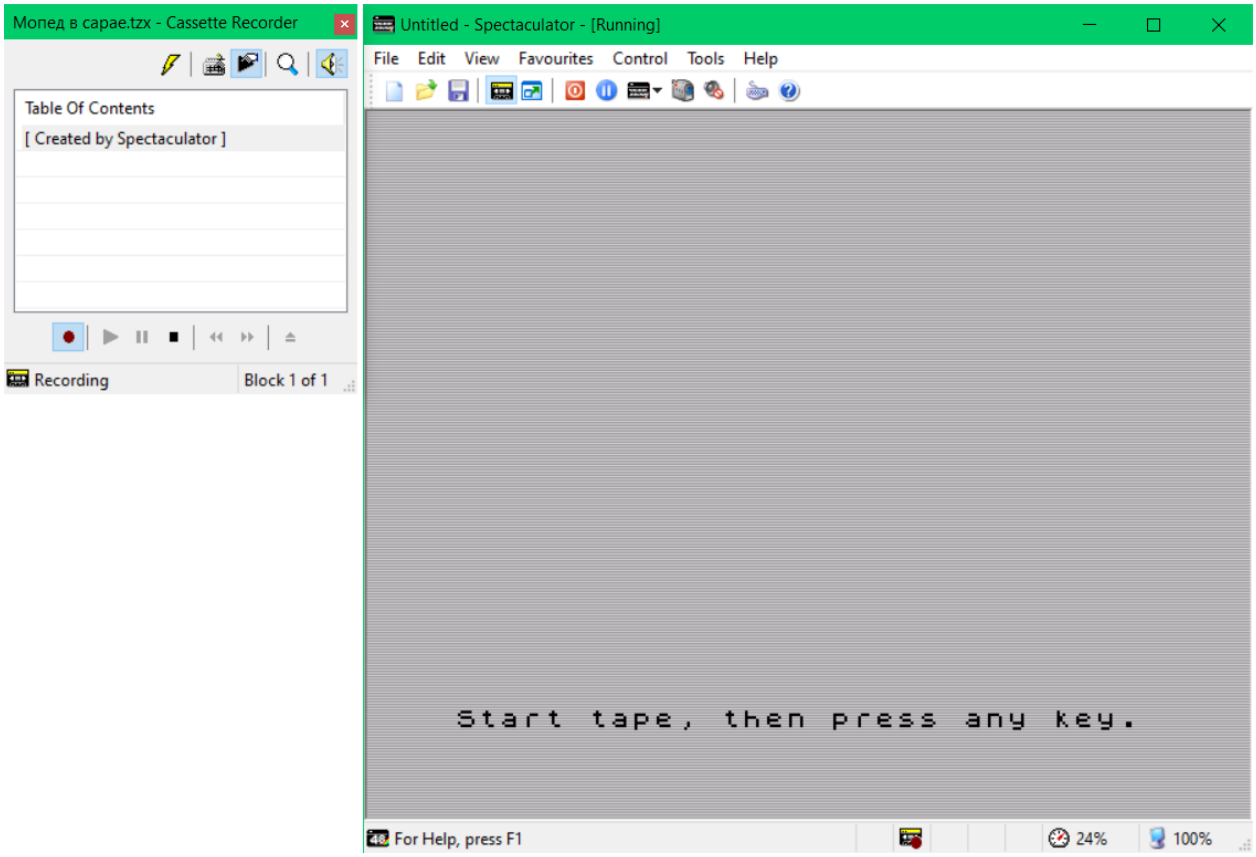


Рис. 315. Ожидание процесса записи.

На экран выскочила знакомая заставка. Нажимайте ENTER. После ввода, программа на секунду задумалась, после чего в виртуальном магнитофоне появилось два блока:

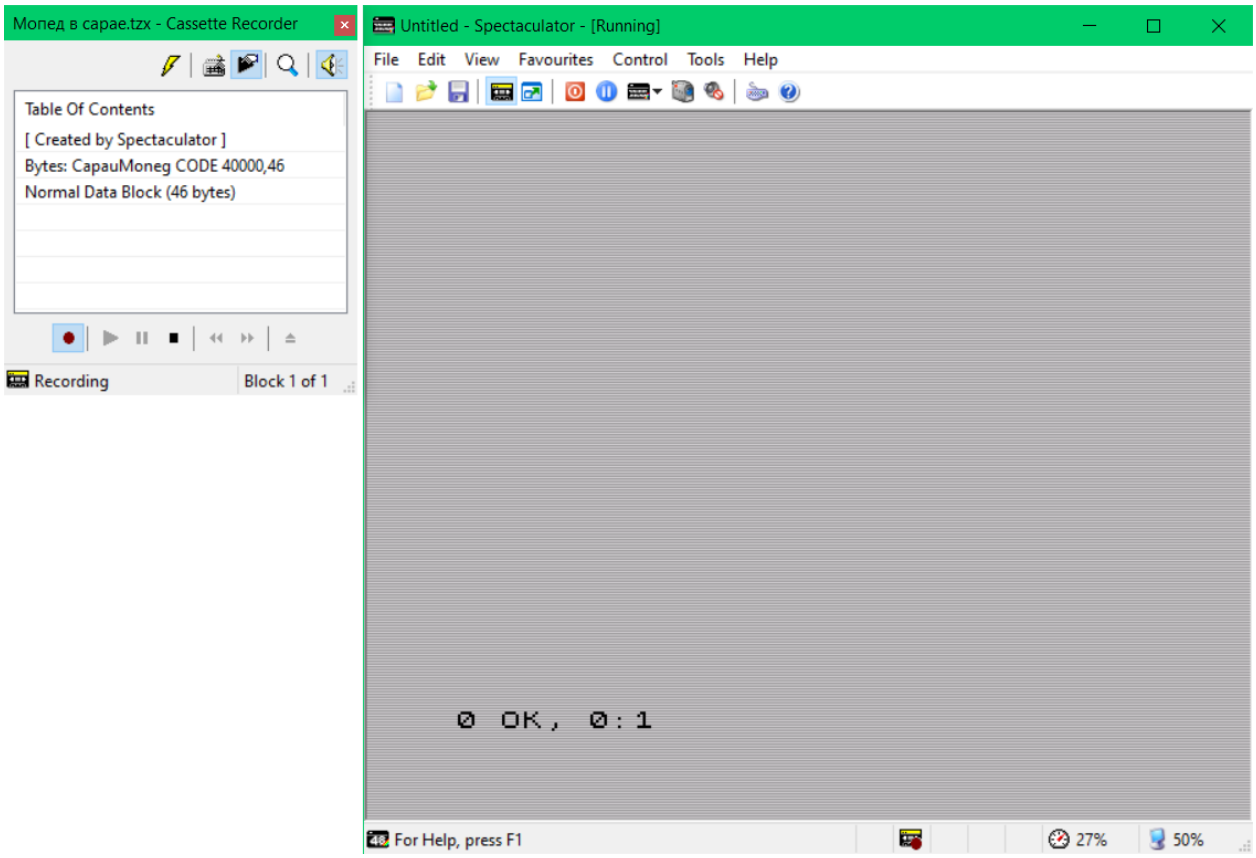


Рис. 316. Окончание записи комплекта «Bytes:»

Нажмите кнопку «*Stop*» с квадратиком ■, пленка остановится и в левом нижнем углу окна, рядом с кассетой появится надпись «*Stopped*». Ваша первая программа сохранена:

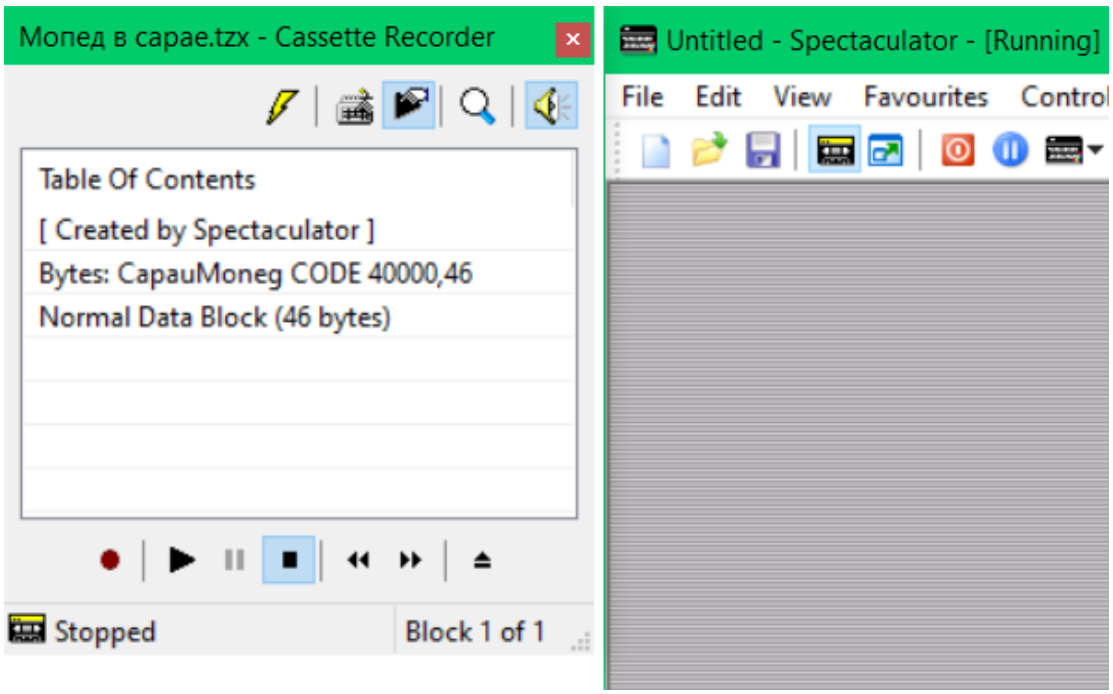


Рис. 317. Остановка записи на кассету в виртуальном магнитофоне.

А теперь предположите, что у вас кончилось место на кассете. Её нужно вынуть и вставить новую. Найдите на магнитофоне кнопочку ⏏ «Eject» и нажмите:

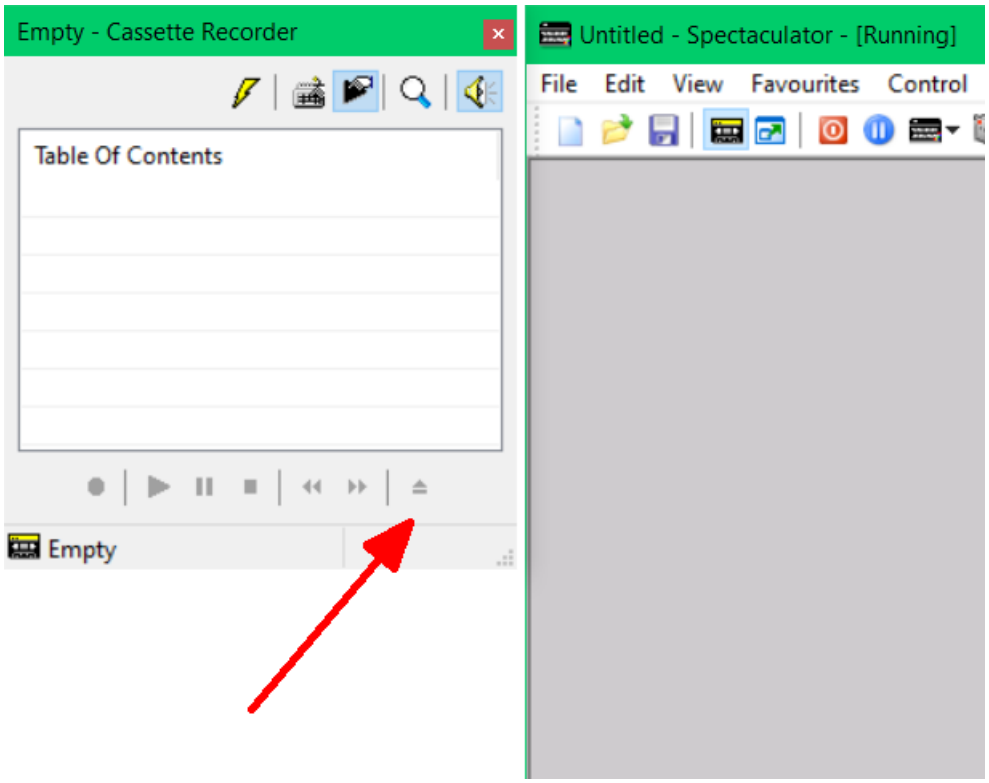


Рис. 318. Извлечение кассеты из виртуального магнитофона.

Виртуальный магнитофон очистился. Можно подводить итоги. Конечный и полный алгоритм программы записи программы, будет следующим:

```
New File [Мопед в сарае.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 23400
23400 ← 3
23401 ← CapauMoneg
23411 ← 46 40000
23415 ← 0 0

23610 ← 255
23613 ← 65364
64364 ← 4867

40000 ← 253 203 2 134 17 77 156 1 33 0 195 60 32
40013 ← 22 11 2 16 2
40018 ← В Сарае у Сосега Смоум Moneg

HL ← 40000
IX ← 23400
SP ← 65364
PC ← 2416
Trace
Cassette Recorder [Stop]
Cassette Recorder [Eject]
```

С этой главы добавились четыре новые команды:

«*New File [*. tzx]*» - создать и открыть для записи новый файл .tzx с требуемым именем (*Ctrl+N → Audio Cassette File (*.tzx)*)

«*Cassette Recorder [Record]*» - включить виртуальный магнитофон на запись кнопочкой  или нажатием комбинации клавиш «*Ctrl+Alt+Space*».

«*Cassette Recorder [Stop]*» - Остановить запись или воспроизведение на кассету виртуального магнитофона кнопочкой  или нажатием комбинации клавиш «*Ctrl+Space*».

«*Cassette Recorder [Eject]*» будет означать окончательное формирование и закрытие файла.

Можно переходить к следующей главе и создавать новую кассету.

Глава 34
Запись «Bytes:» с сохранением нижних строк

Краткое содержание: запись картинки с нижними строками, .scr

Предлагаю немного усложнить задачу и сохранить картинку с нижними строками. Для этого блок «*Bytes:*» придётся записать без вывода стандартного сообщения.

В любом внешнем редакторе, например в «*ZX-Paintbrush*» создайте полноэкранное изображение размером 256x192 и сохраните его под любым именем, например, «*Логотип КС3-1998.scr*»:

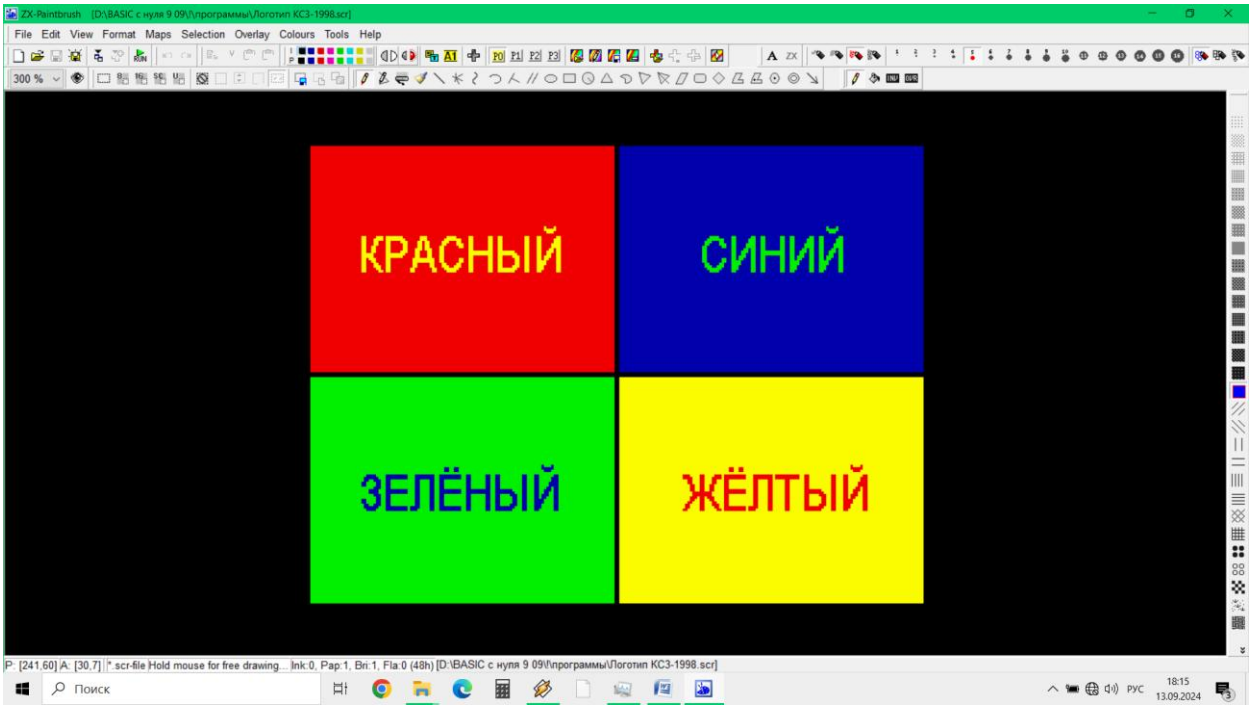


Рис. 319. Создание рисунка в редакторе «ZX-PaintBrush».

Если лень разбираться и рисовать свою картинку, можно взять мою готовую из комплекта к книге или скачать из любой игры. Как рассказывалось в главе «Апокалиптичная *графика*», просто перетащите её на экран «НеСпектрума»:

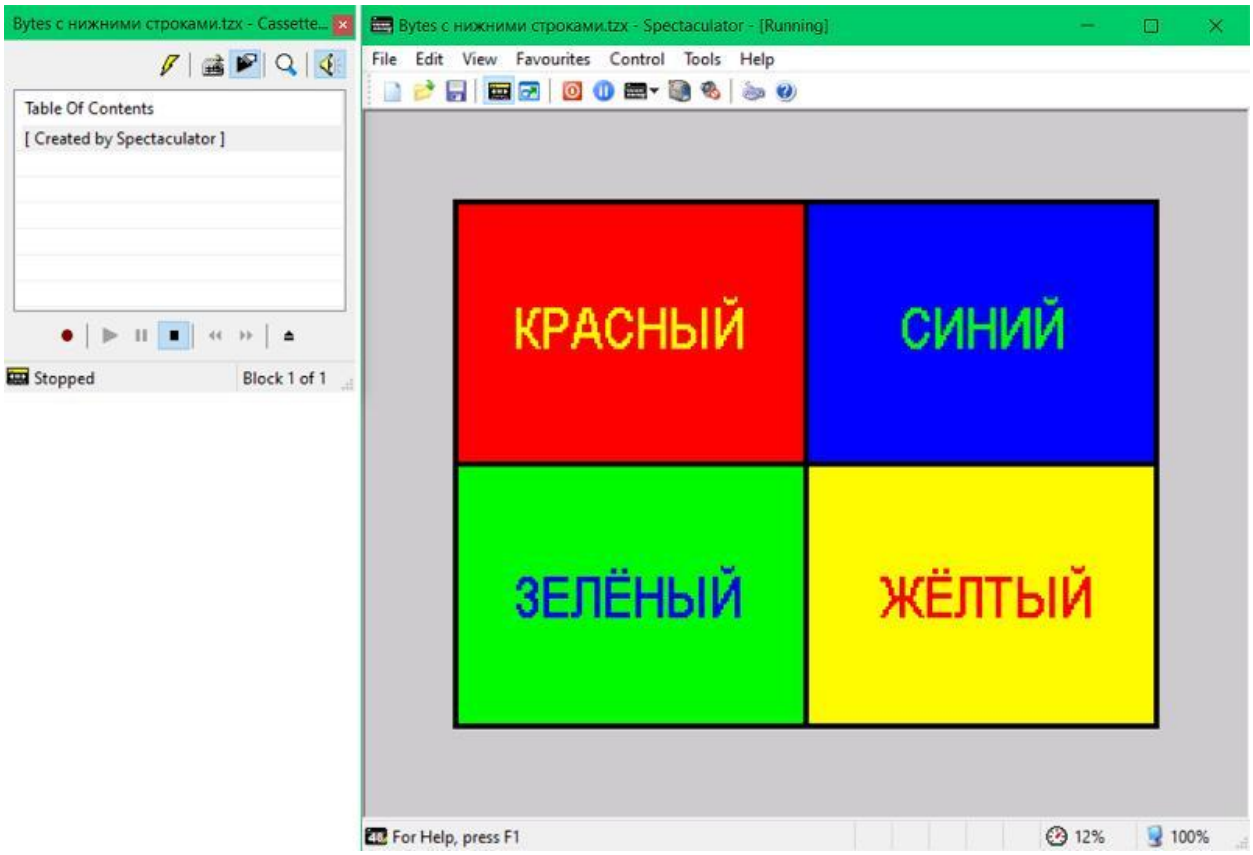


Рис. 320. Загрузка внешней картинки на экран «Spectaculator».

Осталось только записать творение на магнитную ленту. Создайте новый файл *.tzx* На виртуальном магнитофоне снова нажмите кнопочку ● «*REC*»

А теперь введите алгоритм (понятно, что уже без первых трёх строчек):

```
Open File [Логотип КС3-1998.scr]
New File [Bytes с нижними строками.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 23400
23400 ← 3
23401 ← КС3-*1998*
23411 ← 6912 16384
23415 ← 0 0

23610 ← 255
23613 ← 65364
65362 ← 16384 4867

IX ← 23400
SP ← 65362
PC ← 2436
Trace
Cassette Recorder [Stop]
Cassette Recorder [Eject]
```

Без предупреждения, программа запустилась, и на магнитной ленте снова появилось два блока:

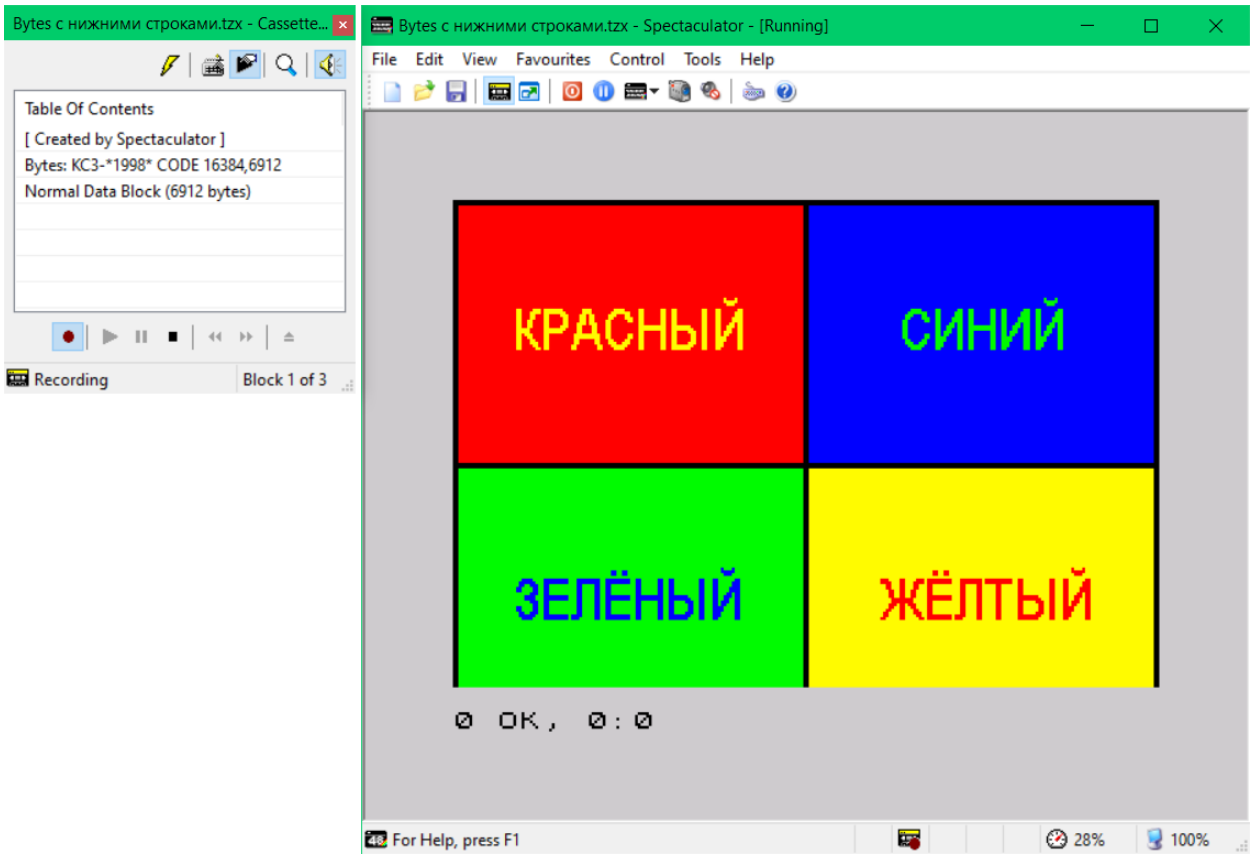


Рис. 321. Процесс записи на виртуальную кассету.

Опять появляется новая команда:

«*Open File [Логотип КС3-1998.scr]*» - загрузить *.scr* файл перетащив его мышкой на экран «*Spectaculator*» или после нажатия «*Ctrl+O*» выбрав соответствующий файл.

Глава 35
Запись заголовков и блоков по отдельности

Краткое содержание: заголовок, данные, раздельная запись, SA-BYTES

Во всех попавшихся самоучителях по ассемблеру, любят приводить пример записи блока данных без заголовка, но почему-то никогда не рассматривают запись заголовка и блока данных по отдельности. Нужно срочно исправлять эту несправедливость. Создайте новый файл виртуальной магнитофонной кассеты «*Заголовок и данные.tzx*», как рассказывалось в позапрошлой главе.

Следующая программа запишет на псевдомагнитную ленту одиночный заголовок.

```
New File [Заголовок и данные.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 23400
23400 ← 3
23401 ←B KOMAPOBO
23411 ← 50 40000
23415 ← 0 0

23610 ← 255
23613 ← 65364
65364 ← 4867
```

```
AF ← 0                  ; сигнал "Пи-и-и" (0-длинный 255-короткий)
DE ← 17                 ; длина блока заголовка
IX ← 23400              ; место размещения заголовка
SP ← 65364
PC ← 1218
Trace
Cassette Recorder [Stop]
```

Набрав и выполнив эту программу, в магнитофоне появится одиночный заголовок.

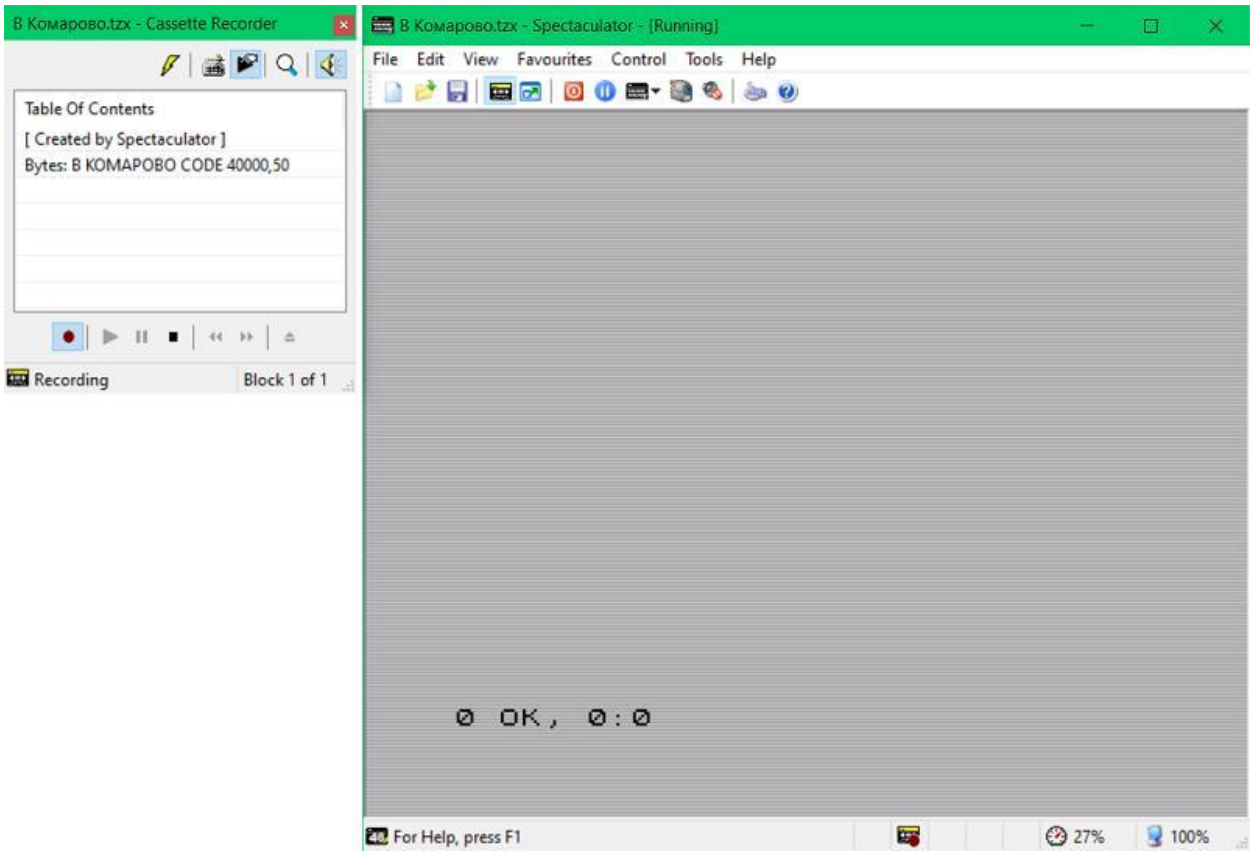


Рис. 322. Запись одиночного заголовка «Bytes:».

Да это так. Подпрограмма `SA-BYTES (1218)` спокойно записывает одиночный заголовок. В «AF» вы задаёте шаблон длины пилот тона «Пи-и-и» или «Пи-и-и-и-и-и-и». Обнуляя «AF», заголовок скомплектуется длинным сигналом. Сделав A=255, следовательно, сдвоенный «AF» должен быть задан 255*256=65280, блок запишется с коротким сигналом. В «DE» выставляется длина данных, а в «IX» стартовый адрес, откуда их считать.

Следом за заголовком предлагаю дописать сам блок данных. Для этого нажмите кнопочку «*REC*» и магнитофон готов продолжать запись следующего блока за предыдущим:

```
Cassette Recorder [Record]
Debugger
Dec
Go To 23610
23610 ← 255
23613 ← 65364
65364 ← 4867

40000 ← 253 203 2 134 17 77 156 1 37 0 195 60 32
40013 ← 22 11 1 17 6 16 1
```

```
40020 ← В КОМАРОВО НА ОЗЕРЕ ОХРЕНЕННО!

AF ← 65280
DE ← 50
IX ← 40000
SP ← 65364
PC ← 1218
Trace
Cassette Recorder [Stop]
Cassette Recorder [Eject]
```

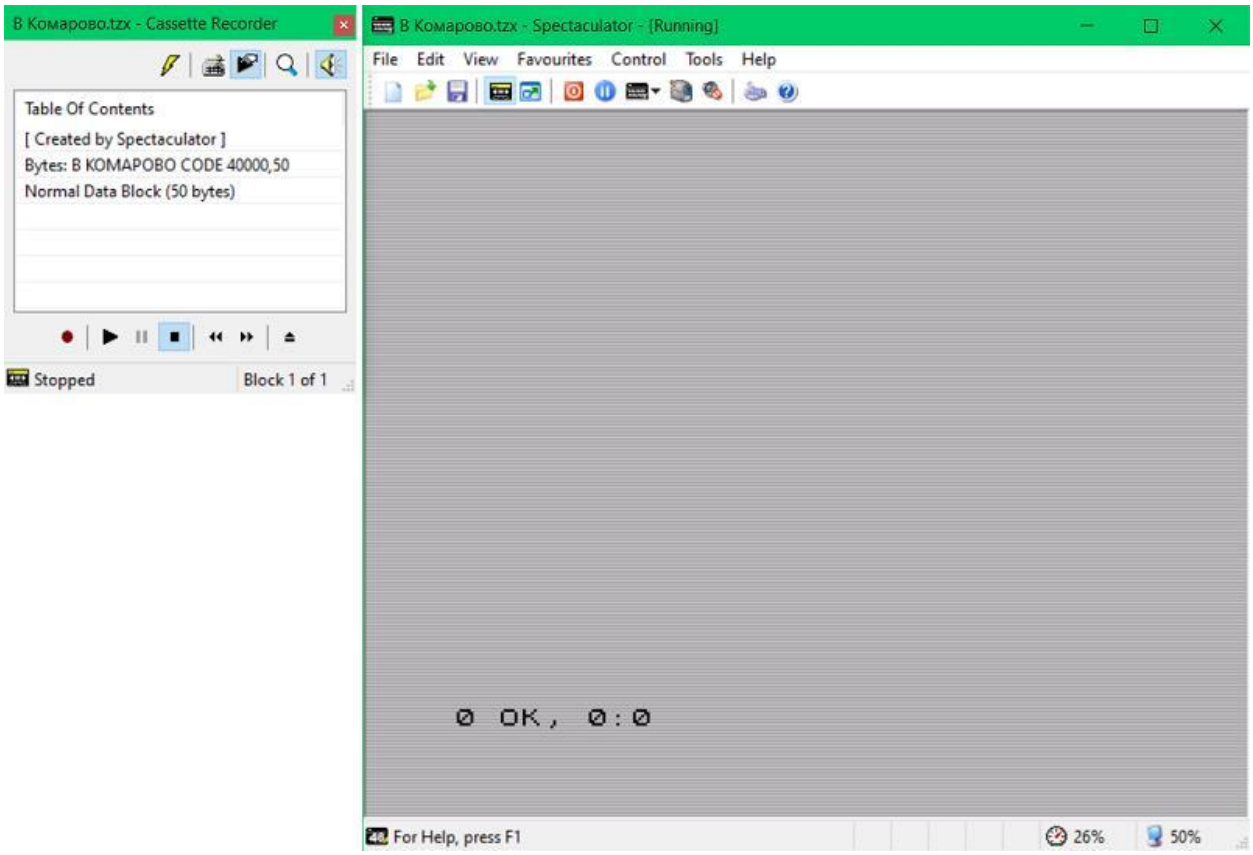


Рис. 323. Запись блока с данными.

Как видите, заголовок является частным случаем блока данных со специфическими параметрами. И заголовок, и блок данных записываются одной и то же подпрограммой `SA-BYTES` по адресу `1218`.

Казалось бы, тема исчерпана, но нет. Всё только начинается. Выньте виртуальную кассету из магнитофона, нажмите «*Reset*». После того, как произойдет перезагрузка, можно приступать к следующей главе.

Глава 36
Телепортационная щель в IBM-мир

Краткое содержание: временная деформация ПЗУ, разлом в адресе 1232, IBM-мир

В прошлой главе на практике убедились, что заголовок является обычным блоком данных, длиной 17 байт и научились раздельно создавать блоки данных и заголовки. Наверняка возникнет вопрос, а какой самый длинный блок данных можно записать? Для этого, первым делом, предлагаю обратиться к подпрограмме `SA-8-BITS (1317)`:

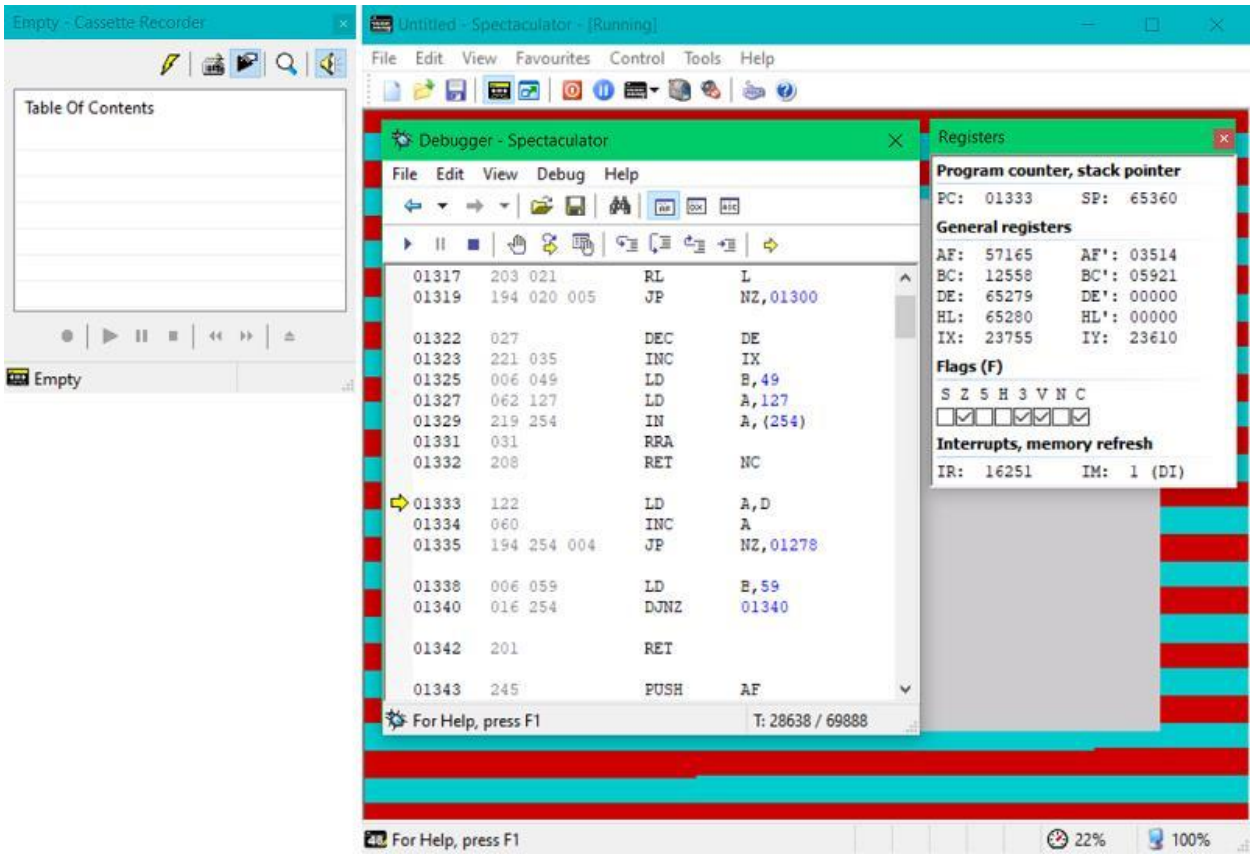


Рис. 324. Анализ работы программы SA-8-BITS в отладчике.

По адресу 1333 происходит проверка старшей части размера будущего блока. Если она 255 (255*256=25280) то запись прекращается. После небольшой паузы (1340) Стрелочка выходит в программу `SA/LD-RET (1343)` и далее на выход. Таким образом, максимальная длина блока данных, который возможно записать, должна быть 65279. При превышении этого значения запишется блок лишь в пару служебных байт длиной. Это теоретические прикидки.

А теперь представьте на практике, что вы поставили ● точку по адресу 1333, набрали команду `SAVE "MAXX"CODE 23755,65500`, открыли файл для записи программы и включили ● кнопочку «*REC*» виртуального магнитофона:

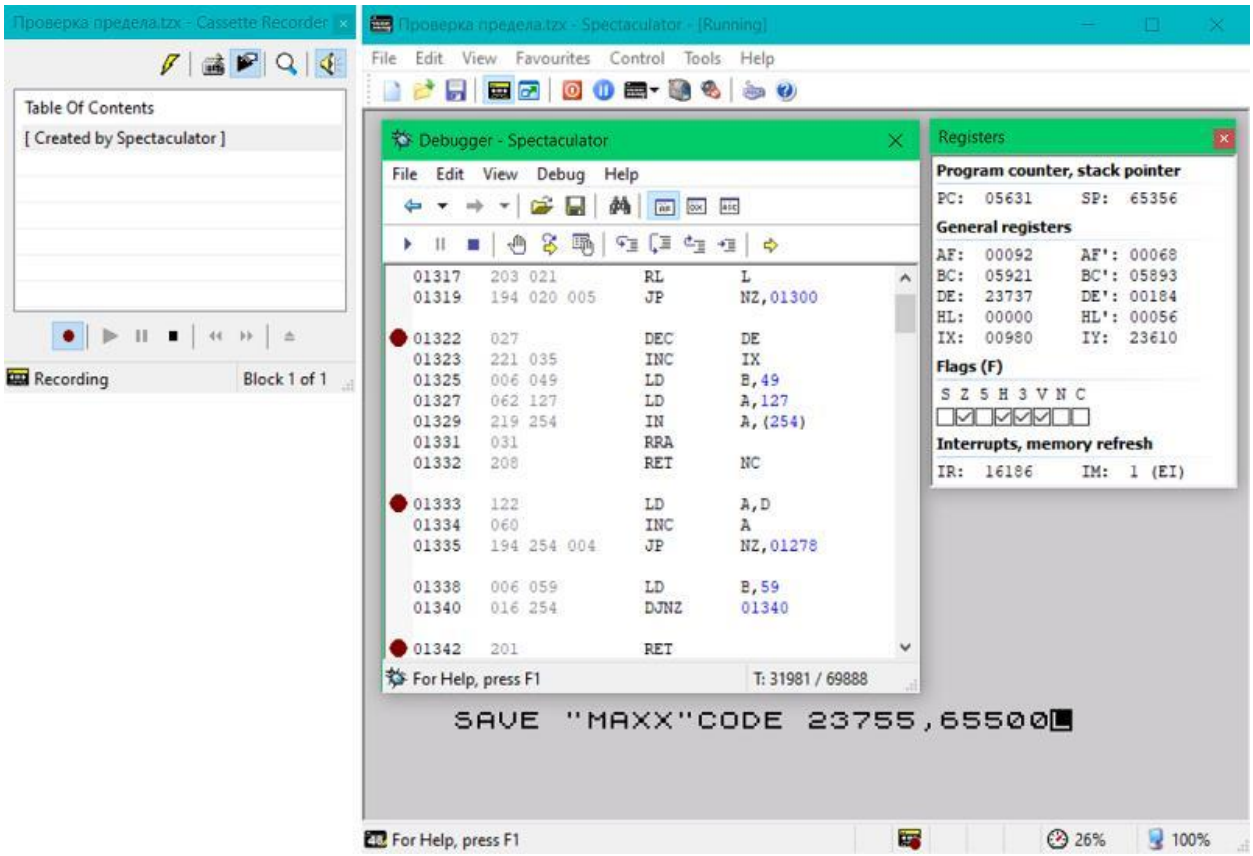


Рис. 325.

Для верности в примере я поставил даже две дополнительных ● точки на 1322 и 1342. По идее, в процессе выполнения команды, должен открыться «*Debugger*». В конечном итоге должна сработать защита, и запись прекратится на первом же действующем байте, создав пустой блок из двух технических значений.

Закрыв отладчик, вводите команду:

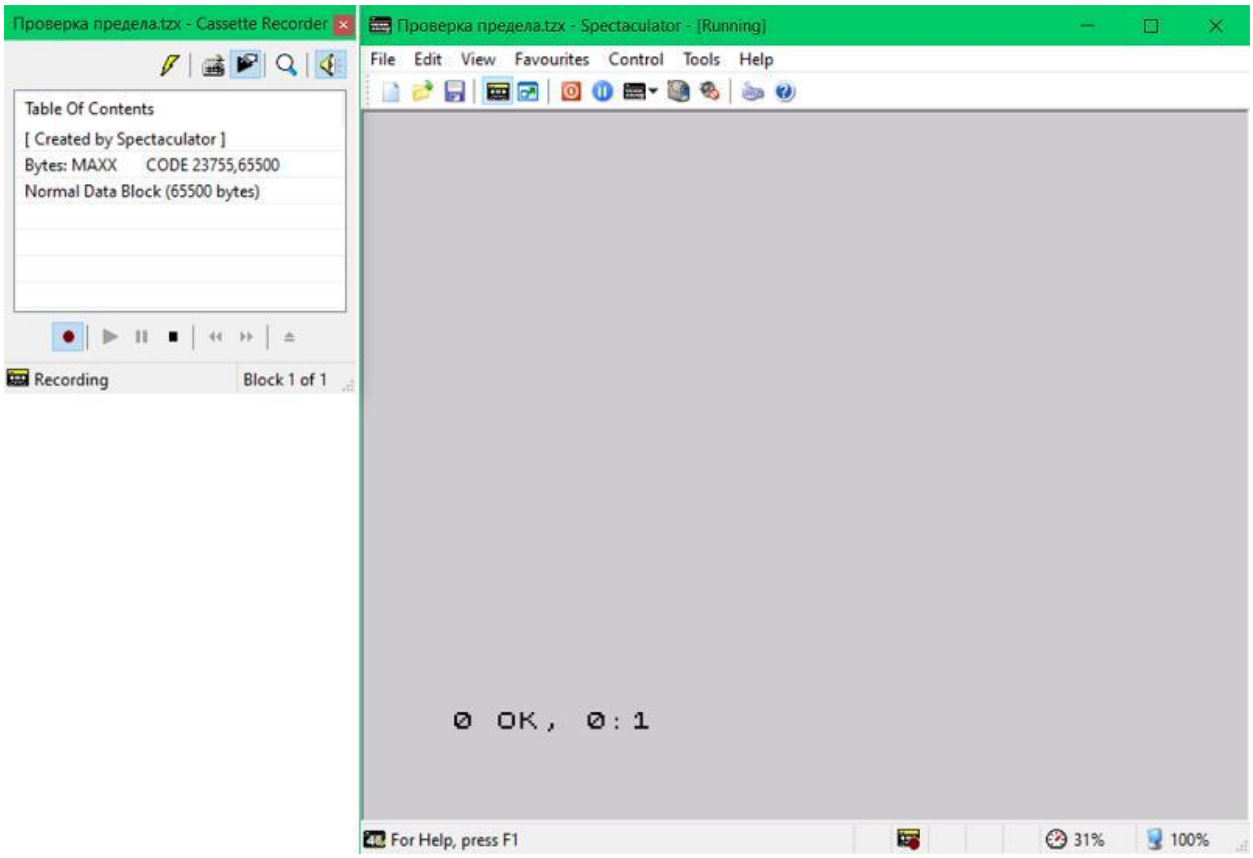


Рис. 326.

Как и в прошлые разы, вместо пробегания полосочек, экран безмолвствует. После двухсекундной паузы в виртуальном магнитофоне появляется записанный комплекс.

Вопреки всему здравому смыслу блоки не только записались, но и отладчик не открылся ни в одной из поставленных точек. А теперь если вы нажмёте на магнитофоне кнопку «*Stop*» и снова введёте команду `SAVE "MAXX"CODE 23755,65500`, отладчик откроется на установленных точках и пройдёт по запланированному маршруту, отказавшись записать данные.

Понятно, что ничего не понятно и происходит какая-то аномалия. Нужно пробовать разбираться. Известно, что любой блок записывается подпрограммой `SA-BYTES (1218)`. Включая кнопку «*REC*» и задавая значения регистрам, командой «*Trace*», вы отправляете Стрелочку по адресу 1218. Логичнее всего, поиск проблемы нужно начать с него.

В чистом *Spectaculator'e* откройте отладчик по адресу `SA-BYTES (1218)`:

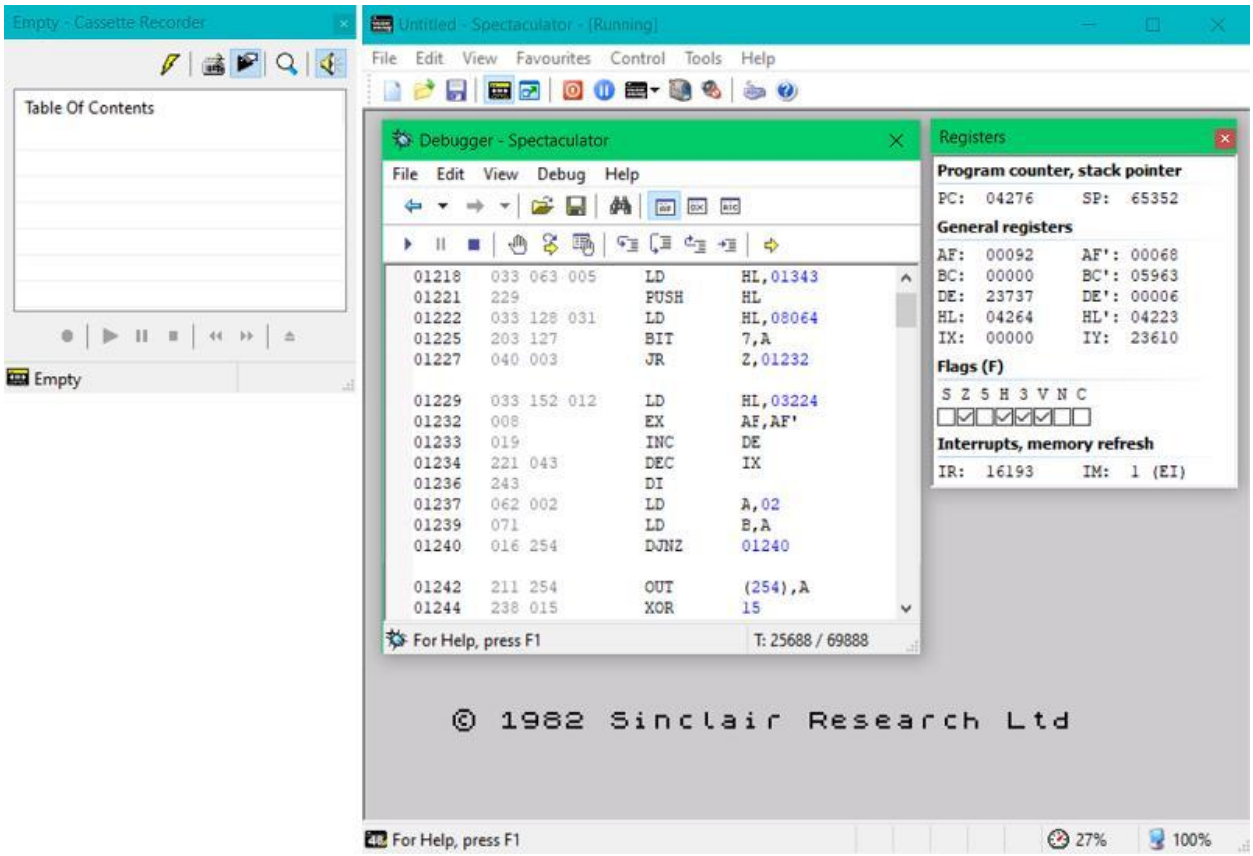


Рис. 327. SA-BYTES: состояние памяти до включения кнопки «REC» на магнитофоне.

На вид ничего подозрительного. Первым делом SP-башенка наращивается значением 1343 – адресом возврата в дополнительную подпрограмму `SA/LD-RET (1343)`, которая после записи и загрузки приводит в порядок экран. Следом в 1222 по значению типа, из «A» происходит задание длины пилот-тона для будущего блока. По умолчанию в «HL» записывается длинный сигнал в 8064 единицы и сразу проверяется «A», Если там значение ниже 128, то это заголовок. Можно перескакивать через ячейку на программу записи.

Если обнаружен бит 7, то длина корректируется на значение 3224 единицы и Стрелочка всё равно проваливается в программу записи. С команды «`EX AF, AF'`» по адресу 1232 начинается сама программа записи сигналов.

Предлагаю посмотреть, как будет происходить процесс формирования *.tzx* файла. Выйдете из отладчика, откройте файл для записи с произвольным именем и нажмите «*REC*» магнитофона. Снова войдите в «*Debugger*» на то же самый адрес:

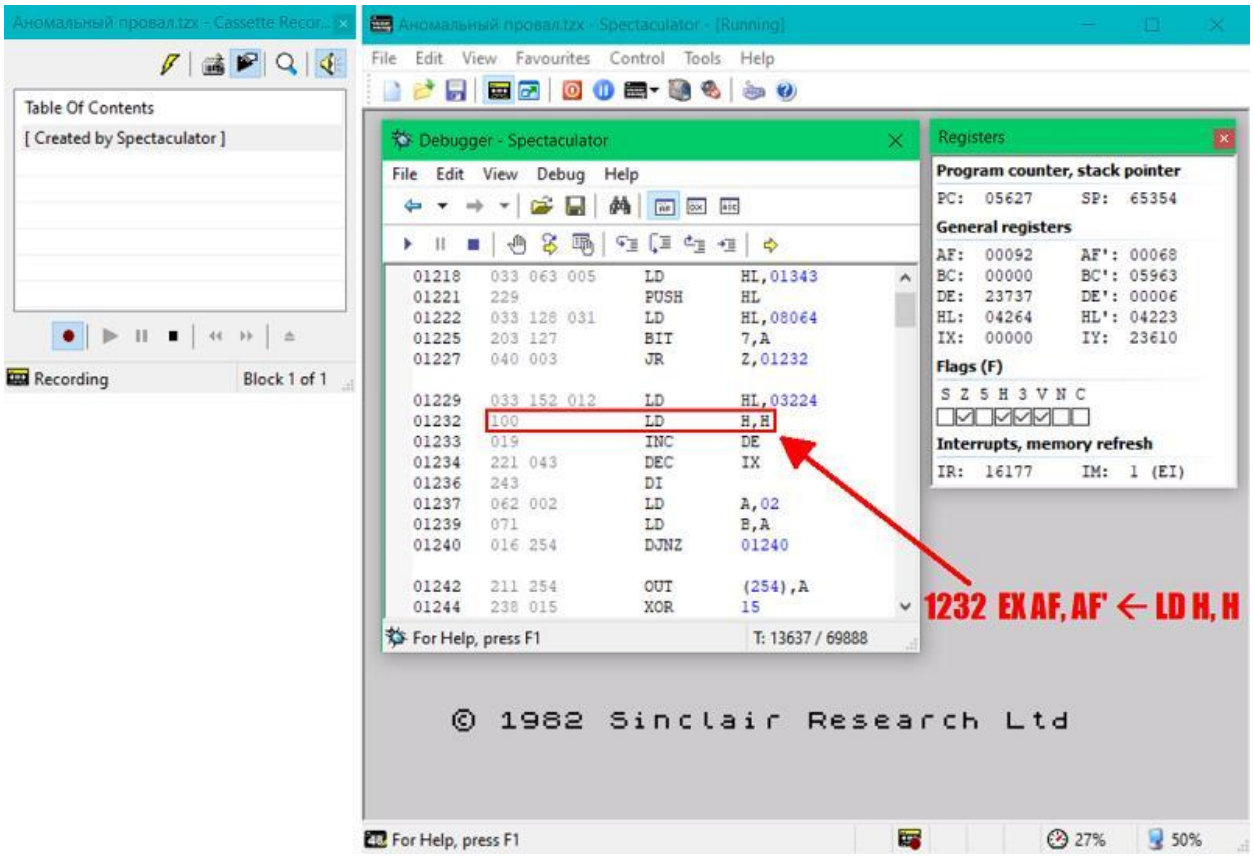


Рис. 328. Появление аномальной команды по адресу 1282 после нажатия кнопки «REC» на магнитоыоне.

Что-то тут не так!? При включении кнопочки «*REC*» команда «`EX AF,AF'`» по адресу 1232 внезапно поменялась на «`LD H,H`». А говорят ПЗУ что-то монументальное! Врут, все врут!

Безо всяких приготовлений, поставьте Стрелочку в ячейку 1218 и нажимайте клавишу «*F11*» или кнопочку «*Step Into*». Через 5-6 шагов Стрелочка встанет на аномально появившуюся команду «`LD H,H`» по адресу 1232. Однозначно, тут что-то должно произойти:

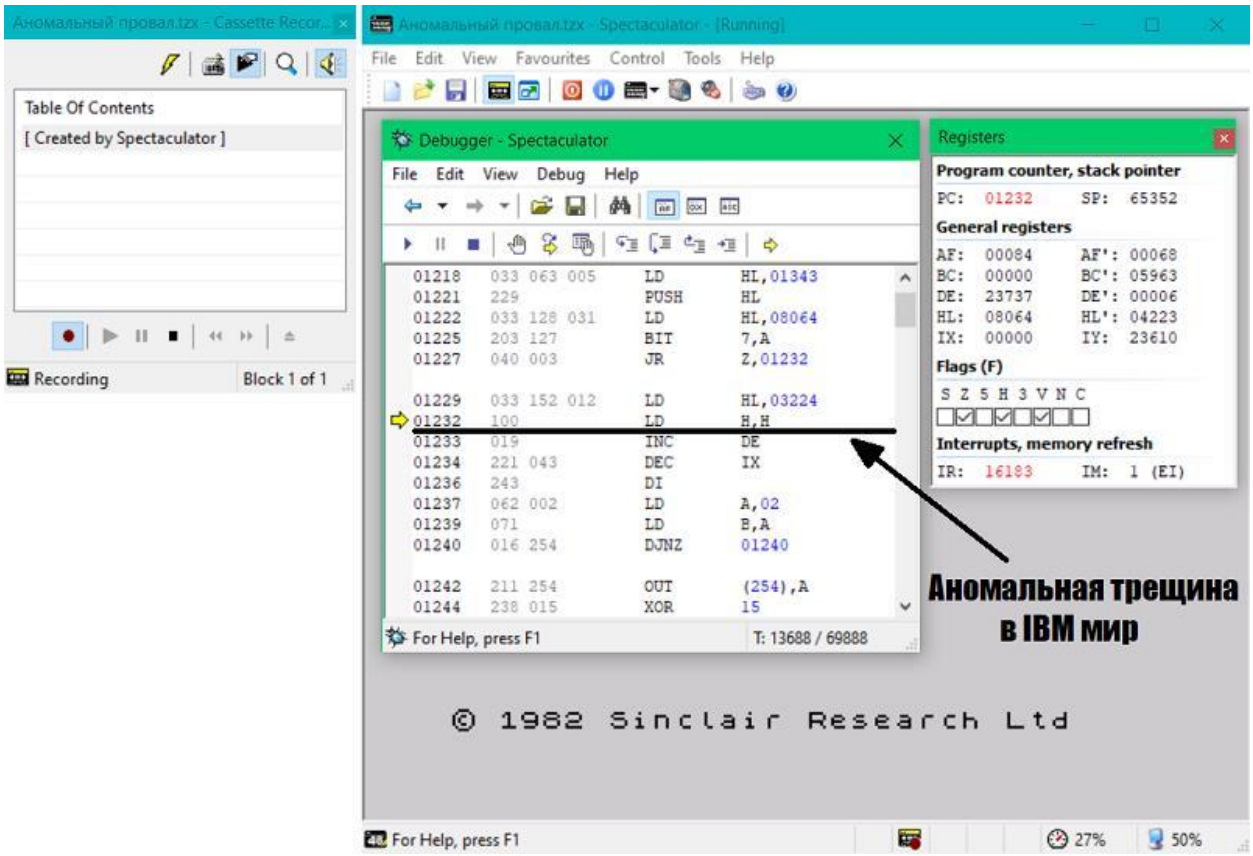


Рис. 329. В шаге от аномального разлома в «IBM-мир».

Снова нажав «*F11*», Стрелочка хочет наступить на следующую ячейку и…

…вот так независимо от значения «T» попадает в аномальный разлом и исчезает в недрах «IBM-мира». Дальше без внешнего отладчика, типа «Ольги» (OllyDbg) отследить ничего не получится. В моём примере связь со Стрелочкой пропала аж до середины программы «сброса»:

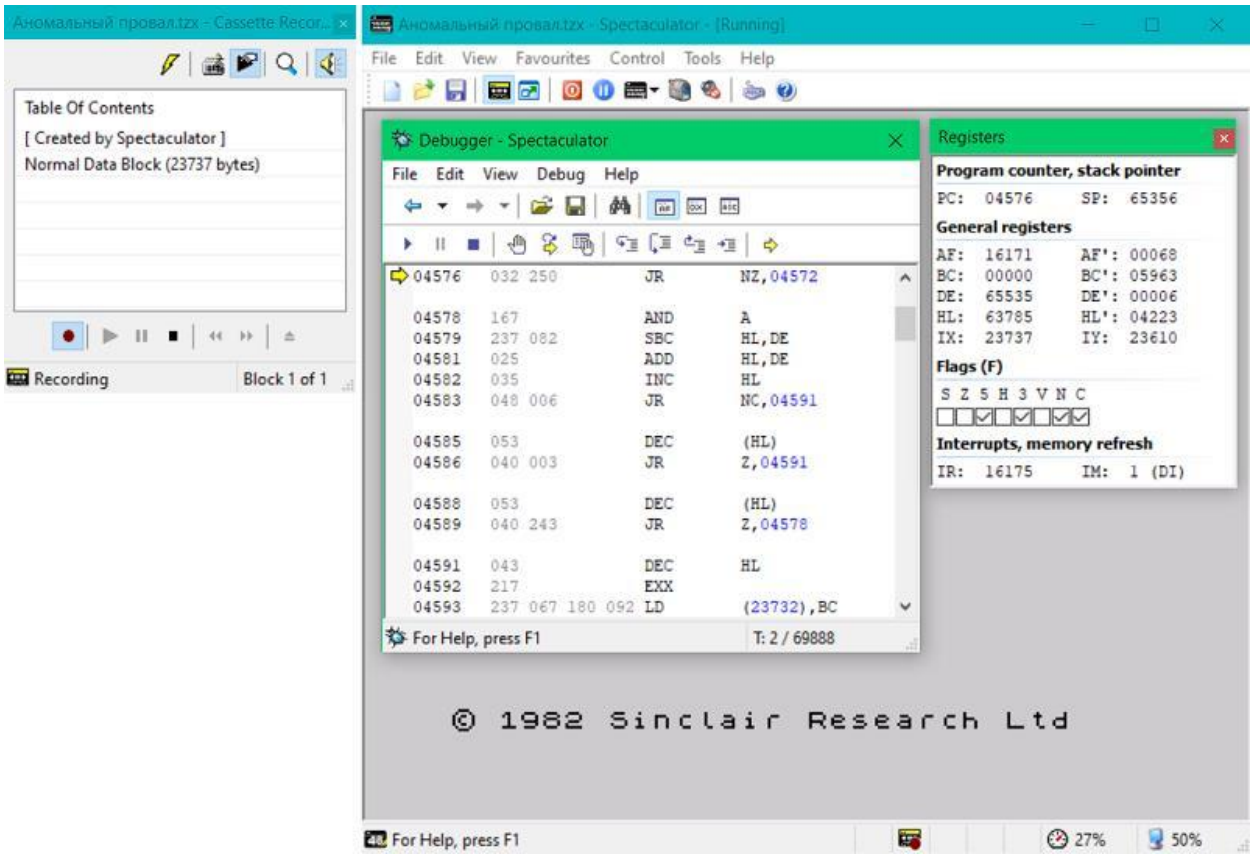


Рис. 330.

А в это время в виртуальном кассетном магнитофоне появляется блок данных произвольной длины, сгенерированный на текущих значениях окошечка «*Registers*». На самом деле, можно было поставить Стрелочку сразу в ячейку с открывшимся телепортационным окном и нажать «*Trace (F5)*» или «*Step Into (F11)*». Точно также Стрелочка провалилась бы в эту аномальную трещину между адресами, попытавшись с 1232 шагнуть в 1233.

А теперь выйдете из отладчика остановите магнитофон кнопкой «*Stop*». Снова зайдите в «*Debugger*» на отметку 1218 и взгляните на аномальное место:

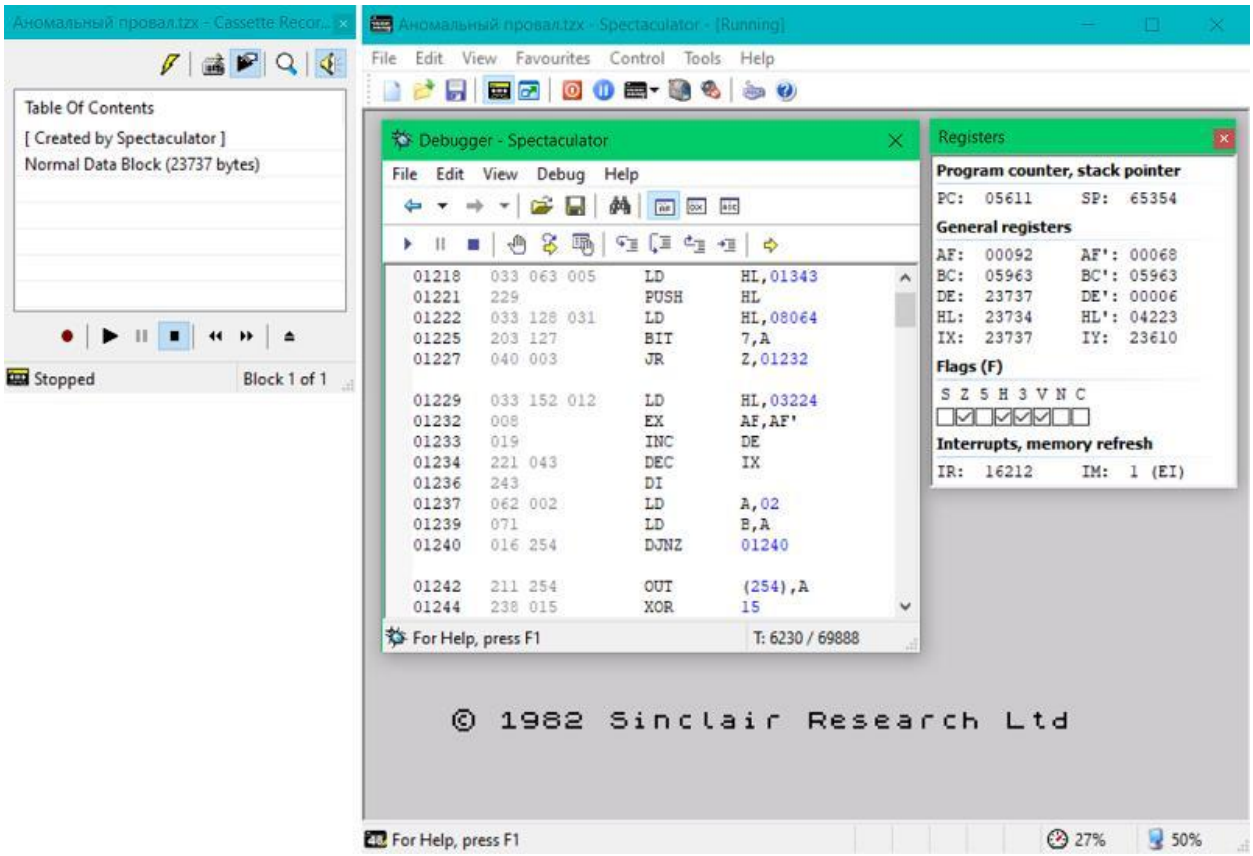


Рис. 331.

Окно телепортации схлопнулось, и команда «`EX AF,AF'`» возвратилась на место. Теперь если поставить Стрелочку на странную ячейку, она перешагнёт это место, как будто ничего не происходило и продолжит путь вниз, выдав на экран блок холостой записи с хаотичными параметрами и случайной длиной пилот-тона (в зависимости от того, что осталось в «HL»).

Таким образом, при нажатой кнопке «*REC*» виртуального магнитофона, любой проход Стрелочки через аномальную трещину на отметке 1232/1233, вызовет принудительное создание блока данных внешним алгоритмом приложения по текущим значениям «DE», «HL» и «IX» (и возможно «A»). При этом видимость процессов в «*Debugger*», приходит не сразу. Функциональность отладчика возвращается с небольшой задержкой. Это как если во время рекламного блока выключить телек, но немного не рассчитав, вместе с рекламой пропустить минуту фильма.

А теперь о максимальной длине блока данных. Поскольку записью *.tzx* файла занимается *Spectaculator.exe* в своём невидимом «IBM-мире», то остаётся почувствовать всю прелесть доэмуляторной эпохи и нащупывать предел вслепую, опытным путём. Нет, конечно, можно поковырять «Spectaculator» приложениями «HexEdit» с «OllyDbg», но это уже за рамками данной книги и «Вселенной ZX-Spectrum».

Как ни странно, но в этой проблеме, первый метод наиболее эффективен. Всего за пару минут выясняется истинный предел размера записываемого блока внешним методом. Этим значением будет число 65533.

Введите следующую программу:

```
New File [Длинный блок с дефектом.tzx]
Cassette Recorder [Record]
Debugger
Dec
AF ← 65280
DE ← 65533
```

```
HL ← 3224
IX ← 23755
SP ← 17000
PC ← 1232
Trace
Cassette Recorder [Stop]
Cassette Recorder [Eject]
```

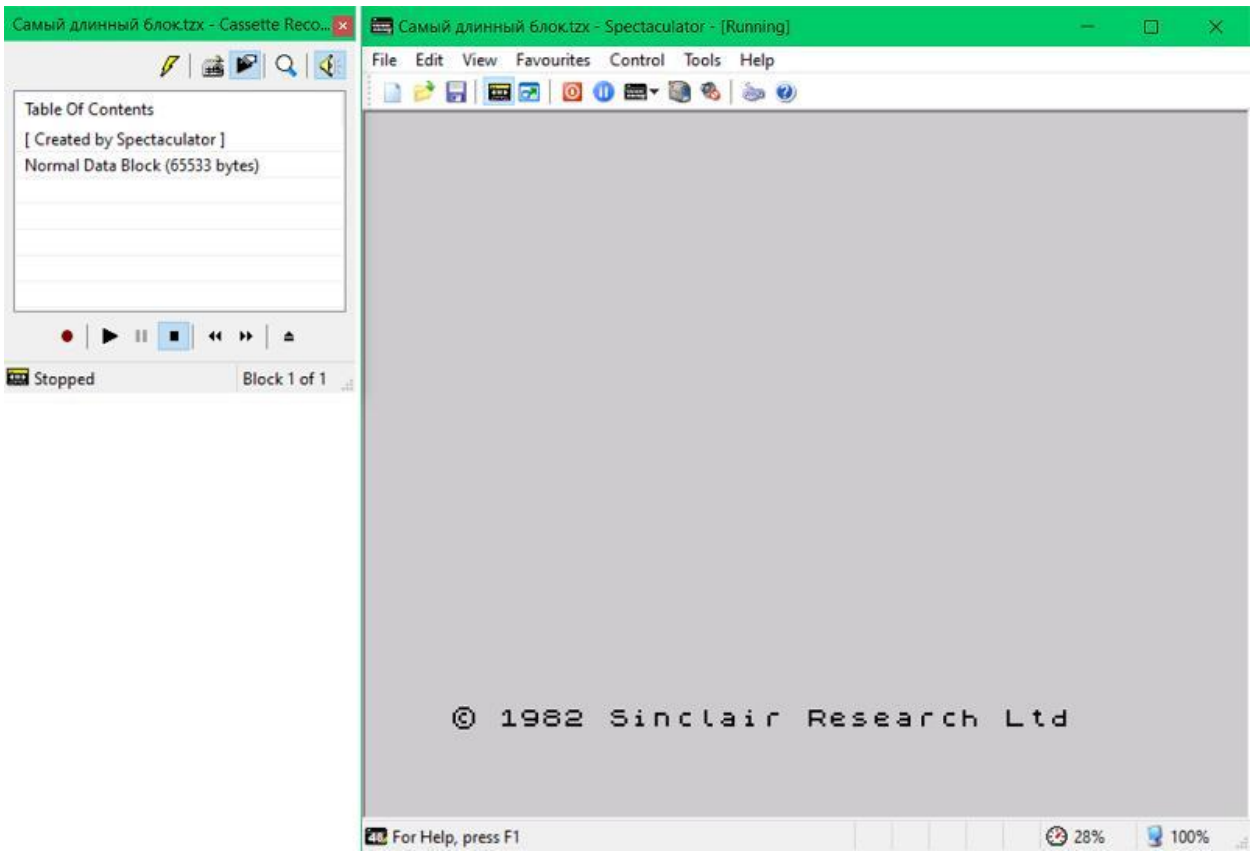


Рис. 332.

Итак, это самый длинный работоспособный блок, который возможно записать программой *Spectaculator*. Это логично, поскольку в спецификации *.tzx* файла на начинку с данными отведено 2 байта. Если к 65533 приложить первый и последний невидимый байт, то получится предел. Попытки сгенерировать блоки длиной 65534 и 65535 будут на выходе давать повреждённые файлы с длинами в 0 и 1 байт соответственно.

Конечно, вручную можно создать файл 70000 и даже 100000 байт. Для этого в «HexEdit» достаточно набить «рыбу» *.tzx* файла и вставить туда данные, но размерность самого формата останется прежней и больше 65535 байт за раз всё равно не считается.

Глава 37
Путешествие с командой LOAD

Краткое содержание: таинственные цветные сердечки

С позапрошлой главы у вас должен остаться «*Заголовок и данные.tzx*». На его примере предлагаю познакомиться с командой загрузки «LOAD». Представьте, что вы зарядили магнитофон этим файлом, набрали строку LOAD "" CODE. Следом нажали кнопочку ▶ «*Play (Ctrl+Space)*» магнитофона:

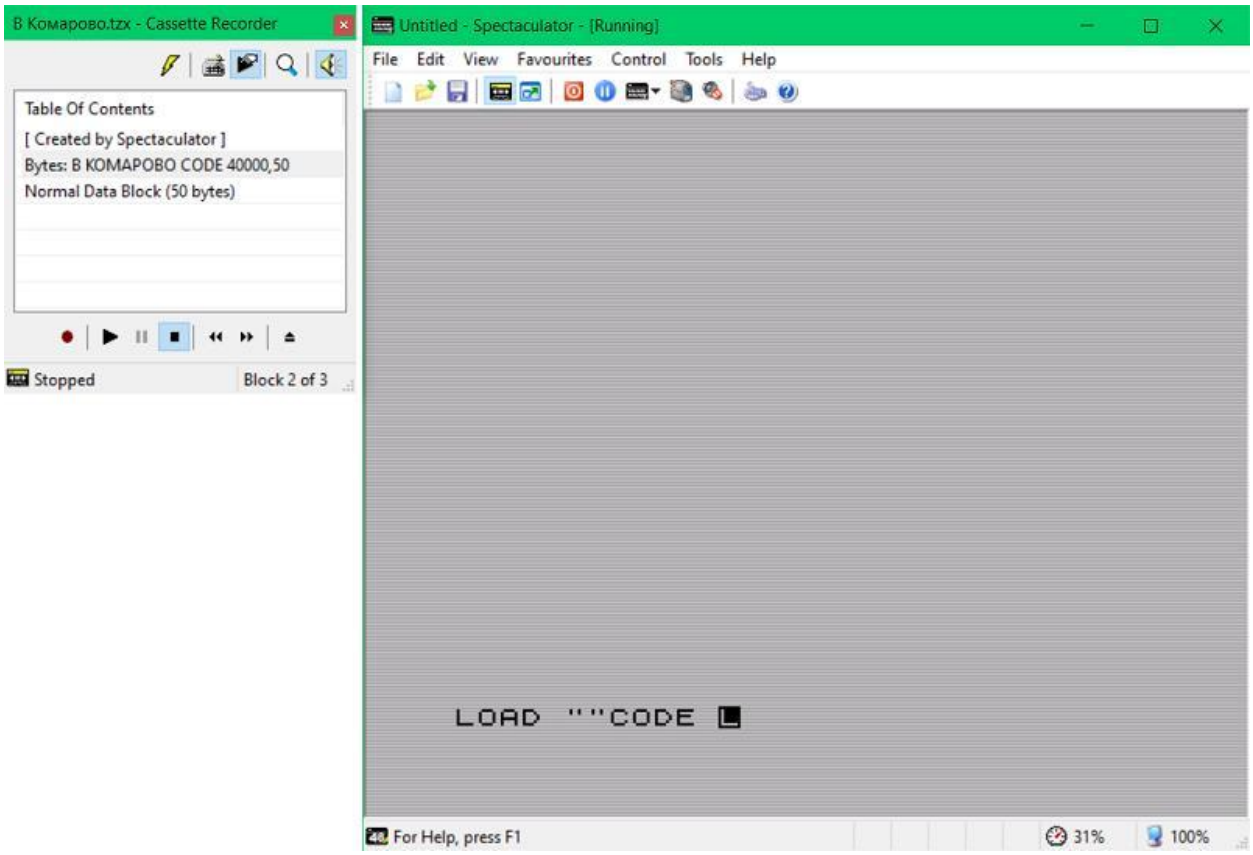


Рис. 333. Процесс набора команды LOAD ""CODE для загрузки блока.

Выдохнув, вы нажимаете ENTER. Открывается отладчик, который переносит вас в волшебную страну. Хотите попутешествовать со Стрелочкой от начала и до конца? Тогда присоединяйтесь и в добрый путь.

Стрелочка выходит из редактора на поверхность и следующим же шагом ныряет в программу синтаксического анализа (`4788 CALL 6935`):

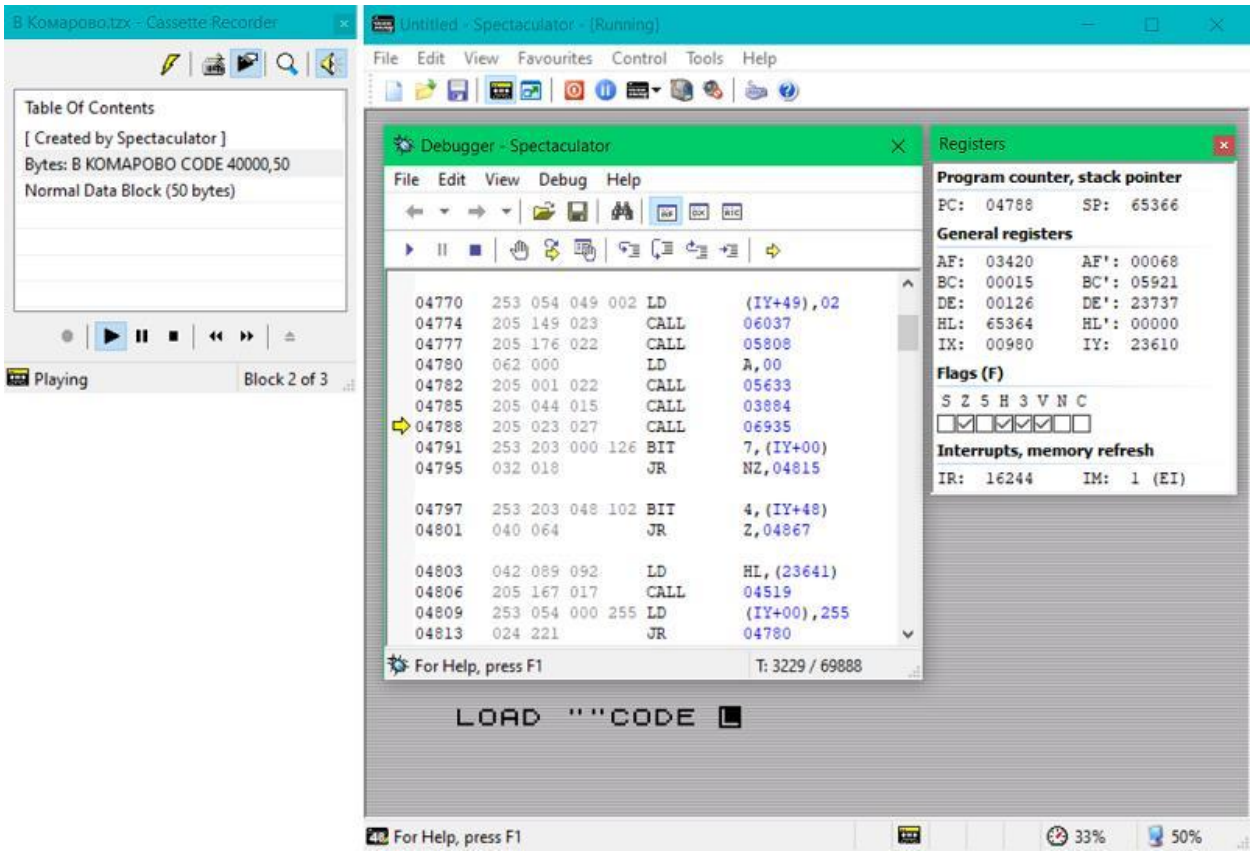


Рис. 334. Перед входом в подпрограмму синтаксического анализа.

После предварительной проверки на грубые синтаксические ошибки с прогоном идентификации, строка традиционно попадает в сортировочный центр (4815). Сейчас хочется акцентировать внимание, на моменты, которым раньше не придавалось значения. В данной ситуации они становятся ключевыми.

Адрес начала вводимой строки из `E_LINE (23641)` переписывается в комплект ячеек 23645/46, который называется `CH_ADD`. Этот указатель играет роль внутреннего курсора, который двигается по текущей BASIC-строке в процессе выполнения команд. За его переключение отвечает комплекс программ «`RST 20`», который состоит из нескольких фрагментов, разбросанных по верхним ячейкам памяти. Обращение к «курсору» и его передвижение вызывается из многих программ, которые проверяют или перерабатывают BASIC-строки, превращая их в видимые действия:

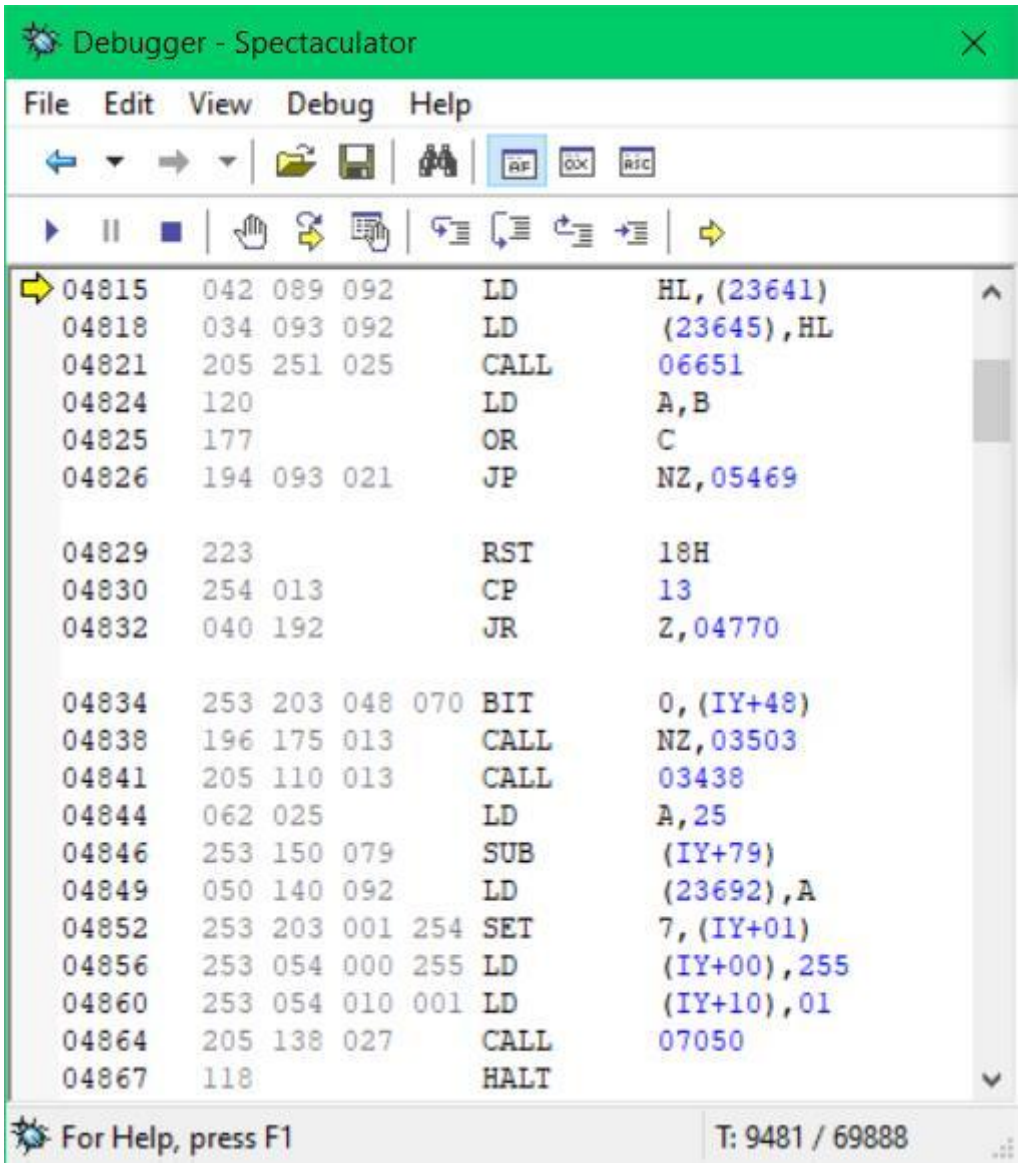


Рис. 335. Подпрограмма MAIN-3 (4815-4864).

Курсор установлен на первый символ в самое начало строки, то есть на команду «LOAD». На чистом компьютере это будет адрес 23756. Дальше производится попытка выяснить номер строки, ну и после отсева, с отметки 4834 начинается подготовка строки к запуску.

В рамках общей подготовки включается бит-тумблер №7 FLAGS (23611), показывая, что строка взята в работу и вот-вот начнёт выполняться. В NSPPC (23620) устанавливается номер выполняемого оператора, а в ERR_NR (23610) авансом выставляется полуфабрикат кода сообщения «0 OK.» об успешном выполнении.

С этими вводными данными Стрелочка заходит в программу LINE RUN (7050), где продолжается перекройка системных переменных. Ячейке номера строки, в которой расположен текущий оператор PPC (23621) выставляется код «65534», что означает выполнение в реальном времени:

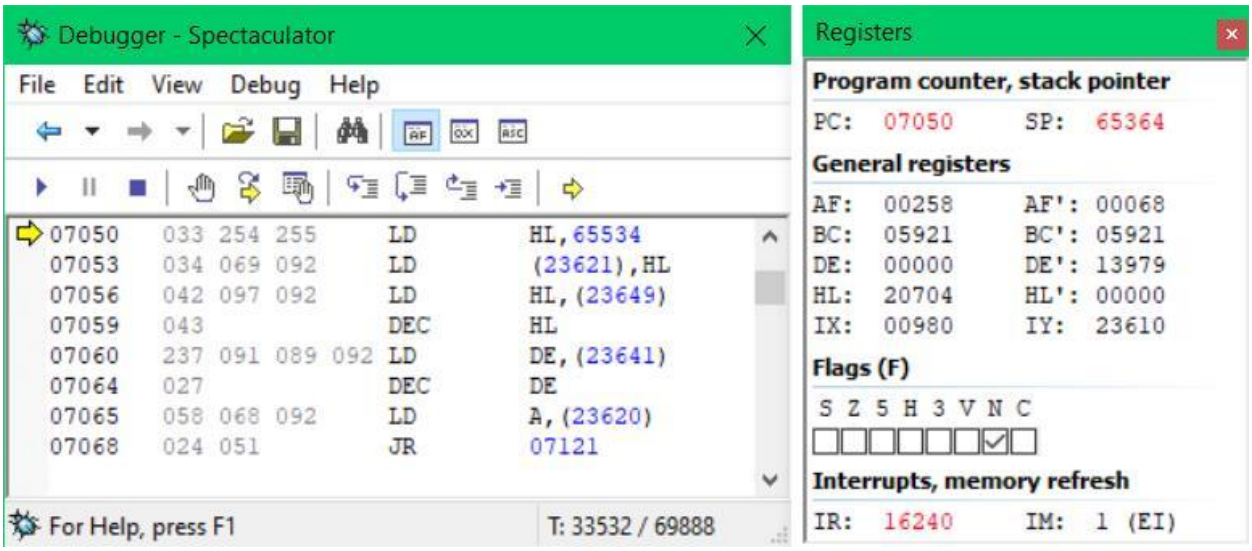


Рис. 336. Начало программы LINE RUN (7050-7068)

По границам соседних областей, вычисляется полная длина строки, независимо от того, сколько в ней блоков команд разделённых двоеточием. Словно клещами, она обхватывается с начала и конца. По результату в «HL» записывается адрес, указывающий на маркер сразу за концом строки, в «DE» на предыдущий байт перед первым символом строки. В текущей ситуации этими значениями будут 23755 и 23761.

Из NSPPC (23620) берётся номер следующего выполняемого оператора, и Стрелочка отправляется на подпрограмму NEXT LINE (7121):

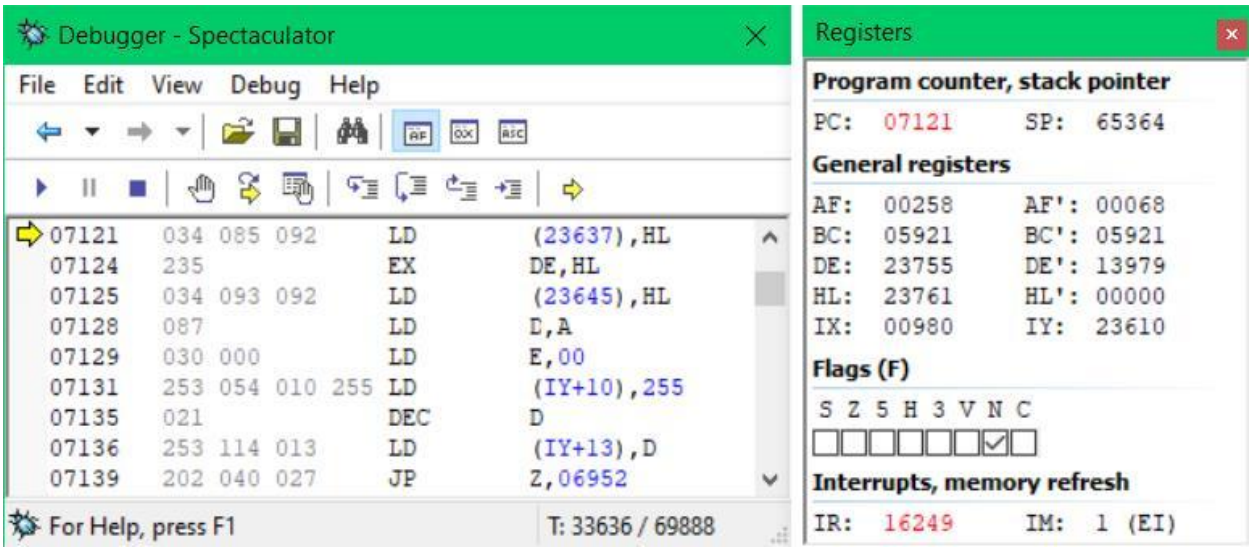


Рис. 337. Программа NEXT LINE (7121-7149) во время выполнения LOAD ""CODE.

Первым делом маркер конца строки, который пока остаётся 23761, забивается в одноимённую переменную NXTLIN (23637). Внутренний курсор CHADD (23645) временно отскакивает на предыдущую ячейку, и встаёт перед командой «LOAD», чтобы торжественно шагнуть на неё, приступив к чистовому анализу разбора строки.

Теперь уже и ячейка следующего выполняемого оператора NSPPC (23620) забивается заглушкой «255», показывая, что процесс выполнения строки пошел. В тоже время, из взятого в NSPPC (23620) номера следующего оператора вычитается единица, и полученное значение переписывается в SUBPPC (23623). Поняв, что строка, выполняемая и одновременно простая, происходит переход на STMT-LOOP (6952) или подпрограмму операторного цикла:

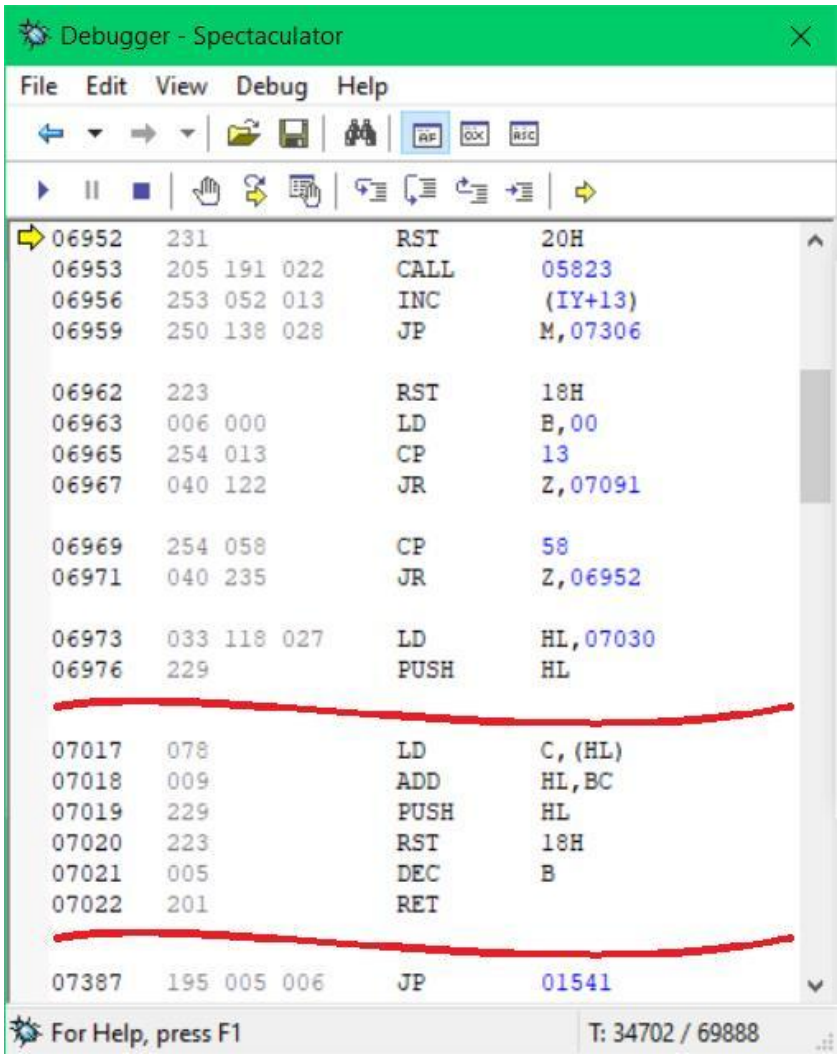


Рис. 338. Подпрограмма STMT-LOOP (6952-6992).

В STMT-LOOP (6952) первой командой стоит «RST 20», поэтому курсор CH_ADD (23645) снова спускается на команду «LOAD». Начинается цикл прокрутки строки для анализа операторов.

Перещелкнув курсор на ступеньку вниз, вызывается подпрограмма SET WORK (5823), которая на основе данных WORKSP (23649) отодвигает и остальные дружественные области STKBOT (23651) и STKEND (23653).

Также прицепом выставляется указатель области MEM (23656). В исходном положении он указывает на огромный пустырь по прозвищу MEMBOT, который вклинился в центр системных переменных и простирается с адреса 23698.

Следом, номер выполняемого оператора SUBPPC (23623) увеличивается на единицу. Ниже по адресу 6959, сразу проверяется значение на 128 блоков операторов в одной строке.

Если текущий оператор в строке 128-й по счету, то это явно перебор. Нужно останавливать программу. Взяв из библиотеки сообщение об ошибке №11, и срезав почти под основание SP-башенку значением ERR_SP (23613), в аварийном порядке завершается работа. Происходит выброс на поверхность по адресу 4867, где и выводится текст выбранного сообщения на экран.

Ну а если всё нормально, то с помощью «RST 18» смотрится символ, на который указывает курсор CH_ADD (23645).

Если это не ENTER, то следом проверяется двоеточие (:) (*в случае обнаружения двоеточия происходит возврат в начало STMT-LOOP и курсор CHADD перещелкивается на следующую ячейку*).

Но в текущей ситуации по указанному адресу всё еще находится LOAD, поэтому можно переходить к стадии анализа и опознания текущей команды.

В точке 6973 задаётся адрес выхода после успешного завершения работы команды и записывается в «SP». Этим числом будет 7030. С этого момента в «SP» пошёл набор каскада значений для выхода на финальной стадии.

Перещёлкнув курсор CH_ADD (23645) на следующую за командой LOAD, первую кавычку (23657), Стрелочка дедуктивным методом начинает распознавать характер команды в текущей строке, чтобы подобрать ей нужный набор свойств. Проходя вниз всю эту подпрограмму, к отметке 7022 сформировывается адрес перехода на пакет подпрограмм для выполнения текущей команды.

По команде RET, через JP-трамплинчик в 7837, Стрелочка переносится в огромный универсальный комплекс подпрограмм записи и считывания (LOAD, SAVE, VERIFY и MERGE) по адресу 1541:

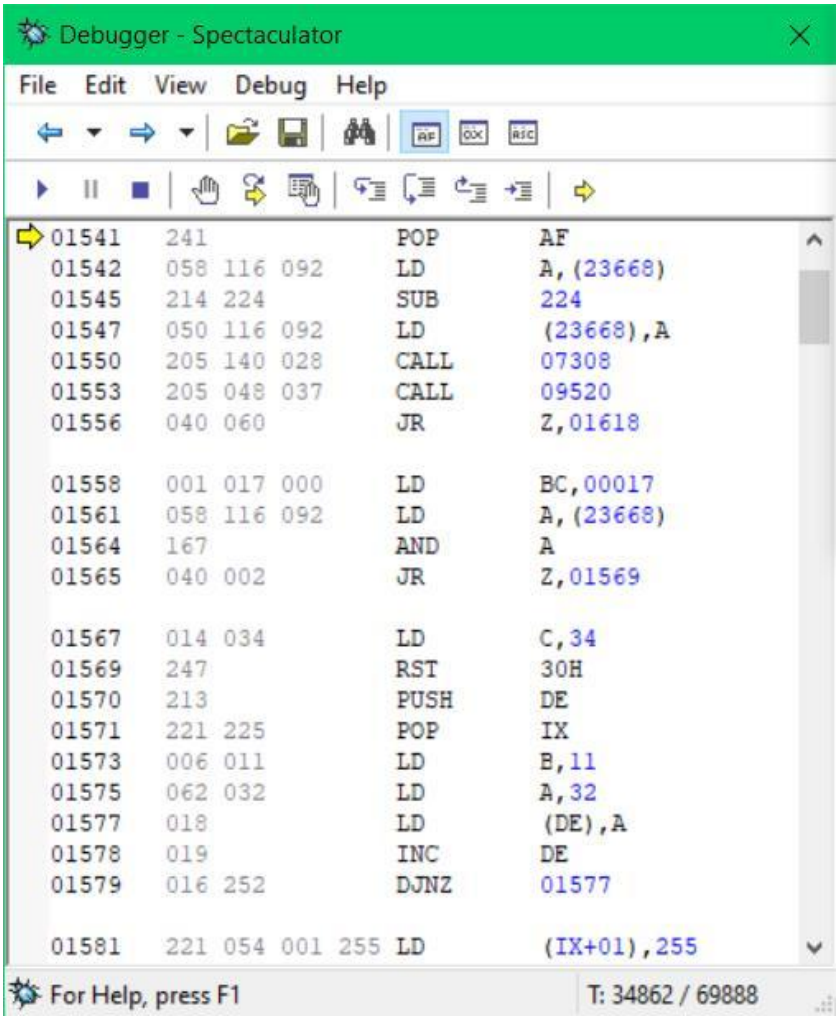


Рис. 339. Начало комплекса подпрограмм «LOAD, SAVE, VERIFY, MERGE» (1541).

И вот в этой программе уже начинается детальный логический разбор, что же именно за конструкция команды прилетела на обработку. Ведь дать команду загрузки или записи можно множеством способов, а не только LOAD "". Например:

```
LOAD ""DATA a$()
LOAD ""SCREEN$
LOAD "Название"
SAVE "Название" CODE 16384,32768,
MERGE ""
```

и прочие.

Все это нужно идентифицировать, чтобы правильно подобрать параметры для записи/считывания информации, ну в конкретном случае для считывания. В данной ситуации выборка комплекта начинается с формирования общей конструкции команды.

Сбросив лишнюю верхушку SP-башенки, начинается детальное распознавание команды. Из переменной T_ADDR (23668) вытаскивается значение и по нему вычисляется, что это за команда (0-SAVE 1-LOAD 2-VERIFY 3-MERGE). Поняв, что это LOAD, вызывается подпрограмма EXPT-EXP (7308). В ней смотрится содержимое внутри кавычек. По окончанию анализа, Стрелочка забежала на уже знакомую программу по адресу 116 и дважды продвинула курсор CHADD (23645), в результате чего он стал указывать на ячейку 23759, в которой стоит последняя команда CODE.

По включенному тумблеру №7 переменной FLAGS (23611), Стрелочка убеждается, что идет реальный процесс выполнения команды, а не холостая синтаксическая проверка после редактора, и продолжает выполнение программы по полному сценарию.

С адреса 1558, исходя из содержимого внутри кавычек, начинает формироваться область упрощенного теоретического или желаемого заголовка. Берётся длина для двух подряд заголовков (17x2=34 байта) и под это дело от рабочей области на заданную длину отодвигаются STKBOT (23651) и STKEND (23653). В «DE» скидывается начало, в «HL» конец отведённого места.

В 1571 «IX» принимает значение 23762 и туда копируется заготовка из 11-ти пробелов. Далее авансом в ячейку первой буквы теоретического заголовка ставится заглушка «255» в стиле «всё равно, с каким именем загружать».

Так как знаков в имени не должно быть более 10, происходит проверка количества символов, обнаруженных внутри кавычек. Если знаков в кавычках больше 10, то происходит отсечка имени, начиная с 11-го. В текущем случае, символов нет, поэтому стрелочка переходит в адрес 1618 к опознанию следующей за LOAD команды. Считав текущий символ из ячейки 23759, начинается последовательный перебор допустимых комбинаций. Поняв, что это не DATA, не SCREEN$, а всего лишь CODE, с адреса 1735 продолжаются приготовления к загрузке:

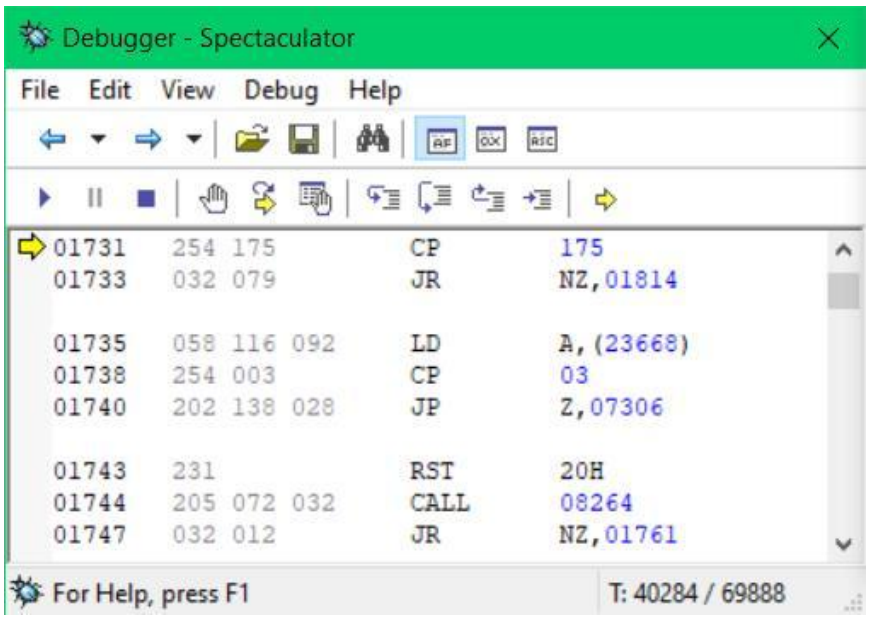


Рис. 340. Подпрограмма SA-CODE (1731-1759).

Команда опознана, и теперь можно переключать внутренний курсор. И вот по адресу 1743 Стрелочка заходит по «RST 20» и заключительный раз перед началом записи перещёлкивает CH_ADD (23645). Теперь он указывает на ячейку 23760, в которой содержится код ENTER:

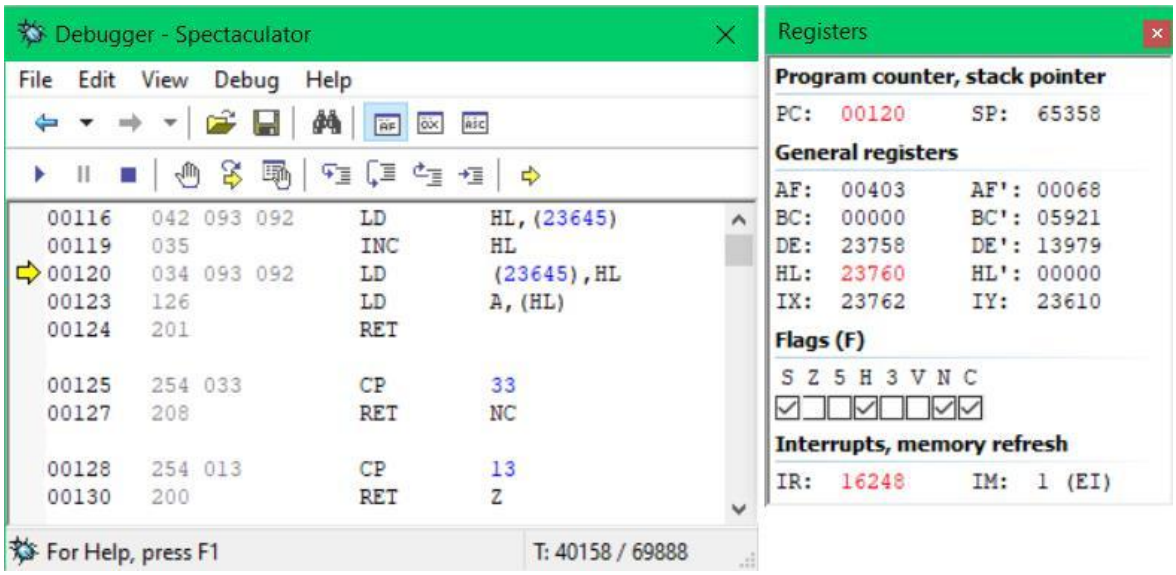


Рис. 341. Подпрограмма CH_ADD+1 (116-124).

Итак, ячейки CH_ADD (23645/46) накануне загрузки становятся равными 23760.

После проверки на управляющие символы, начинается поиск желаемого адреса считывания. Поскольку после CODE была пустота, то параметры длины и стартового адреса не имеют значения. Таким образом, с адреса 1791 в область теоретического заголовка заносятся нули. Заключительным штрихом, по адресу 1808 в начало заголовка, вместо головного пробела, вписывается число «3», означающее «*Bytes:*».

Теоретический заголовок сформирован и выглядит так:

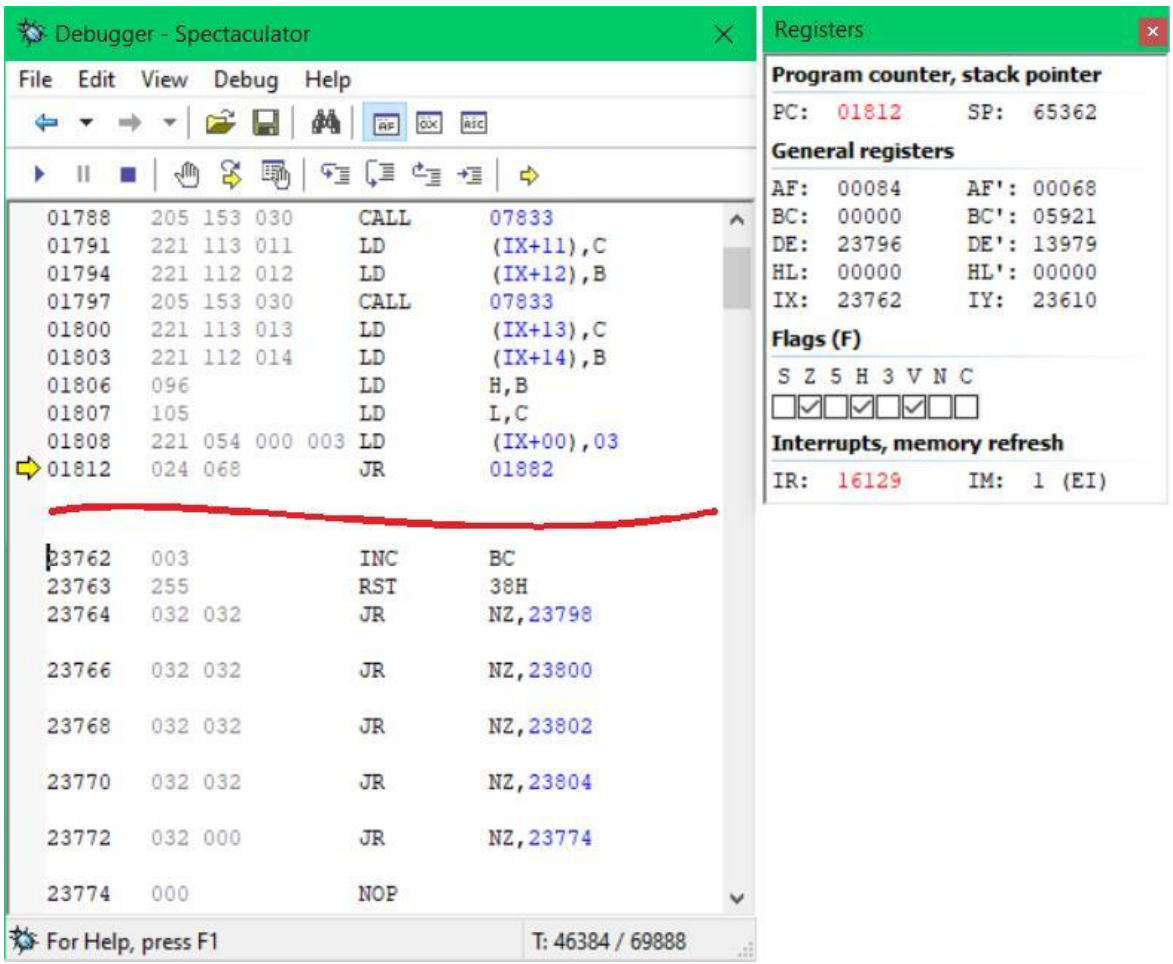


Рис. 342. Подпрограмма SA-CODE-4 и сформированный «теоретический» заголовок.

С адреса 1889 начинается подведение итогов.

Для считываемого реального заголовка задается область следом за «теоретическим», после чего в «IX» корректируется адрес. В «DE» выставляется длина 17 байт. В «A» задаётся тип блока «0», что означает заголовок и командой «SCF» принудительно устанавливается галочка «C», поясняя, что данные нужно реально загружать (LOAD), а не проверять (VERIFY). Наконец, в точке 1902 вызывается программа считывания блока заголовка:

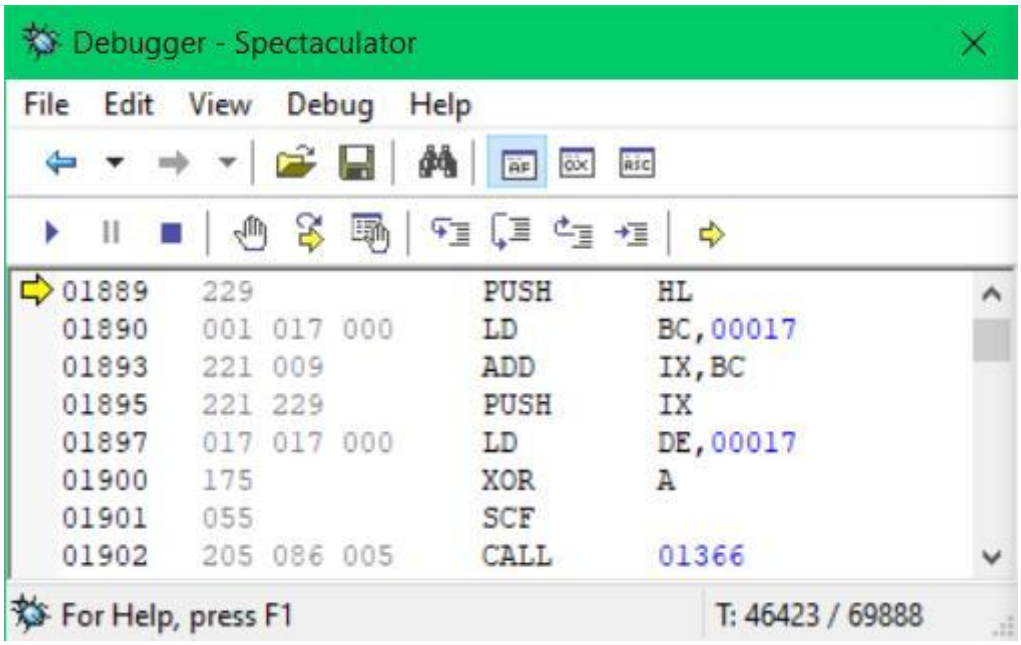


Рис. 343. Последние приготовления перед считыванием заголовка.

После длинного (~4,564 сек) и протяжного «Пи-и-и-и-и-и-и-и-и-и-и», последует короткое 17-ти байтное «Тщю-у-у», и по адресу [WORKSP]+17 появляется реальная копия заголовка. Таким образом, друг за другом встают ожидаемый теоретический и реальный блоки заголовков:

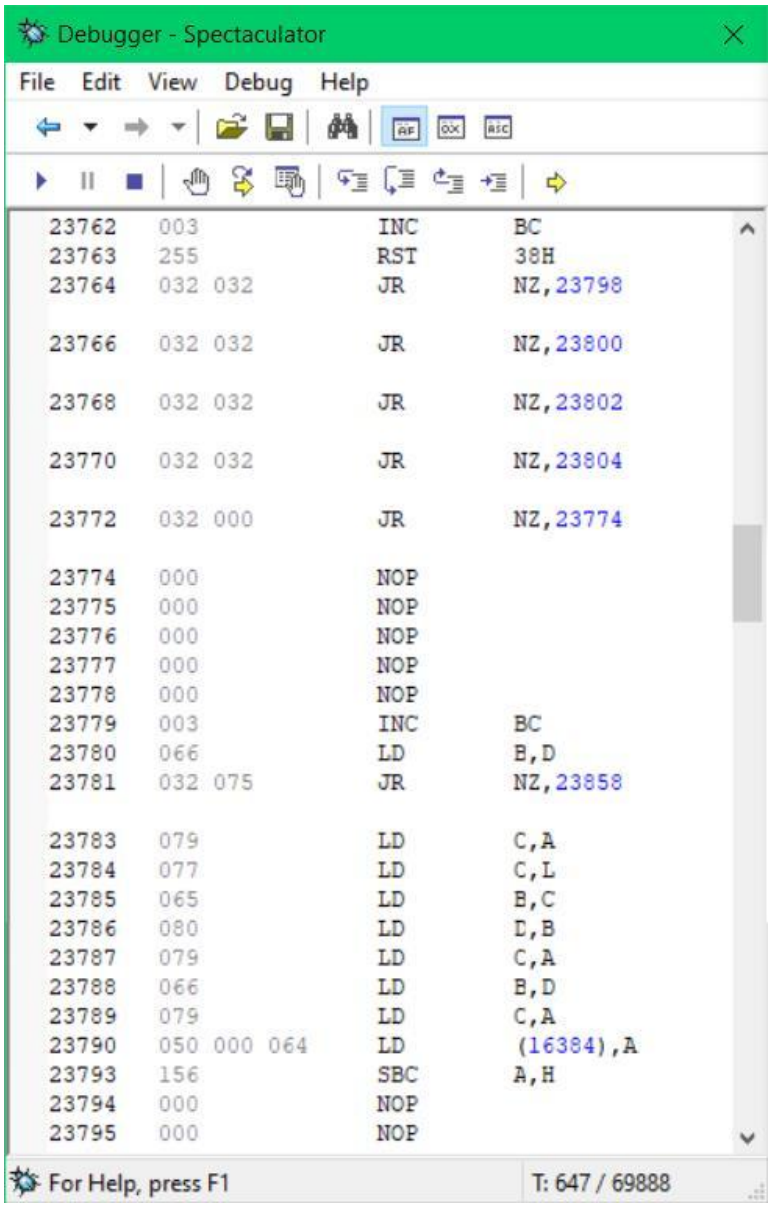


Рис. 344. «Теоретический» и «реальный» заголовки в рабочей области.

И Стрелочка возвращается назад. Если при считывании обнаружился блок данных вместо заголовка, или что-то совсем зашкварное, то нужно вернуться к считыванию и повторять до упора, пока не встретится заголовок. В данном случае, заголовок успешно считался, поэтому берётся адрес реально считанного заголовка (23779, IX+17) и подготавливается вывод в основной экран.

По типу байта данных в «A» (0-Program … 3-Bytes), отсчетом от начала, в таблице сообщений отбирается нужное слово и выводится на экран с помощью подпрограммы `PO-MSG (3082)`. Следом в 1965, командой «`RST 16`» выводится имя считанного заголовка с одновременным сравнением копий. Если теоретический или желаемый заголовок имел более, чем пустые кавычки, то производится сравнение и выставляется C ☑ галочка C, сигнализирующая о несовпадении желаемой и действительной информации.

После вывода заголовка немедленно принимается решение о дальнейших действиях. Если считанный заголовок не подошел по критериям, то происходит возврат в режим считывания заголовка. Стрелочка будет пытаться считать все попавшиеся заголовки, игнорируя данные, пока не найдет нужный или не кончится виртуальная/реальная кассета. Если в заголовке вместо первой буквы имени стояла заглушка «255», значит можно приступать к считыванию любого блока данных.

Печатается ENTER для перевода строки, чтобы следующий считанный заголовок вывести ниже:

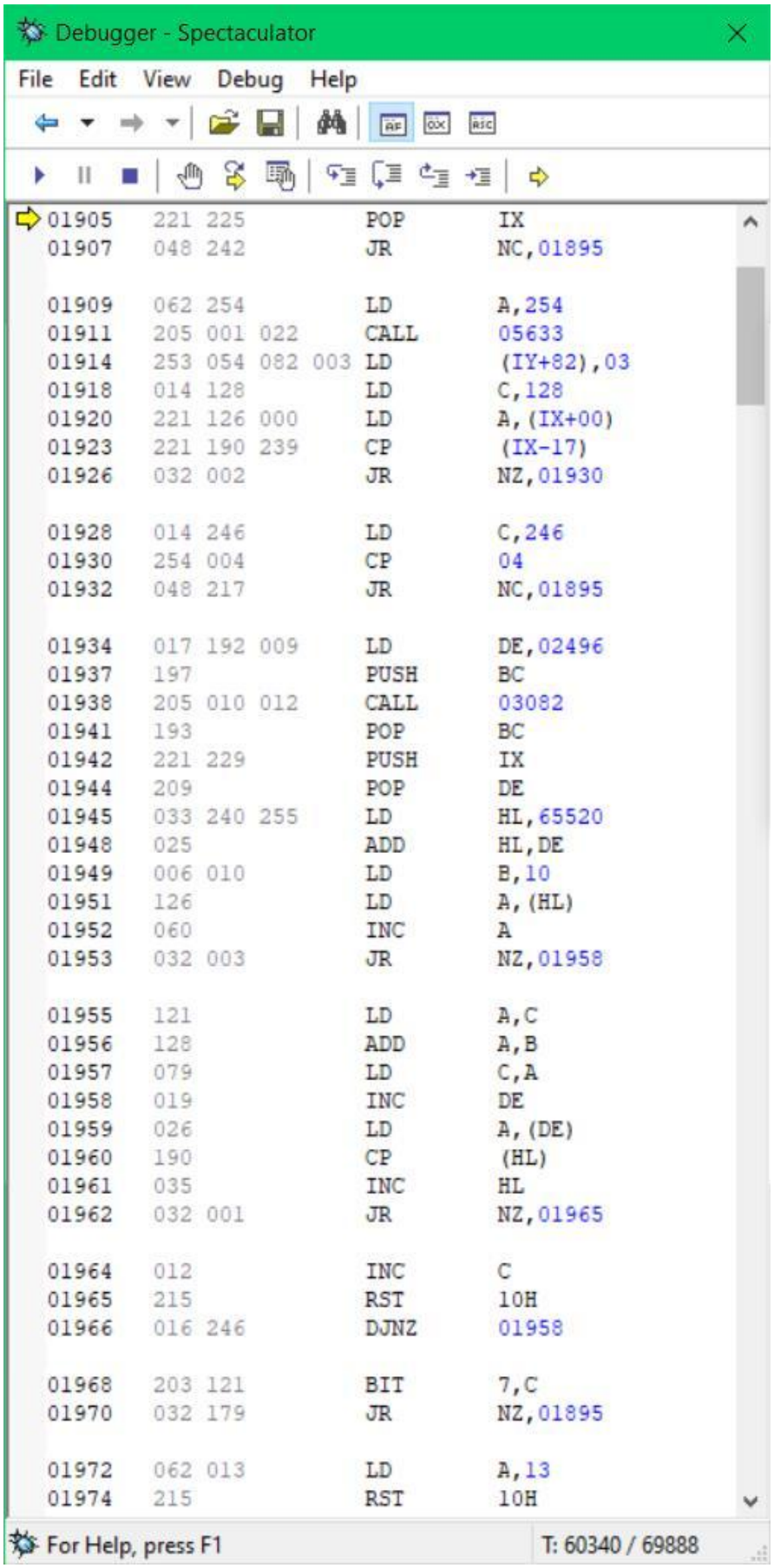


Рис. 345. Вывод считанного заголовка на экран.

С ячейки 1995, вперемешку с данными для других команд, из считанного заголовка начинается сбор параметров для загрузки блока данных:

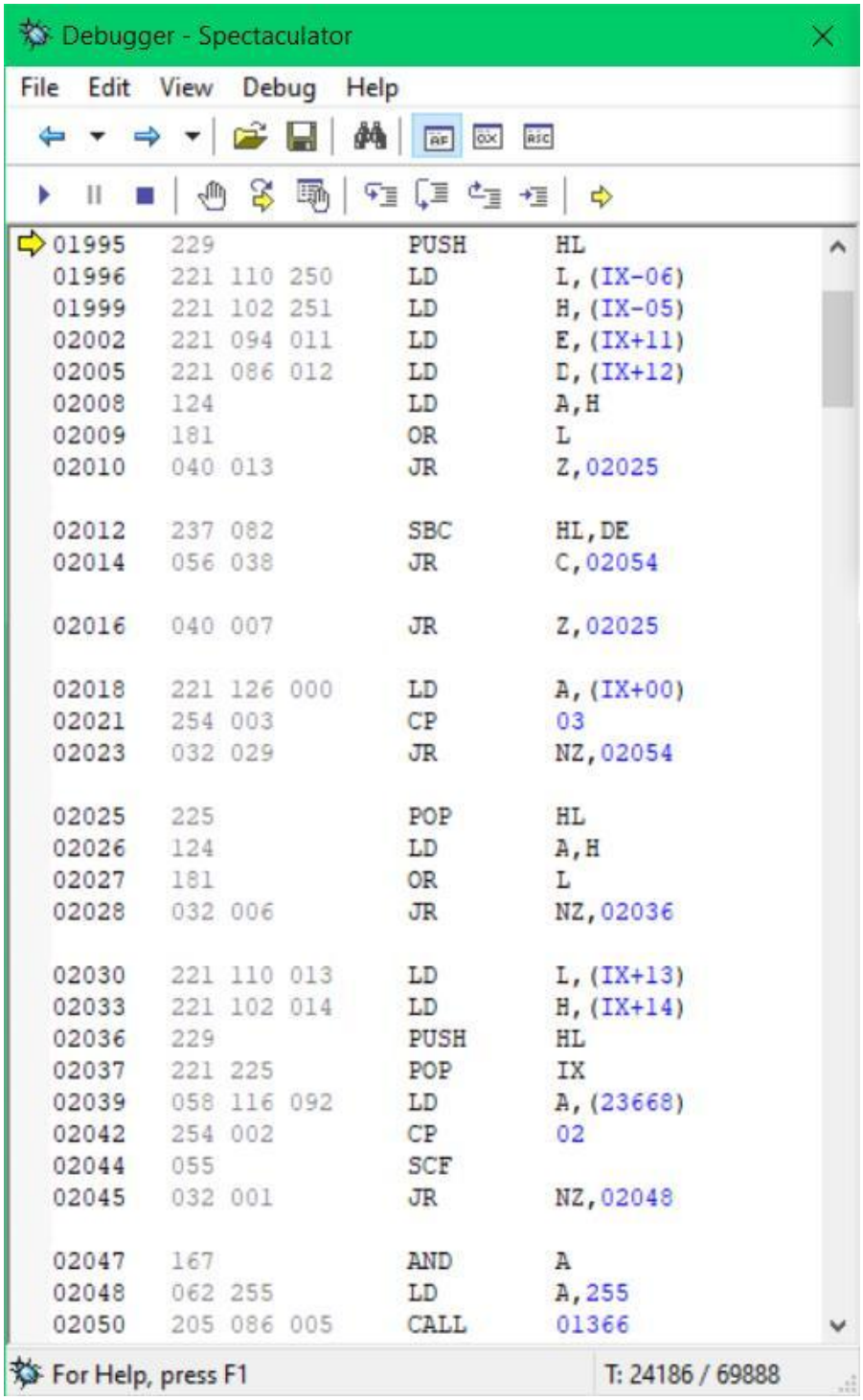


Рис. 346. Формирование входных данных для считывания заголовка.

Первым в «DE» попадает длина будущего блока, затем с адреса 2037, в «IX» добавляется стартовый адрес. Программно устанавливается C ☑ галочка/флаг «C», для загрузки данных и в «A» ставится маркер блока данных «255». Приготовления закончены.

В точке 2050, наконец, происходит вызов программы `LD-BLOCK (1366)` для считывания блока данных.

Итак, Стрелочка ныряет в 1366, поставив на вершину SP-башенки очередной важный адрес возврата в виде значения 2053. Сам SP-столбик в высоту становится 65360:

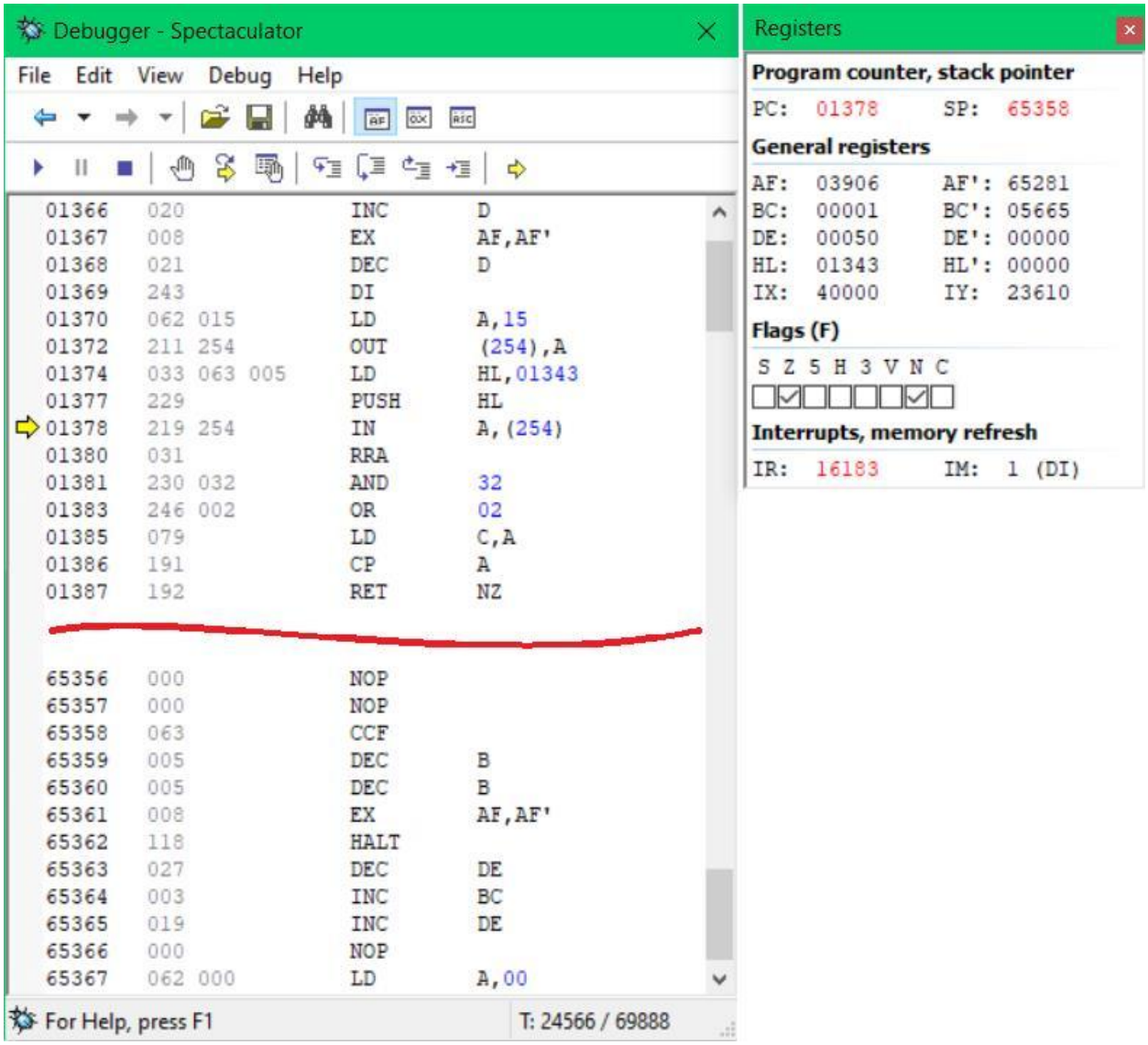


Рис. 347. Подпрограмма LD-BYTES (1366) и состояние SP-башенки перед началом загрузки.

И тут снова внимание! Для того, чтобы первый скрытый байт, с которого пойдёт отсчёт контрольной суммы, считать, но не записывать в память, требуется сбросить Z ☐ галочку «Z». Авторами данного шедевра было придумано «мудрое» решение сделать это временным приращением старшего байта длины загружаемого блока (`1366 INC D`). Ведь по задумке, команда `SAVE` не может записать блок данных, длиной свыше 65279 байт (255+254*256). Следовательно, максимальное значение «D» не может превысить 254. А если вы синтезировали блок данных свыше 65280 байт внешним IBM-методом через телепортационную щель, то из-за этих манипуляций произойдёт ошибочная запись незапланированного маркера 255 и сдвинет на байт всё остальные данные.

Если «D» не превышает значение 254, то ничего критичного не произойдёт. Погасив галочку «Z», она вместе с «C» запоминается для дальнейшей работы. Тем временем, значение «D» восстанавливается до прежнего значения (`1368 DEC D`).

На весь цикл «пищания-считывания» запрещаются прерывания. Временно ставится белая рамка. И вот тут в 1374 устанавливается ловушка, чтобы после выхода из считывания, Стрелочка заглянула на дополнительную подпрограмму `SA/LD RET (1343)` (*для возможности корректного выхода по BREAK и приведению в порядок экрана после загрузки*). На верхушку SP-столбика ставится третий подряд адрес возврата, и в высоту он

становится 65358. Таким образом, к этому моменту (1378) SP-башенка имеет следующую раскладку:

65358 – навязанная подпрограмма SA/LD RET (1343)
65360 – планируемый возврат, откуда только что пришли LD-BLOCK (2053)
65362 – возврат на проверку окончания строки STMT-RET (7030)
65364 – возврат на поверхность для выдачи сообщения в MAIN-4 (4867)
65367 – маркер фундамента SP-башенки (62) с паразитным нулём.

Далее опрашивается клавиша BREAK (1378) и на этом подготовка закончена. С адреса 1388, рамка начинает мигать красно-голубым цветом и ожидать короткого сигнала «Пи-и-и». Вскоре (1403) принимается и сам сигнал «Пи-и-и»:

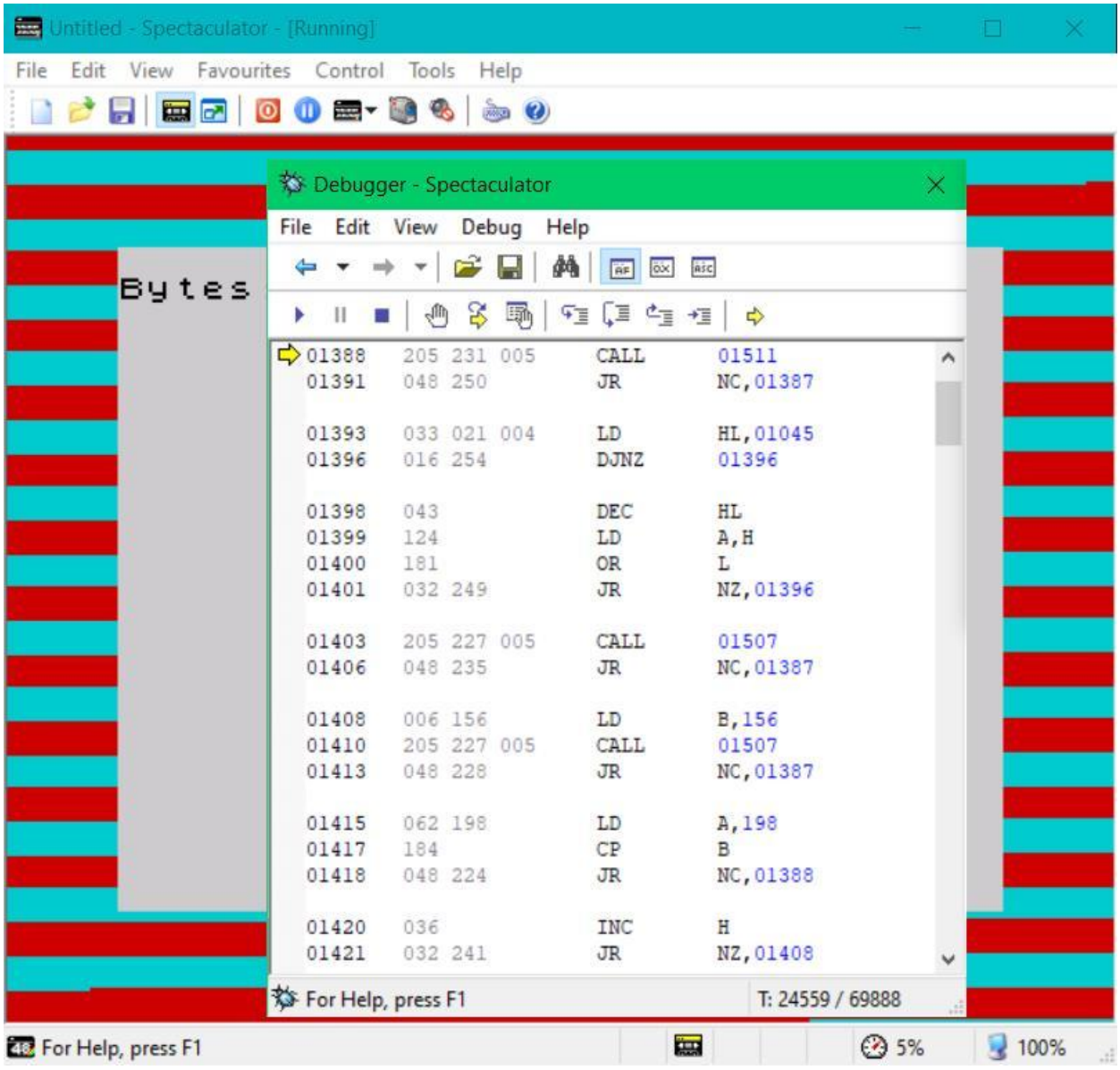


Рис. 348. Пилот-тон или сигнал «Пи-и-и-и».

Следом за ним (1423), следует короткий и незаметный для слуха сигнал синхроимпульса. И с 1439 начинается самый важный для данной главы этап – загрузка данных. Установив систему для подсчёта «псевдоконтрольной» суммы и подготовившись к приёму скрытого 0-го байта «255», Стрелочка перескакивает на адрес 1480, чтобы считать этот неучтённый байт без записи в основную память:

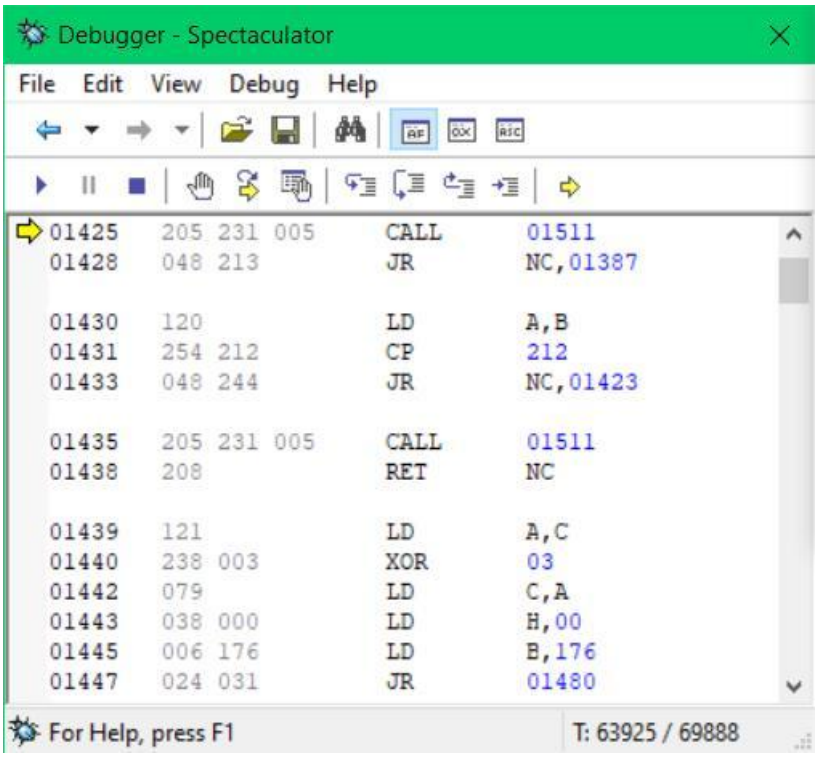


Рис. 349. Подготовка к загрузке вступительного скрытого байта «255».

Стрелочка впрыгивает на адрес 1480 и вызывает программу считывания сигнала бита данных `LD-EDGE-2 (1507)`. После записи первого невидимого служебного байта «255», Стрелочка попадает на адрес 1449, где подводятся итоги загрузки головного вступительного байта, и начинается подготовка считывания первого реального байта в память. В это время, красно-голубые полоски, начинают уходить с экрана и на смену им приходят сине-желтые:

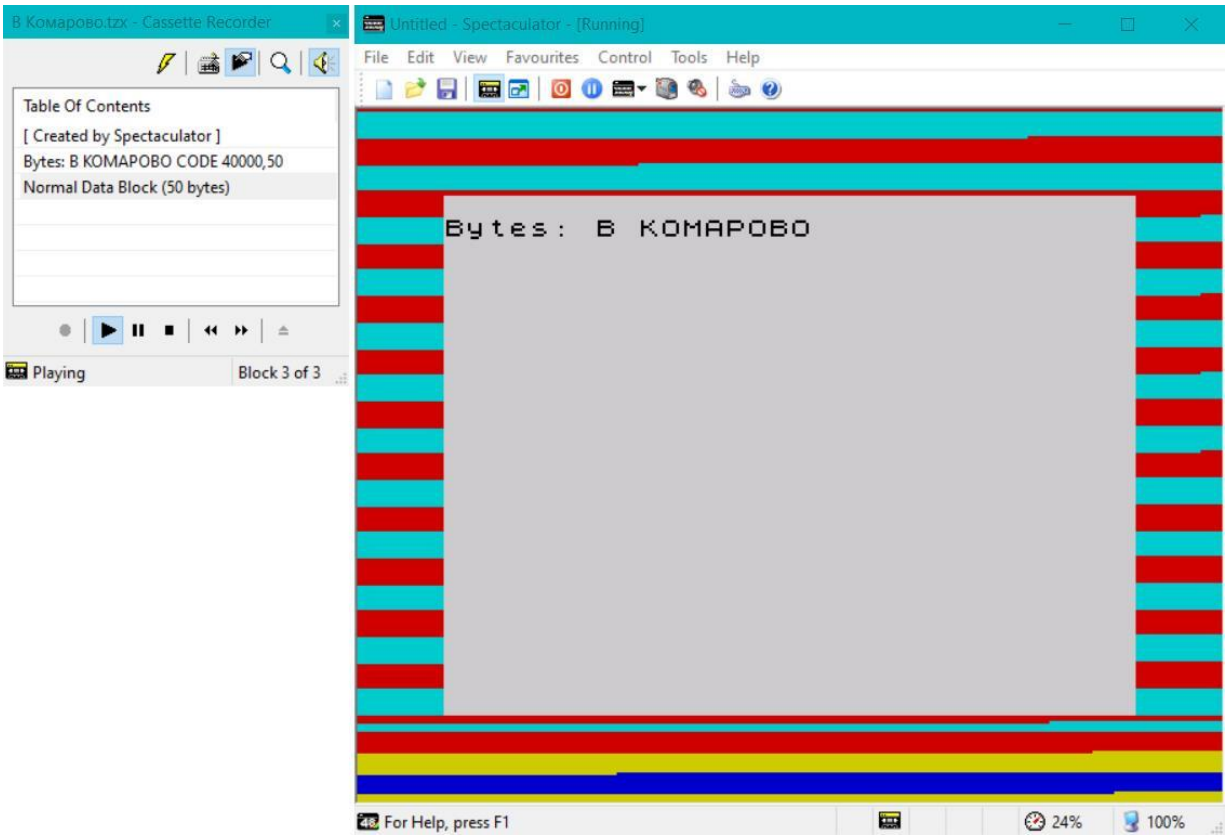


Рис. 350. Момент перехода пилот тона «Пи-и-и» в синхроимпульс и загрузку данных «Тшю-у-у».

Разобравшись с подготовительными работами, с 1476 начинается подготовка, а затем считывание первого реального байта блока данных, который нужно записать в память. Стрелочка снова попадает на адрес 1449 и в этот раз надолго. С момента записи в память первого и начала загрузки второго байта, цикл устаканивается. С неизменившимся значением столбика SP=65358, Стрелочка начинает ходить кругами с 1449 по 1501, пропуская по ходу некоторые куски, которые предназначены для считывания вступительного байта-маркера:

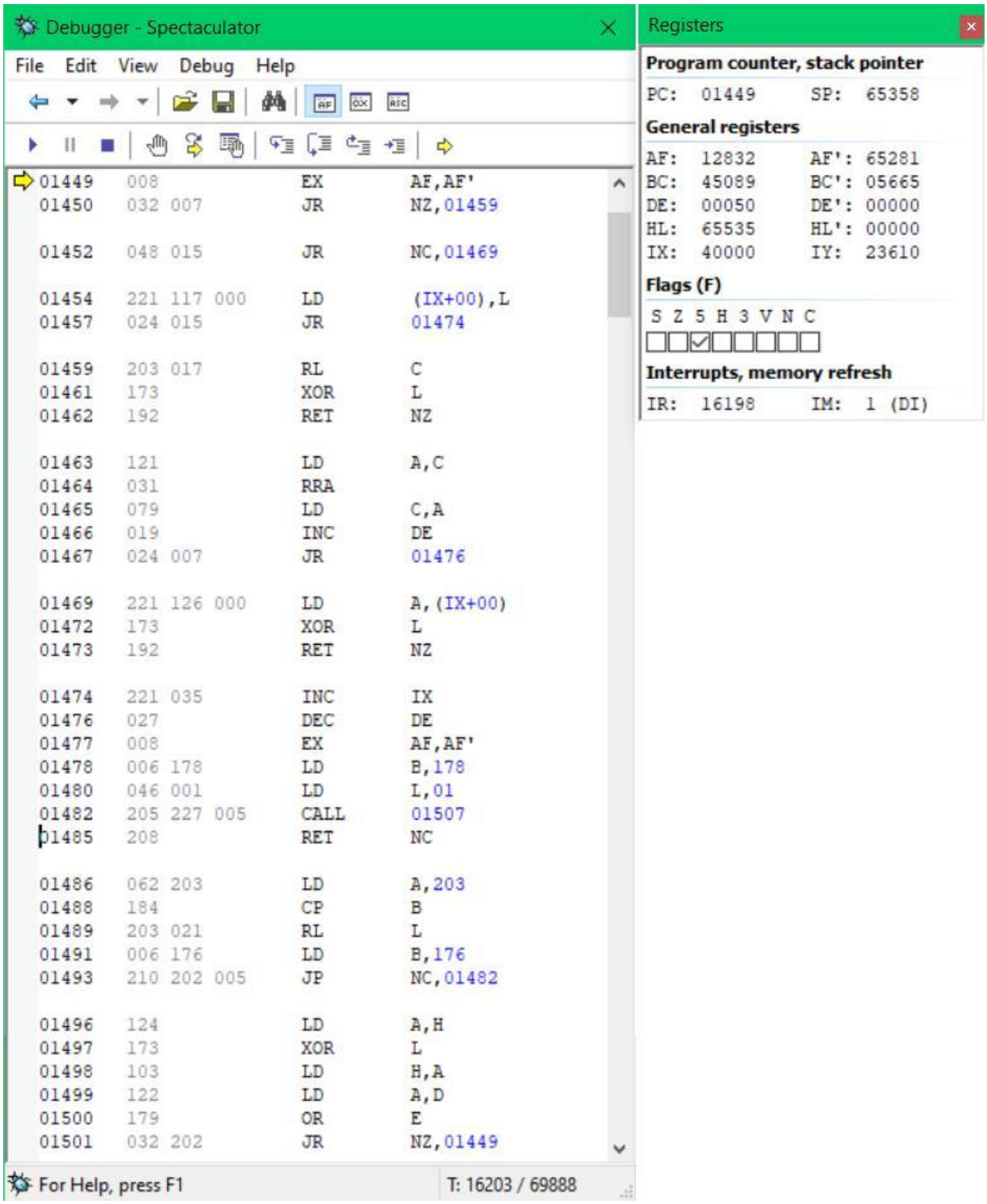


Рис. 351. Комплекс подпрограмм загрузки данных (1449-1501).

И вот этот цикл считывания данных, предлагаю рассмотреть чуть подробнее.

Первым делом в память по текущему значению «IX», из «L» записывается склеенный и собранный из 8-ми кусочков байт, считанный в предыдущем цикле. Приращивается адрес загрузки байта (1476) и уменьшается количество загружаемых байт. (*Таким образом, в реальном времени можно перенаправить поток загружаемой информации в произвольное место, не прерывая загрузку*).

После подготовки, с 1482 Стрелочка отправляется в подпрограмму `LD-EDGE-2 (1507)` ждать и вылавливать очередной сигнал бита информации. Алгоритм её работы будет в виде такой петли:

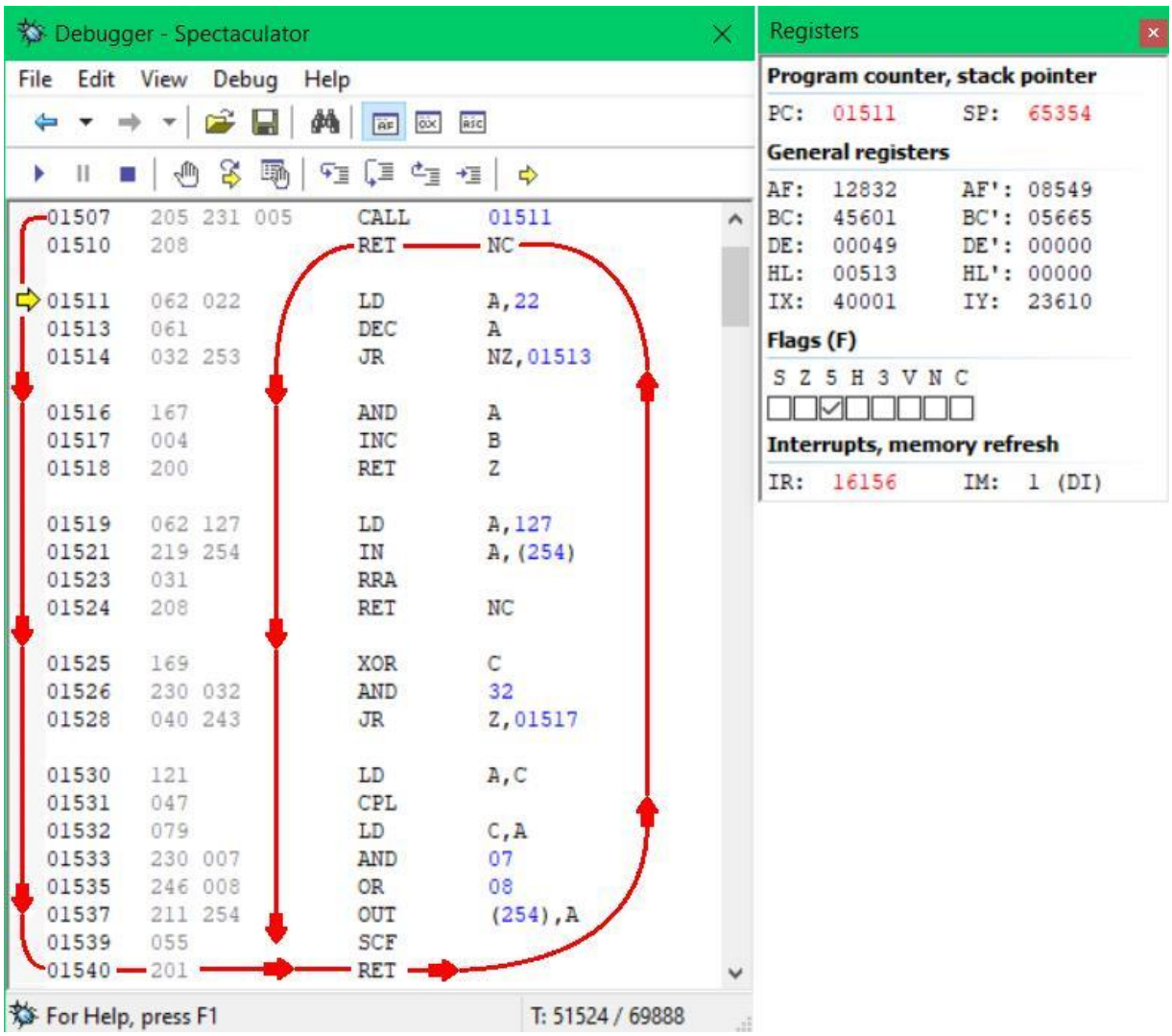


Рис. 352. Программа ловли битов. Процесс считывания бита для записи 2-го байта.

Итак, это сердце загрузки, иначе говоря, универсальная программа ловли и считывания неделимого сигнала, который равен половине бита или одной полоске. Она же – последняя инстанция и самое глубокое вложение. В контексте данной главы хочу обратить внимание на следующие особенности работы данной программы.

Заходя на эту небольшую подпрограмму, командой «`IN A, (254)`» по адресу 1521 одновременно ловится сигнал и попутно опрашивается клавиша `BREAK`.

Посмотрите, как интересно она дважды сама из себя вызывается. Таким образом, проходя программу, Стрелочка своим маршрутом как бы рисует заваленную на бок цифру «6» чтобы принять две полоски (синюю и желтую), которые равны одному биту. Столбик «SP» кратковременно поднимается на две пары значений и становится 65354

После успешного считывания сигнала, происходит выход в основной цикл программы (1449-1501). При возврате SP-столбик восстанавливается до прежнего значения 65358.

Возвратившись с добычей, Стрелочка проверяет выход на нажатие BREAK (1485 RET NC) и заботливо подклеивает пойманный бит к байту в «L», после чего снова возвращается ловить следующий сигнал. Цикл продолжается, пока из битов не соберётся байт.

К адресу 1496 в «L» собрался свежесчитанный байт, в «H» хранится значение «псевдоконтрольной» суммы от расчета предыдущих. Результат XOR нового значения с суммой запоминается вместо старого. Проверяется количество оставшихся байт для загрузки. Если они ещё имеются (DE > 00000), то Стрелочка возвращается на верхнюю точку цикла, и записав в память текущий байт, приступает к считыванию следующего.

Обратите внимание: запись очередного байта в память происходит после выхода из программы принятия сигнала, когда SP-столбик опускается до прежнего значения 65358. В этот момент, 4 байта по адресам 65364-65367 являются отработанным мусором. Таким образом, сплошная запись данных сквозь SP-область, вплоть до значения 65368 не вызовет сбоя процесса. Значения, записанные в эти ячейки, просто затрутся очередным вызовом CALL 1507, на отметке 1492.

Настаёт торжественный момент, когда в память записывается последний значимый байт программы. Счётчик остатка «IX» обнуляется. По инерции Стрелочка проходит весь цикл до конца и принимает ещё один неучтённый технический байт, который является заранее рассчитанным значением «псевдоконтрольной» суммы. Обычно он сформировывается и приписывается к концу блока данных командой SAVE, но остаётся невидимым.

Запомнив его, Стрелочка последний раз проверяет, сколько нужно загрузить. Убедившись, что там пусто и Z ☑ горит Z-галочка, по команде JR NZ, 1449, Стрелочка больше не возвращается назад, а проваливается вниз.

Всё! Цикл загрузки окончен и можно прекращать дальнейшее считывание сигнала. Начинается подведение итогов:

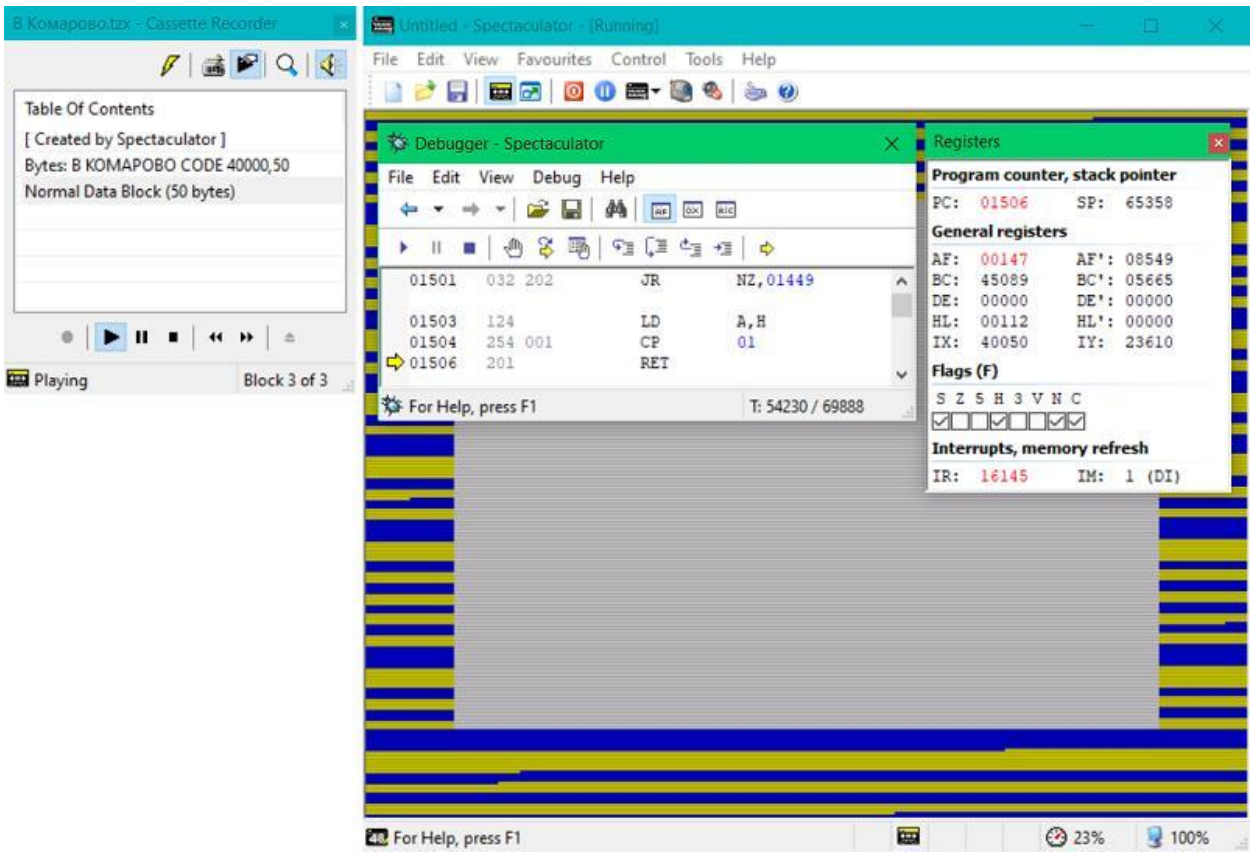


Рис. 353. Формирование «псевдоконтрольной» суммы и оценка правильности считанных данных.

Берётся рассчитанная во время загрузки псевдоконтрольная сумма и сравнивается с только что считанным готовым результатом. Если они совпали, то всё хорошо и в качестве экспертного заключения (`1504 CP 01`) появляется ☑ галочка «C». Если значения не совпали, то галочка не выставляется.

После этого, независимо от результата происходит возврат…

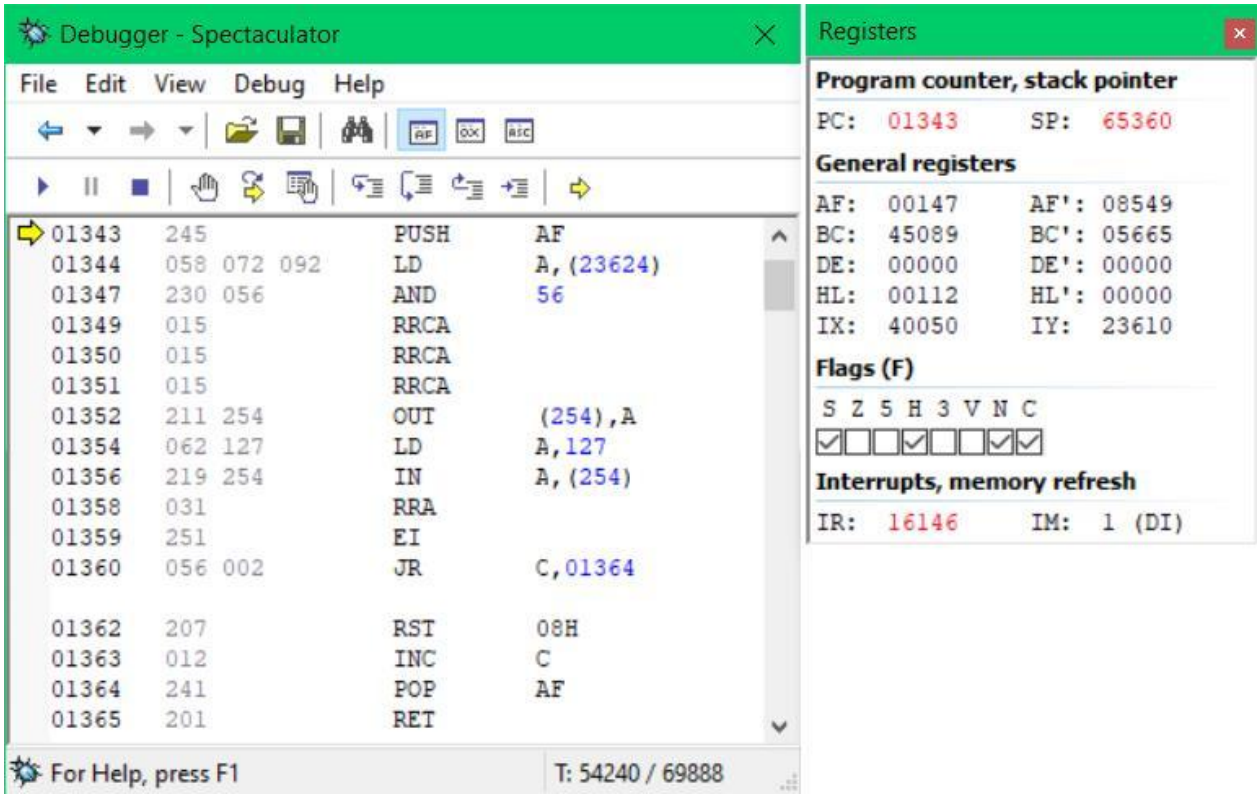


Рис. 354. Подпрограмма SA/LD-RET (1343). Восстановление параметров после загрузки.

…на программу `SA/LD-RET (1343)`, которую ей дополнительно навязали еще в самом начале загрузки на отметке 1374. Отсюда начинается постепенный каскадный выход на поверхность. ♥ Посмотрите, что тут происходит.

Первым делом при попадании в эту подпрограмму, башенка SP укорачивается на 2 этажа и становится 65360. В виртуальной сумочке у Стрелочки лежит отчёт о считывании блока, в виде ☑ галочки «C». Сдав отчет в камеру хранения, она восстанавливает цвет рамки, который был до работы команды `LOAD`. Опрашивается клавиша `BREAK` и восстанавливаются прерывания, необходимые для опроса клавиатуры в BASIC режиме.

Если сейчас (*или во время приёма сигнала в точке 1521*) была нажата клавиша `BREAK`, то из библиотеки берётся комплект сообщения №12, срезаются неиспользованные остатки SP-башенки (путём забора значения из `ERR_SP (23613)`) и происходит выход на поверхность, где в 4867 выводится сообщение об ошибке:

«`D BREAK - CONT repeats`»

Если нажатия `BREAK` ни сейчас, ни ранее зафиксировано не было, Стрелочка забирает назад отчёт о правильности считывания данных. По `RET` в 1365 она возвращается в 2053, откуда когда-то спускалась на запись блока данных:

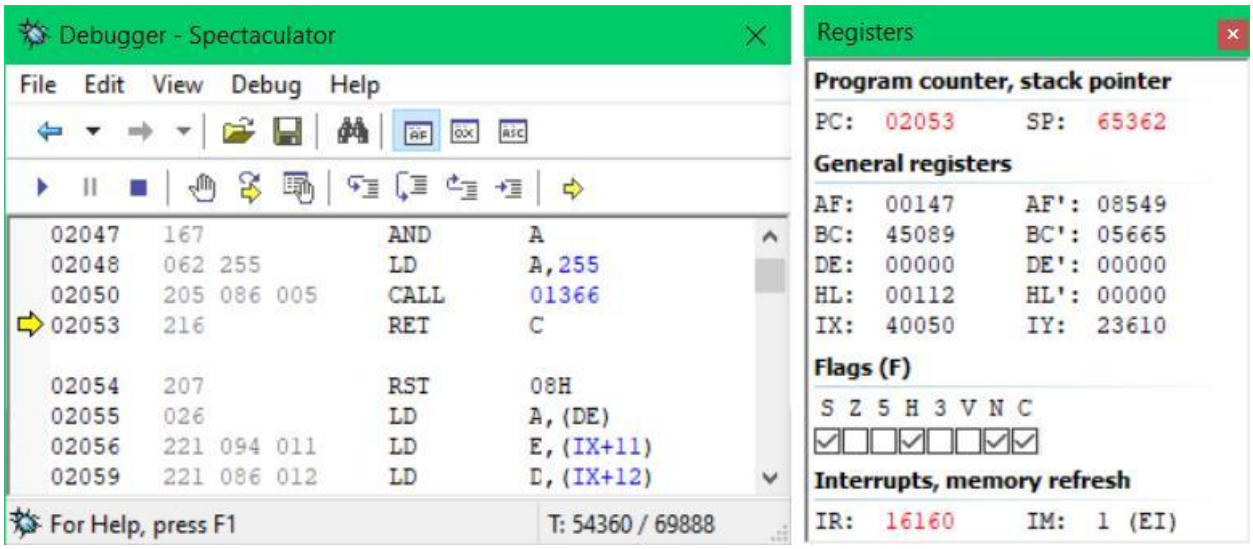


Рис. 355. Перед возвратом из комплекса подпрограмм записи/считывания.

Знакомый кусок программы? Снова столбик SP укорачивается на два кубика и становится 65362.

Итак, проанализировав состояние галочки «C», выносится окончательное решение. Если её нет C☐, значит, считывание произошло с ошибкой. Следовательно, нужно взять сообщение №26, и срезав SP-столбик значением из `ERR_SP (23613)`, выйти на поверхность в 4867. Там вывести на экран:

«`R Tape loading error`»

и вернуться в режим ожидания BASIC.

Ну а если галочка «C» присутствует C☑, то всё в порядке и можно выныривать на следующий уровень в `STMT-RET (7030)`:

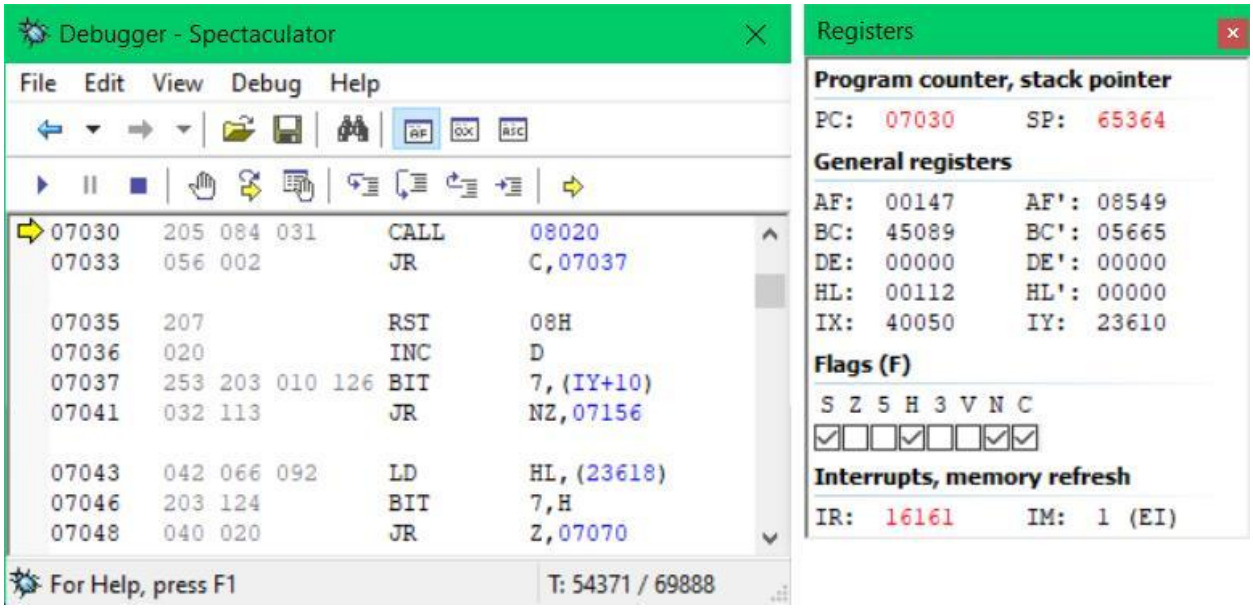


Рис. 356. Подпрограмма STMT-RET. Возврат в BASIC-систему.

И вновь Стрелочка возвращается во владения BASIC. Столбик SP становится 65364 и до выхода на поверхность остаётся всего один этаж.

После того, как команда LOAD выработала свои свойства и выдохлась, она снова рассматривается в составе выполняемой строки LOAD ""CODE [ENTER] теперь уже для формальной процедуры завершения. Быстро сбегав на опрос проверки нажатия «BREAK+CAPS SHIFT», Стрелочка возвращается назад. Если галочка «C» при опросе потерялась C☐, значит, была нажата комбинация клавиш «BREAK+CAPS SHIFT». В этом случае, готовится сообщение №20, и по известной дорожке в обход через адрес 00008, происходит возврат в 4867. Там выводится сообщение:

«L BREAK into program»

Если ничего не нажималось, то перепрыгнув вывод сообщения, проверяется бит №7 NSPPC (23620) на наличие следующего оператора в строке. А там заглушка «255», которая установилась еще при проходе через 7131 идентификацией команды.

Убедившись, что операторов, к выполнению нет (*а если бы были, проверилась бы ячейка 23618 на номер строки и в случае нахождения там номера, произошёл бы автозапуск*), то в текущей ситуации происходит выход на последнюю проверку в STMT-NEXT (7156):

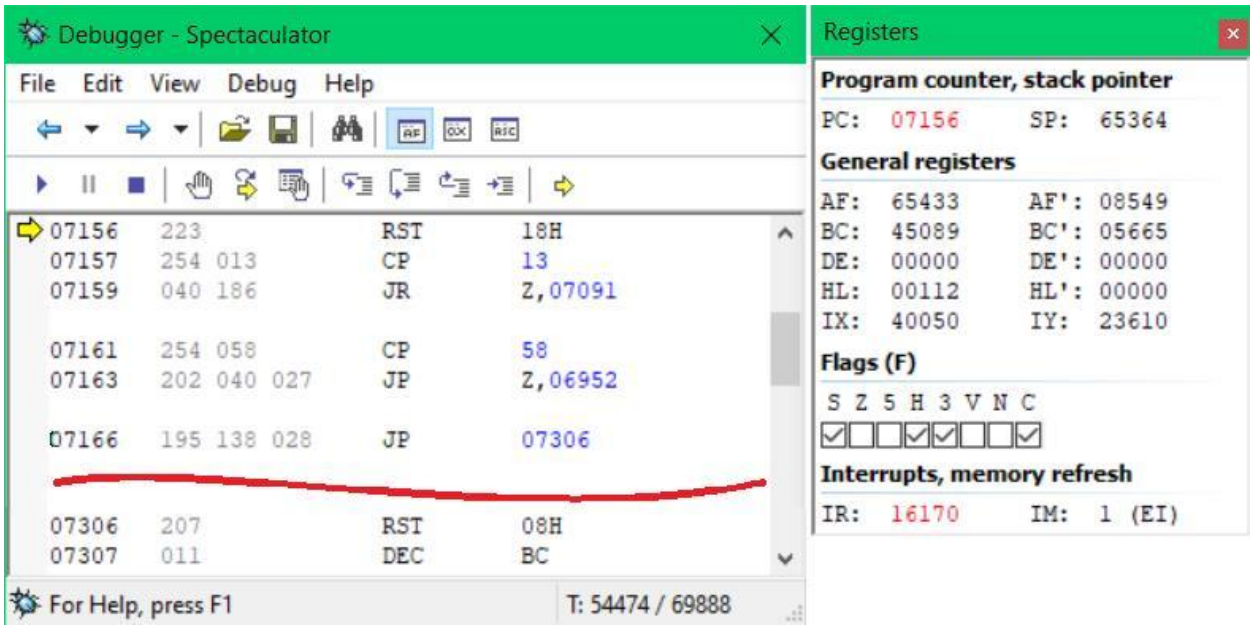


Рис. 357. Подпрограмма STMT-NEXT (7156-7166). Последняя проверка на ENTER в конце строки.

И тут уже были. Узнаёте? Только теперь маршрут получается совершенно иным. Через «RST 18» Стрелочка берёт символ, на который указывает внутренний курсор CH_ADD (23645). Вернувшись, для порядка она проверяет наличие ENTER в данной ячейке (7157 CP 13). Как вы помните, после опознания строки, по адресу 1743 (RST 20) курсор последний раз перед загрузкой, переключился на ячейку 23760 и с тех пор к нему обращений не производилось.

Убедившись, что в конце отработанного комплекса команд всё еще ENTER ♥ (*если конечно в процессе загрузки туда не записалось что-то другое*), можно переходить к заключительному этапу обработки строки на подпрограмму LINE-END (7091).

В случае обнаружения чего-то отличного от двоеточия, произойдет переход на 7306, где задаётся сообщение №11 и выбравшись через «RST 8» на экран выведется сообщение:

«C Nonsense in BASIC».

Итак, Стрелочка оказалась в подпрограмме `LINE-END` по адресу 7091:

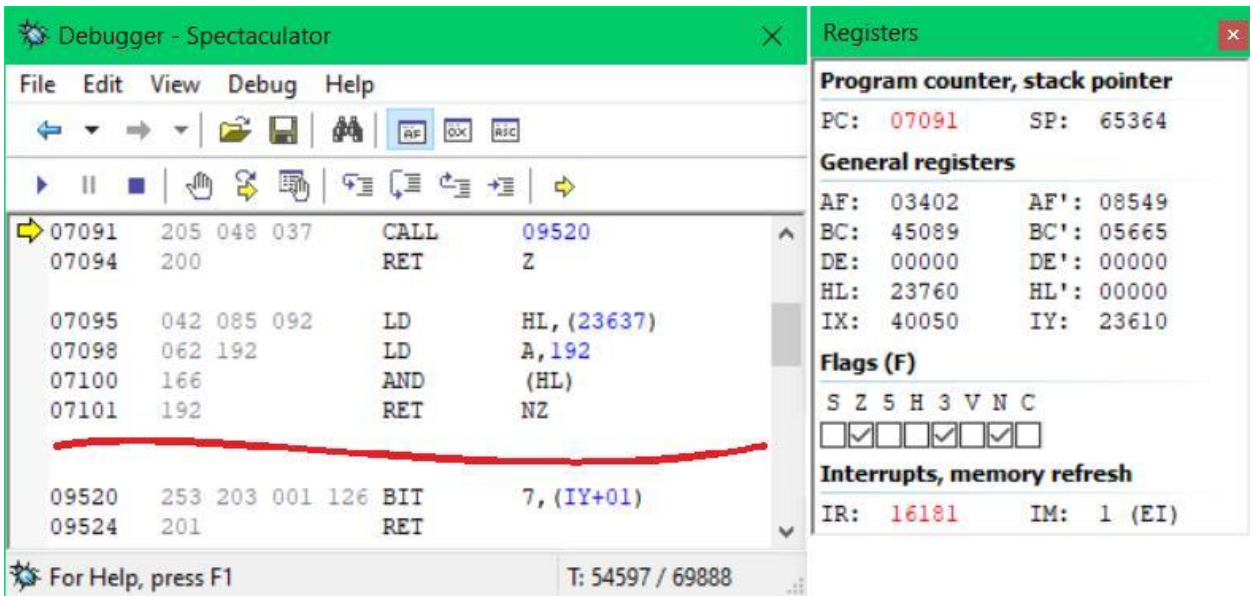


Рис. 358. Подпрограмма LINE-END (7091-7101). Проверка на отсутствие выполняемых строк.

Сходу вызывается короткая подпрограмма `SYNTAX-Z (9520)`. В ней Стрелочка смотрит состояние бит-тумблера №7 переменной `FLAGS (23611)` и сразу возвращается назад. Если он отключен, значит, строка теоретически проверяется синтаксическим анализатором при вводе. Если включен, как в данном случае, то можно продолжать.

На заключительном отрезке Стрелочка берёт переменную `NXTLIN (23637)`. А в ней содержится адрес 23761, указывающий на первый байт номера строки. Это значение было забито в самом начале подготовки к выполнению команды `LOAD`, по адресу 7121. В текущей ситуации там находится маркер «128». Он стоит сразу за командой `ENTER` в выполненной строке `LOAD "" CODE`.

Осталось проверить, что за число там стоит. Если значение оказалось больше 63-х, значит, программных строк нет. Ведь по убеждению разработчиков данной прошивки, номер строки никак не может оказаться больше 16384 (64*256+XX). По гашению первых 6-ти разрядов числа (63) смотрится результат и принимается решение.

Если там ничего не оказалось, то программа продолжится и начнётся выполнение строки. ♥

В данном случае по указанному адресу (23761) стоит маркер «128», что явно больше 63 и намекает на то, что программных строк больше нет. Значит можно смело выходить на поверхность в `MAIN-4` по адресу `4867` для печати сообщения:

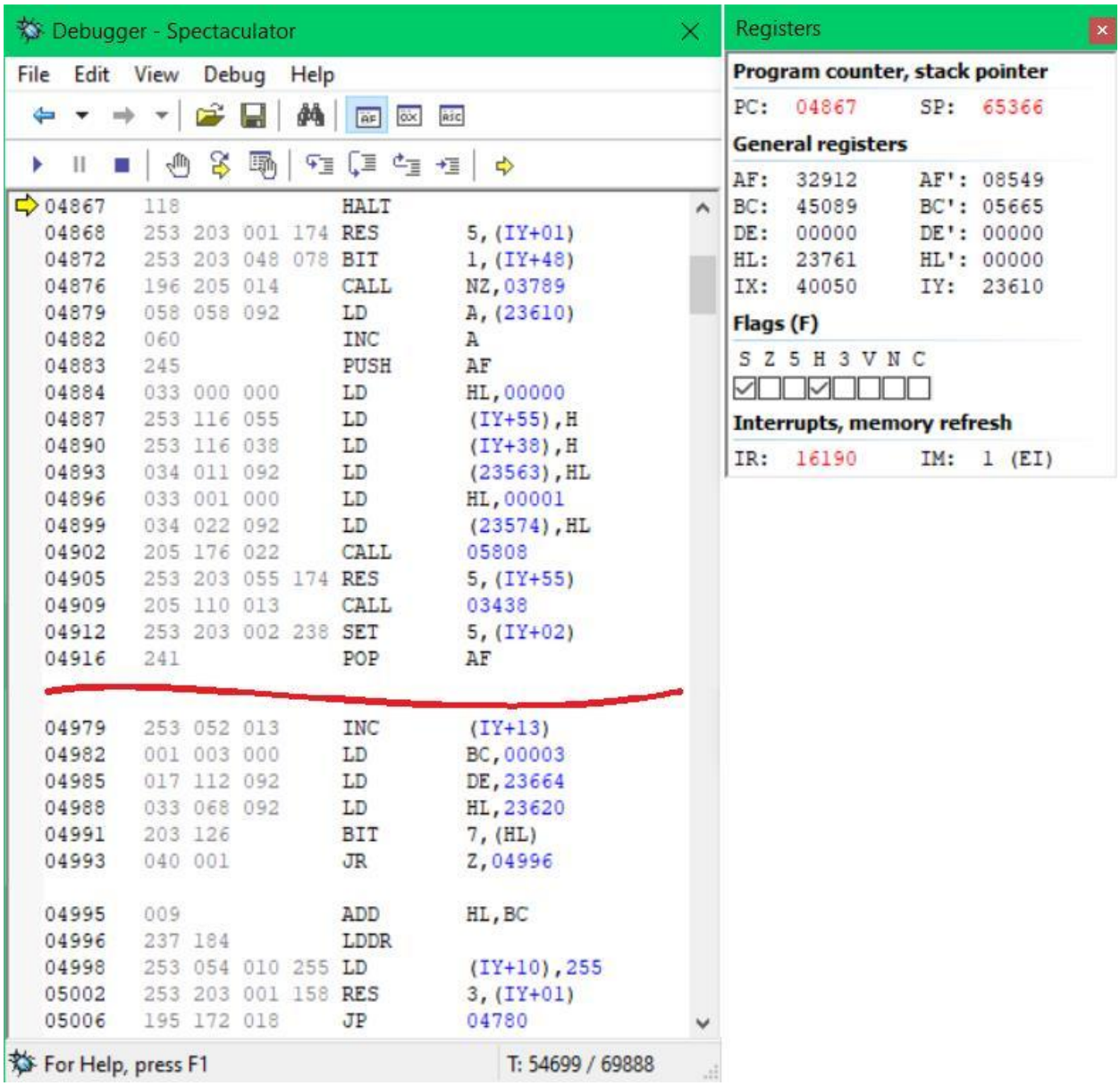


Рис. 359. Выход на поверхность в подпрограмму MAIN-4 (4867). Подготовка к печати сообщения.

И вот он долгожданный выход. Столбик SP израсходовался до основания и стал равен 65366. ♥

После вывода сообщения «▣ OK.» Стрелочка отправляется в подпрограмму режима ожидания `MAIN-2` по адресу `4780`, замкнув полный круг:

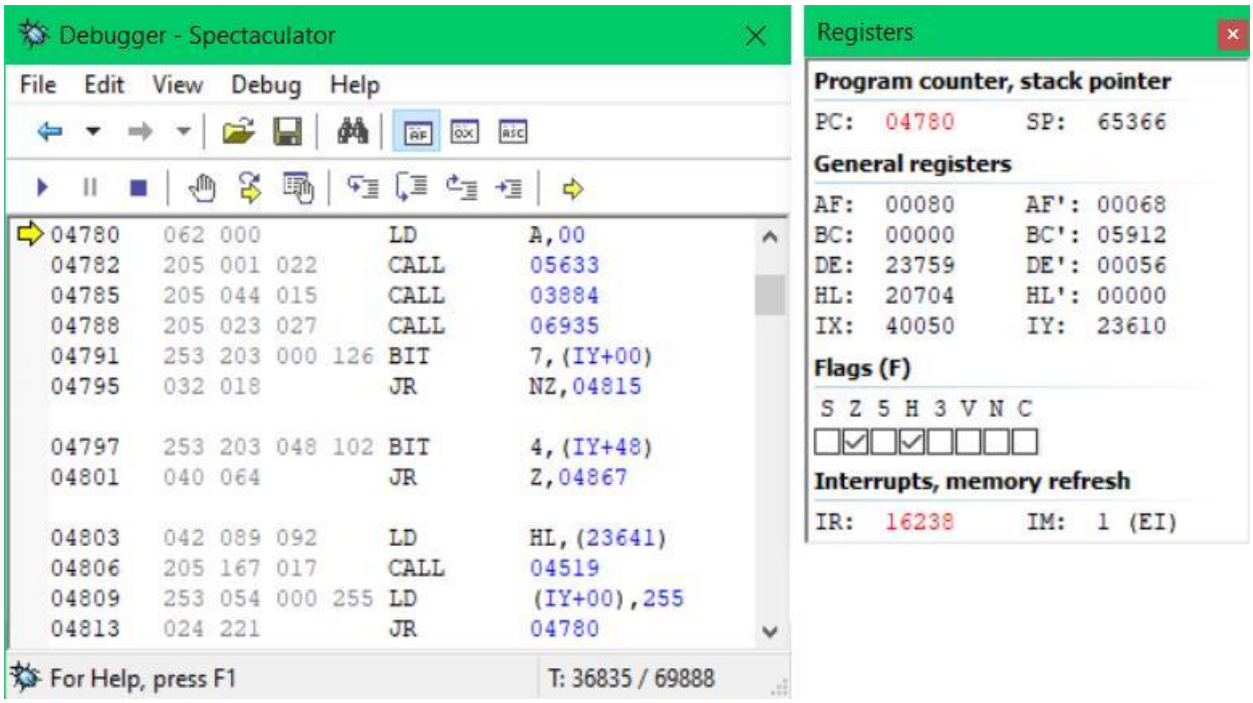


Рис. 360. Подпрограмма MAIN-2 (4780-4813). Дорога к дому в родной EDITOR (3884).

Вот такие дела.

Что же такое `LOAD ""CODE [ENTER]` ? Да это всего лишь безномерная выполняемая строка в области [E_LINE] и ничего более. От `BORDER 0 [ENTER]` она отличается только тем, что дольше выполняется. Во время выполнения `LOAD`, происходит стандартный набор действий, который сопровождает любую другую команду: распознавание, переразметка областей, контроль и сопровождение выполнения, и возврат в главный цикл:

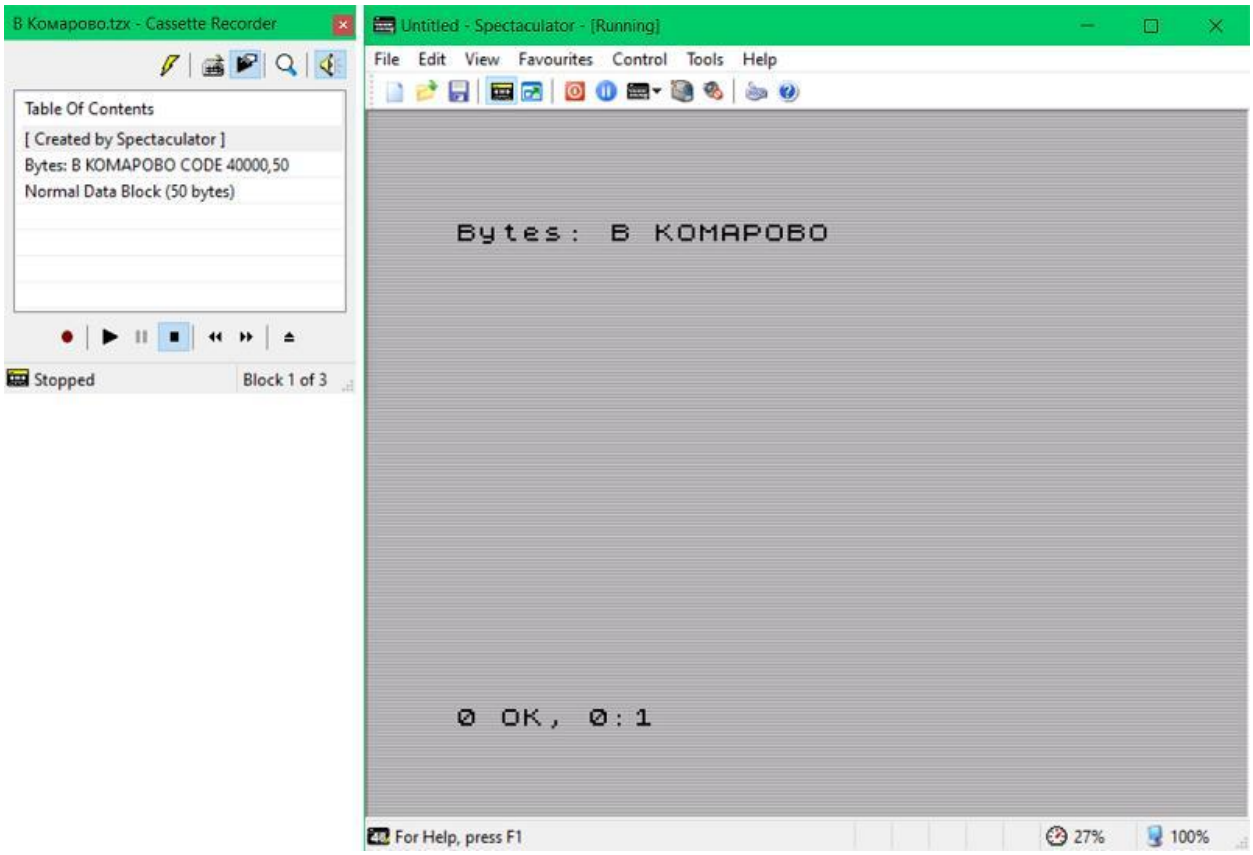


Рис. 361. Окончание загрузки блока «Bytes:».

Обратите внимание, что для считывания заголовка и данных вызывается одна и та же программа, с немного разными входными параметрами. Снова убеждаемся, что заголовок, это такой же блок данных, но всегда имеющий длину 17 байт, длинный пилот-тон и плавающий адрес загрузки [WORKSP]+17. В гонке за экономией байт и стремлению унификации, программу сделали универсальной для всех команд записи и считывания, из-за чего внутри полная каша.

Глава 38
Классическая загрузка блоков «Bytes:»

Краткое содержание: синтез двухблоковой загрузки

После такого детального разбора, просто необходимо научиться загружать блоки «**Bytes:**» всеми возможными методами.

Откройте *Spectaculator* и из папки «*Программы*» мышкой перетащите файл «*Моnед в сарае.tzx*», который вы создавали двумя главами ранее, в окошко виртуального магнитофона «*Cassette Recorder*»:

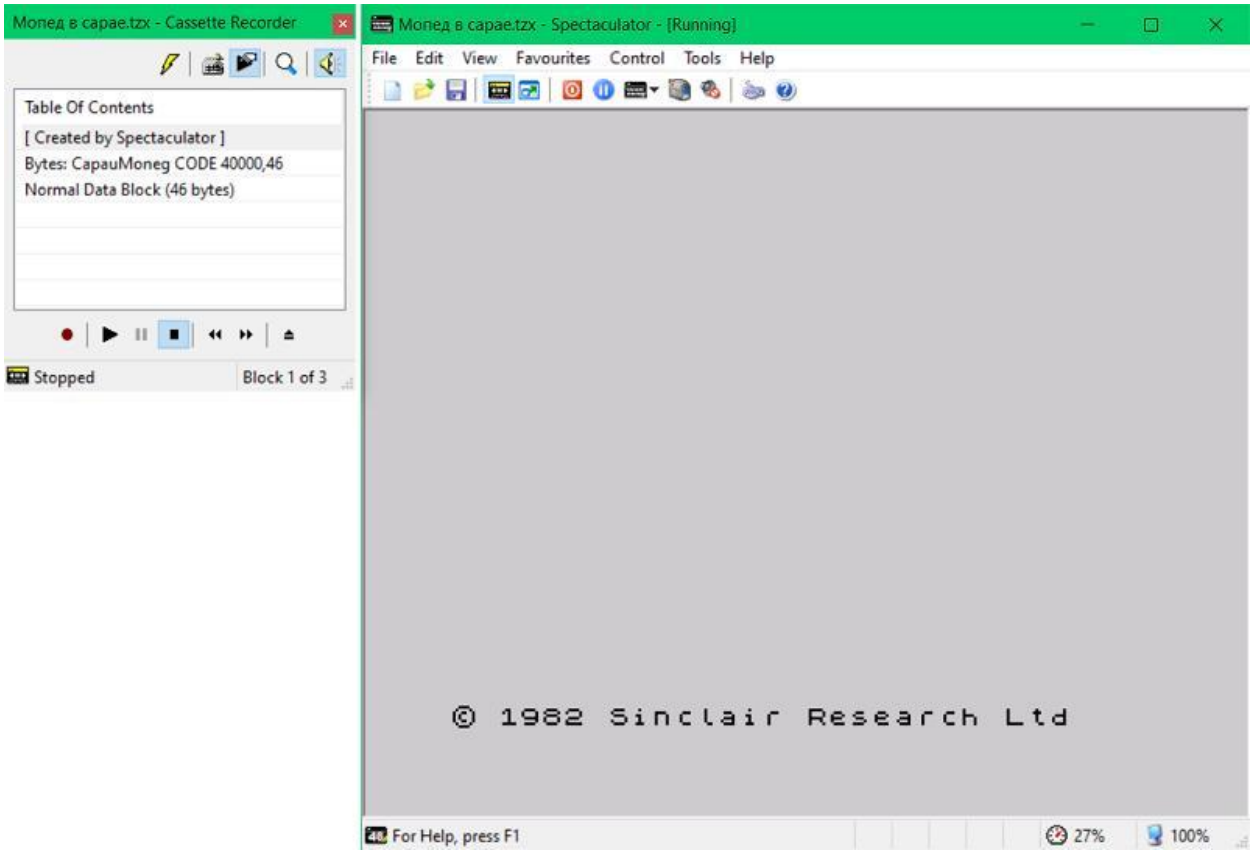


Рис. 362.

Кассета со всеми блоками немедленно появилась в окошке.

А теперь введите следующую программу считывания блока «**Bytes:**» классическим методом:

```
Debugger
Dec
Go To 23400
23400 ← 3 255 0 0 0 0 0 0 0 0 0 0 0 0 0 0
23610 ← 255
23613 ← 65364
65364 ← 4867
```

```
HL ← 0
IX ← 23400
SP ← 65364
PC ← 1889
Trace
Cassette Recorder [Play]
```

После ввода не забудьте нажать кнопку ▶ «*Play (Ctrl+Space)*» магнитофона, если у вас не включена соответствующая настройка «*Auto Play / Stop*».

Рамка ностальгически замигала, и вскоре раздался до боли знакомый «Пи-и-и-и-и-и-и-и Тщю-у-у-у-у…», а следом щемящие звуки мелодии загрузки:

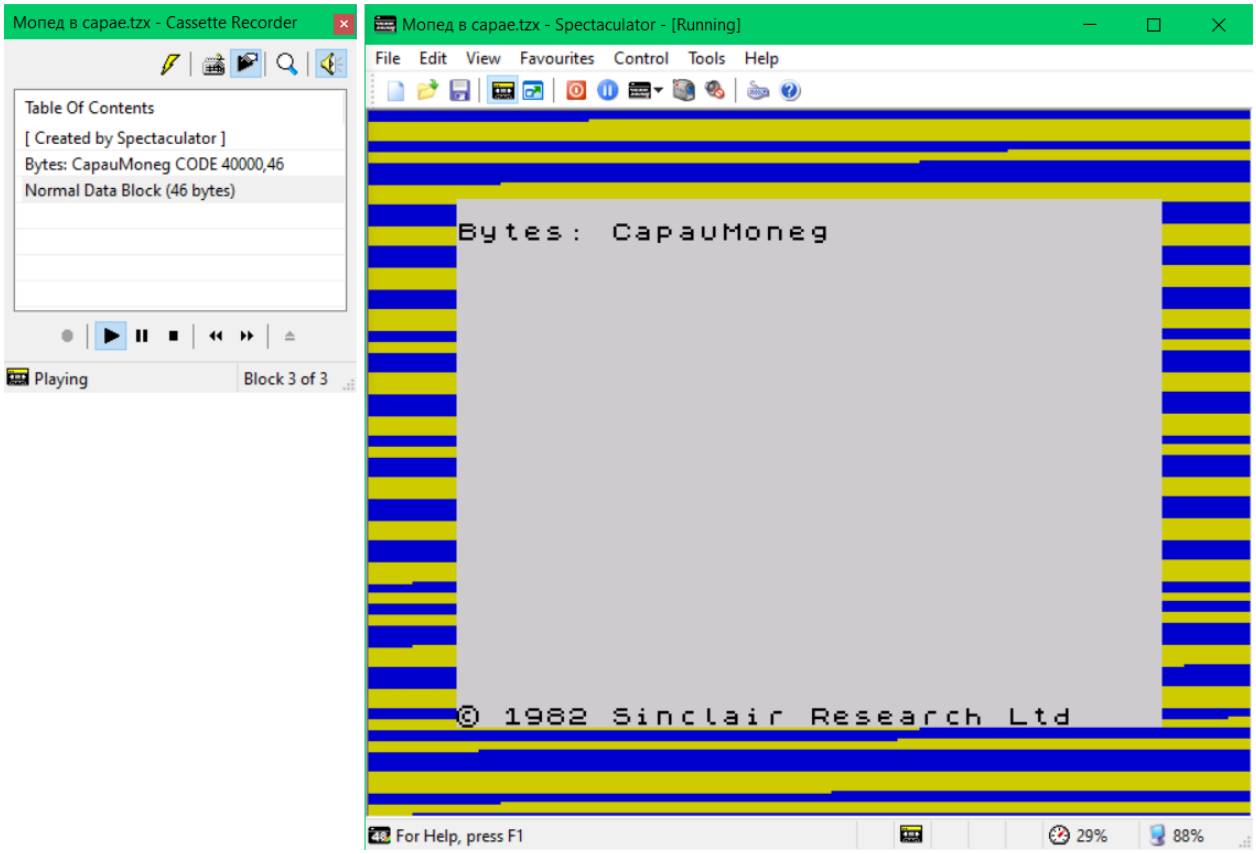


Рис. 363. Процесс загрузки комплекса «Bytes:» искусственным методом.

Для проверки натуральным способом наберите и введите строку `RANDOMIZE USR 40000`:

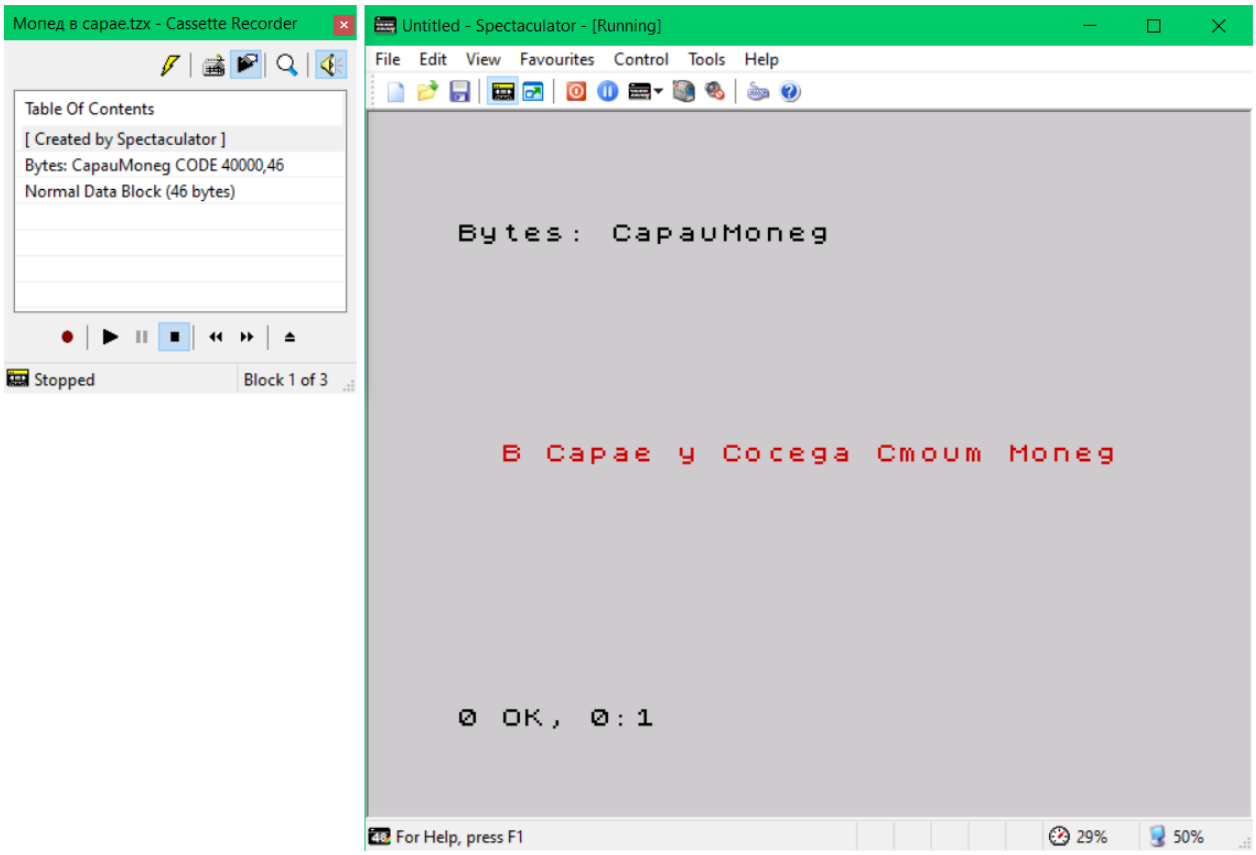


Рис. 364.

Как говорится в нетленном шедевре:

```
    PRINT USR 32500
то получите ответ: 99
  Возврат в  бейсик-программу  осуществляется обычным образом по
командe микропроцессора RET.  В машинной программе вы  не  должны
использовать регистры IY и I.
```

Рис. 365. Фрагмент из главы 26 «Использование машинных кодов» самоучителя по BASIC.

Чуть перефразирую. Набрав `RANDOMIZE USR 40000` получите ответ: «В сарае у соседа стоит мопед».

Возврат в бейсик программу осуществится обычным образом по команде микропроцессора `RET`, которая встретится в конце подпрограммы вывода текста. В машинной программе вы можете использовать абсолютно любые регистры, включая «IR», «SP» и «PC», не говоря уже об «IY». Короче, что захотите, то и используйте.

Можно считать эту программу так, что она будет автоматически выполняться после загрузки. Для этого вспомните главу «*Основы маршрутизации Стрелочки*» и нажмите «*Reset*». После сброса кнопкой мыши или курсорными клавишами в «*Cassette Recorder*» выделите для загрузки блок «*Bytes: CapauMoneg CODE 40000,46*», если он по причине настроек сам не встал на исходную позицию:

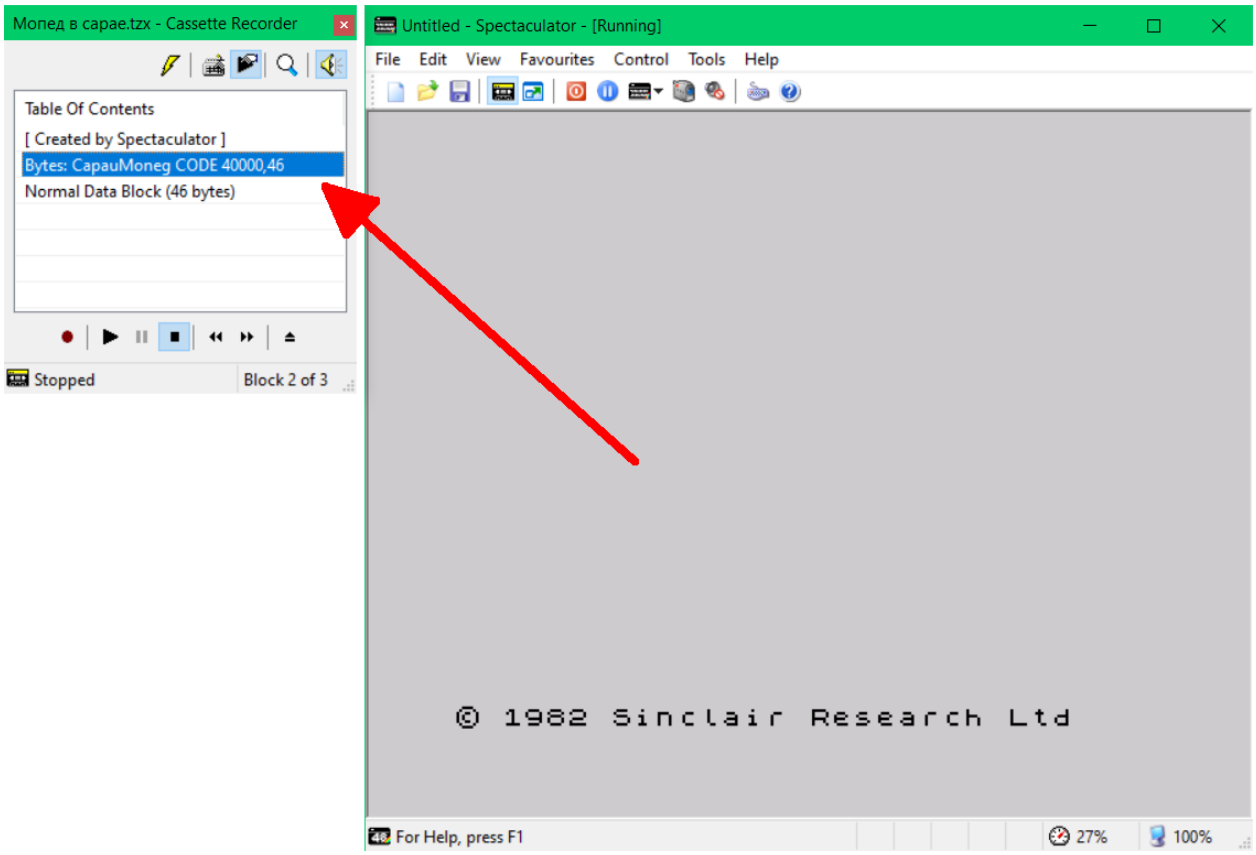


Рис. 366.

А потом выполните следующий алгоритм:

```
Debugger
Dec
Go To 23400
23400 ← 3 255 0 0 0 0 0 0 0 0 0 0 0 0 0 0
23610 ← 255
23613 ← 65364
65362 ← 40000 4867

HL ← 0
IX ← 23400
SP ← 65362
PC ← 1889
Trace
Cassette Recorder [Play]
```

Это приведет к тому, что вначале будет загружена и автоматически выполнена бейсик-программа, которая, в свою очередь, загрузит и выполнит программу в машинных кодах. Книга "Искусство схемотехники" П. Хоровиц, У. Хилл Мир,1986, том 2, стр.579-58 0.

Рис. 367. Окончание фрагмента из главы 26 «Использование машинных кодов» самоучителя по BASIC.

Немного не так! Это приведёт к тому, что вначале будет выполнена загрузка программы «*CapauMoneg*», которая, в свою очередь закончится, выйдет по команде RET и выполнит эту загруженную программу в машинных кодах по адресу 40000:

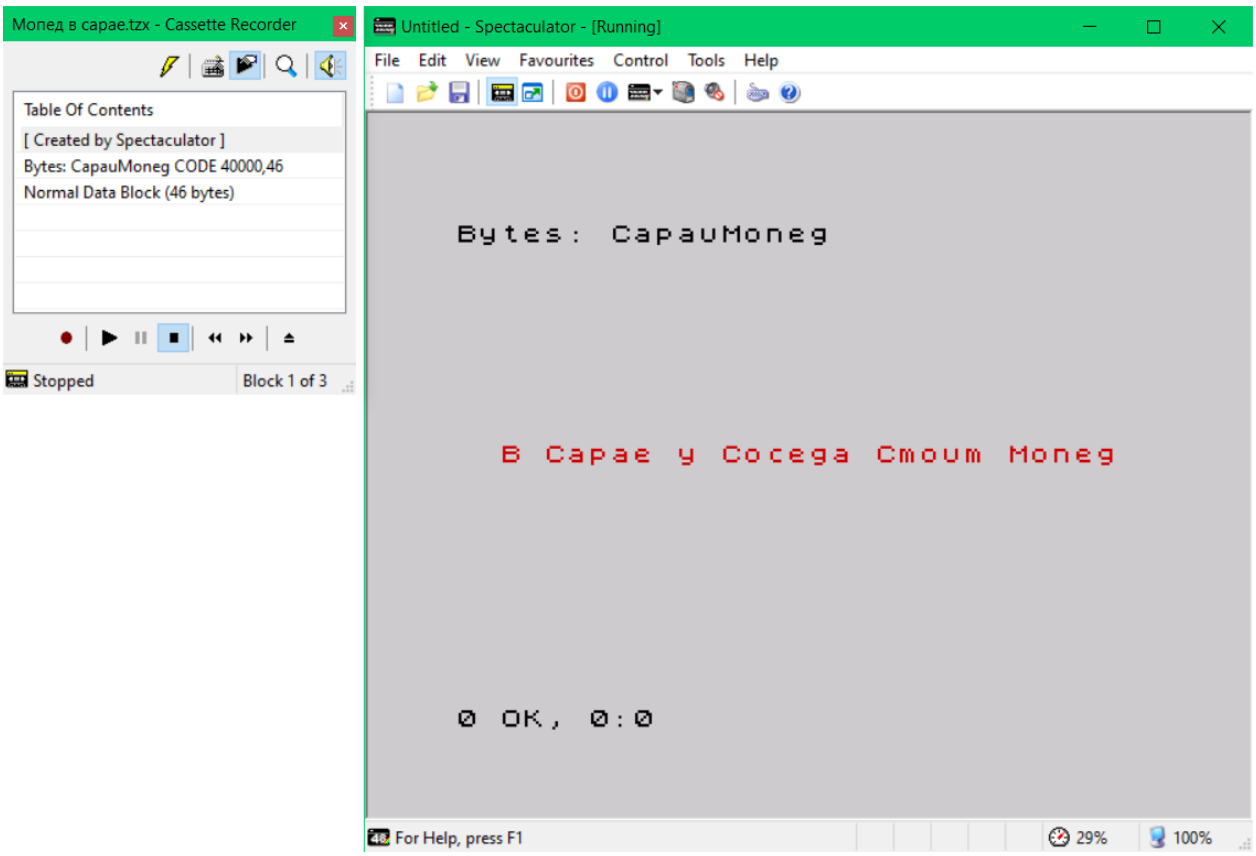


Рис. 368.

Вот как-то так.

Глава 39
Загрузка картинки без повреждения нижних строк

Краткое содержание: WAIT-KEY, TV_FLAG, системная пауза, загрузка

Каждый сталкивался с проблемой при загрузке картинки с нижними строками. После окончания загрузки нижние строки подло затираются, и туда выдаётся сообщение «0 OK».

Чтобы этого не произошло, в `TV_FLAG (23612)` нужно включить тумблер №5 для подавления курсора и пустить Стрелочку в обход вывода сообщений сразу в конечную точку – режим ожидания редактора `WAIT_KEY (5588)`. Проще всего это сделать через адрес 4764, где стоит готовая команда включения бита.

В виртуальный магнитофон перетащите файл «*Bytes с нижними строками.tzx*». Для загрузки блока кодов с началом в 16384, введите следующий алгоритм:

```
Debugger
Dec
23400 ← 3 255 0 0 0 0 0 0 0 0 0 0 0 0 0 0
23613 ← 65364
65364 ← 4764

HL ← 0
IX ← 23400
SP ← 65364
PC ← 1889
Trace
Cassette Recorder [Play]
```

После ввода заиграла задорная мелодия, под которую на экране стала появляться картинка:

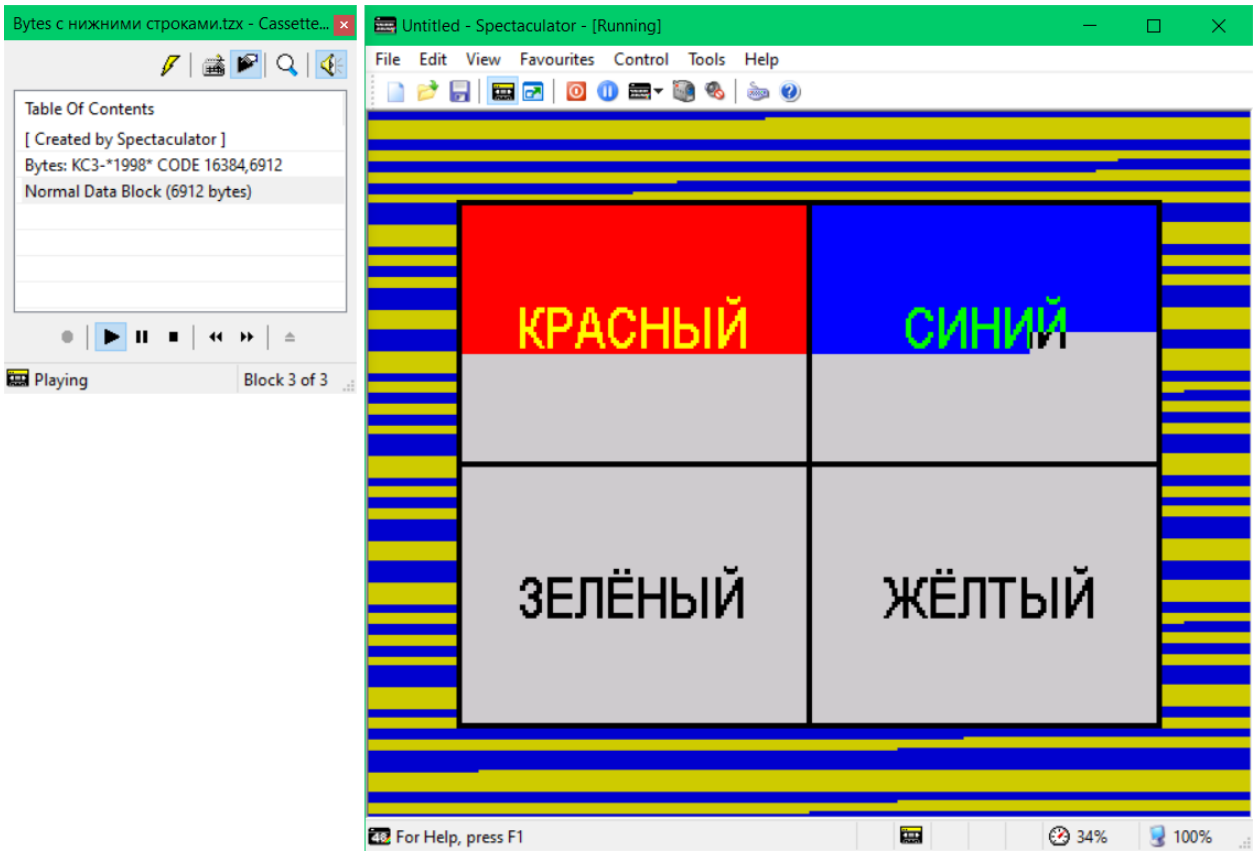


Рис. 369.

Я подозреваю, что гоблины точно не хоронили рыцарей под такую жизнерадостную мелодию… а может рыцари гоблинов. Да хер их там разберёт в этой гонке на лафетах.

После загрузки компьютер вошел в режим ожидания нажатой клавиши.

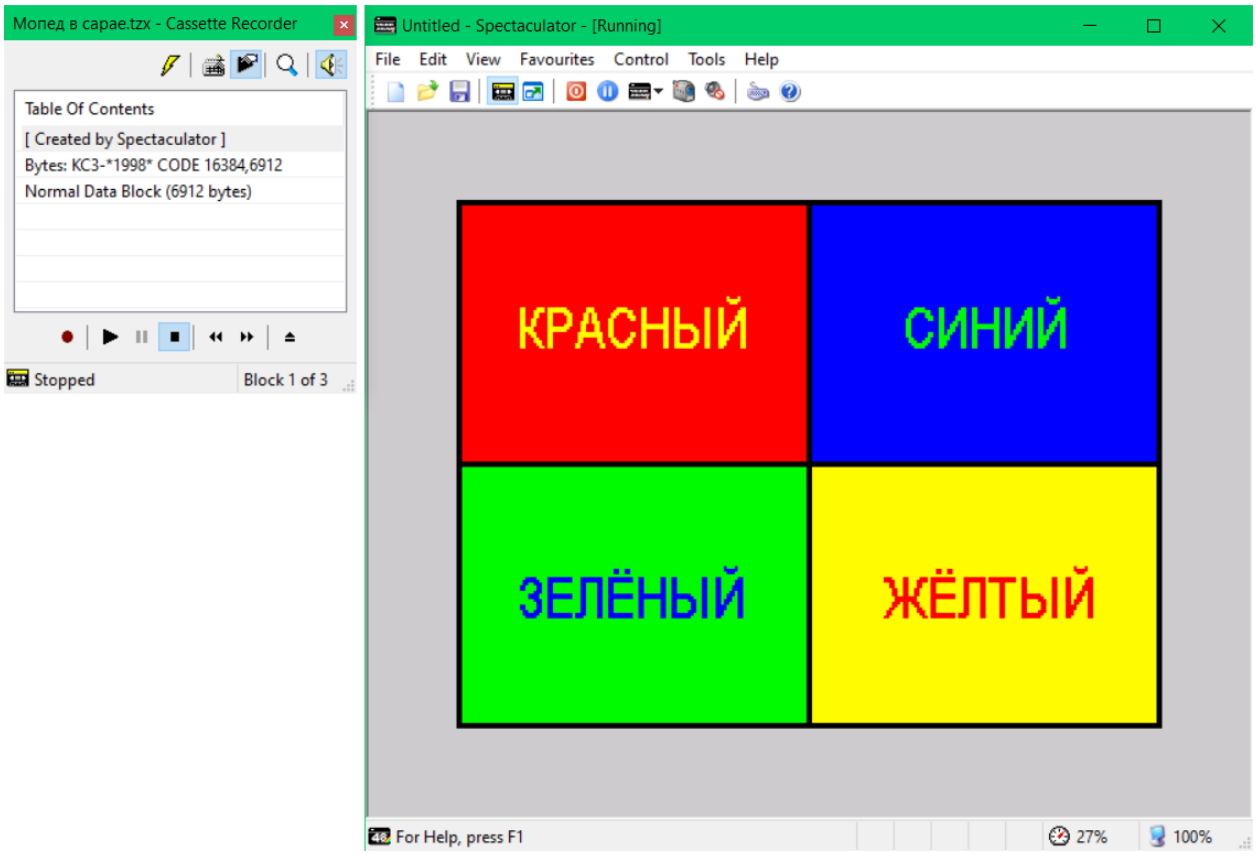


Рис. 370.

Нажмите ENTER. Экран очистится и в углу замигает курсор K.

Глава 40
Загрузка любого блока данных

Краткое содержание: синтез загрузки любого блока

Поскольку заголовок, это частный случай блока данных, длиной 17 байт, то нет никакого смысла загружать его отдельно. Как вы могли убедиться, заголовок и блок данных загружается одной и той же программой, просто следом за заголовком расположен вывод на экран считанных данных. Исходя из этого, предлагаю сразу рассмотреть классический приём загрузки без заголовка, который считает любой попавшийся блок данных.

Перетащите в магнитофон файл «*В Комарово.tzx*» и выберите пункт «*Normal Data Block (50 Bytes)*»:

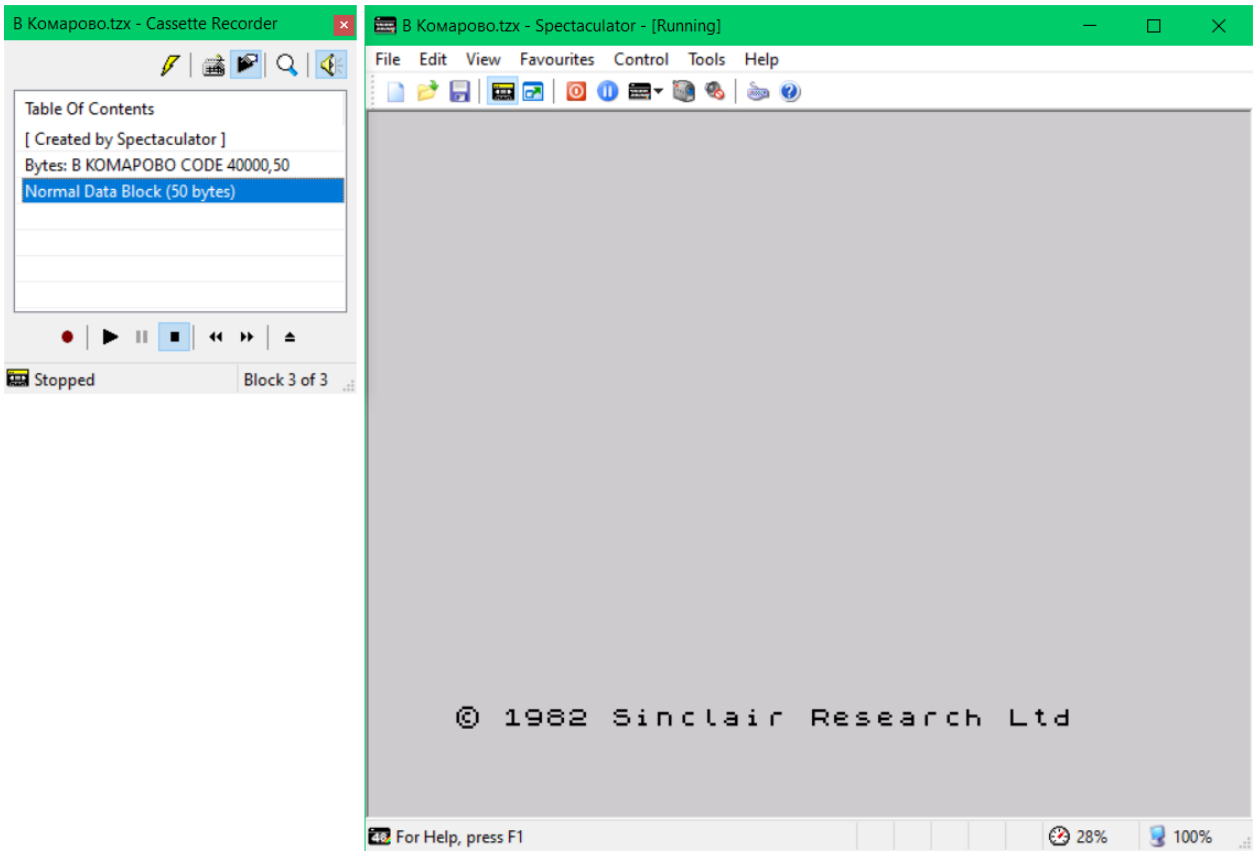


Рис. 371.

Следом введите алгоритм:

```
Debugger
Dec
23610 ← 255
23613 ← 65364
65364 ← 4867

AF ← 65280+1
DE ← 50
IX ← 40000
SP ← 65364
PC ← 1366
Trace
Cassette Recorder [Play]
```

После ввода начал загружаться блок данных без заголовка:

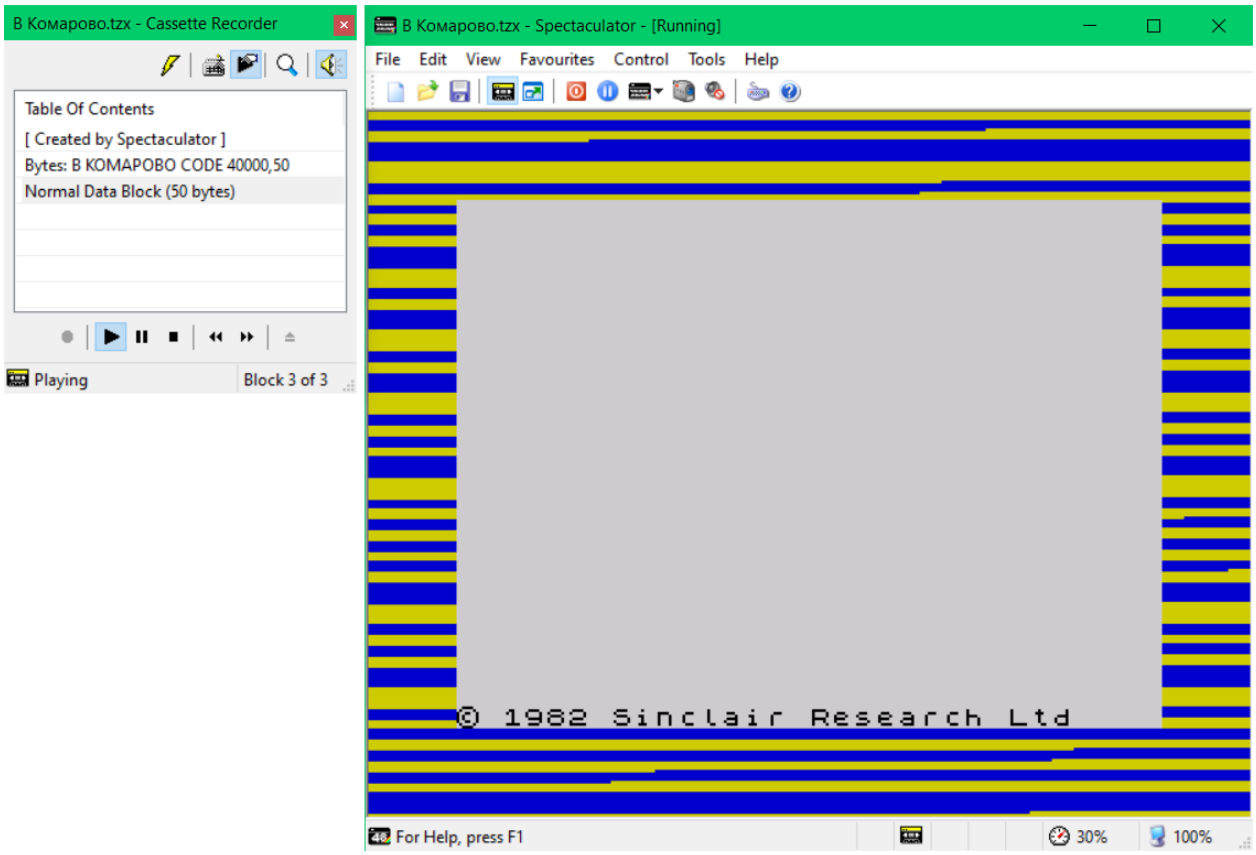


Рис. 372.

Для проверки введите `RANDOMIZE USR 40000`:

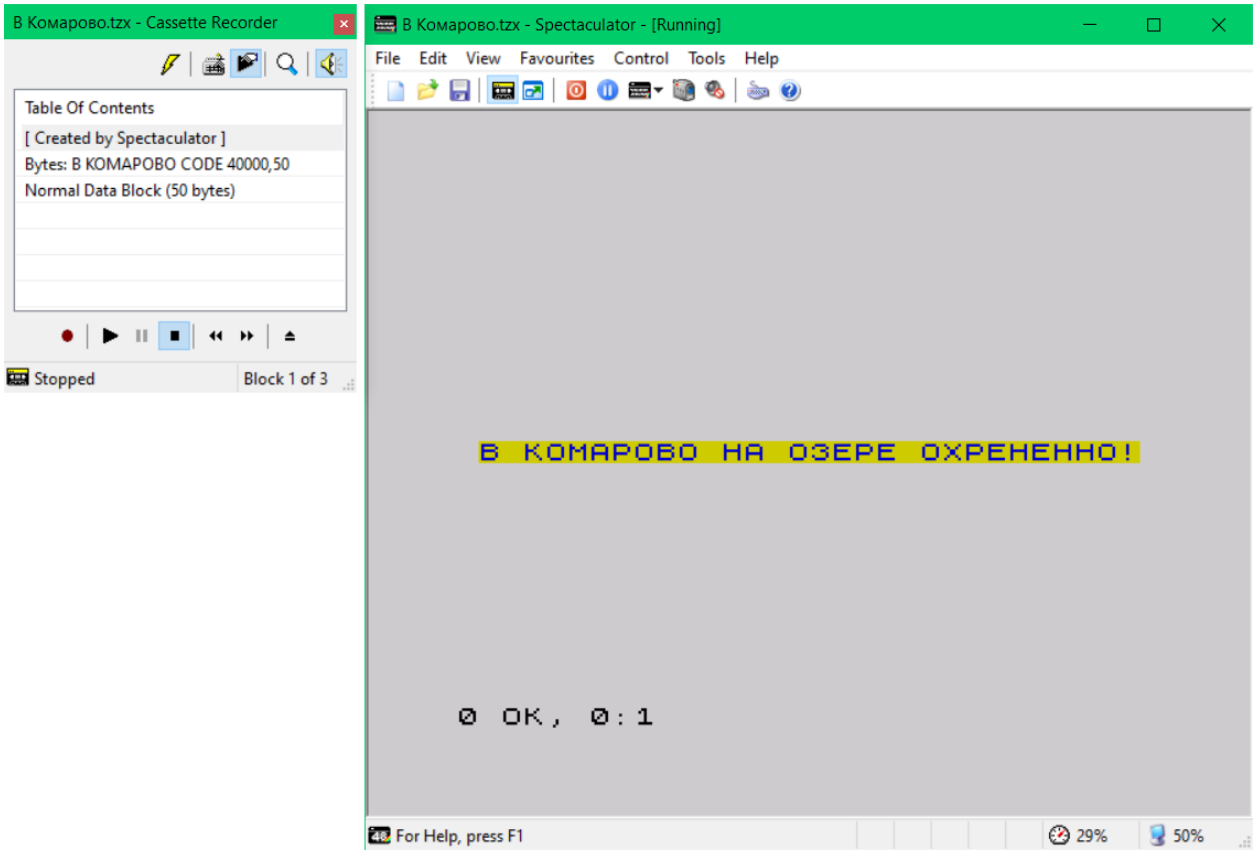


Рис. 373.

На экран выдался загруженный текст.

Для загрузки блока данных с автоматическим запуском, можно выполнить такую программу:

```
Debugger
Dec
23610 ← 255
23613 ← 65364
65362 ← 40000 4867

AF ← 65280+1
DE ← 50
IX ← 40000
SP ← 65362
PC ← 1366
Trace
Cassette Recorder [Play]
```

Результат будет идентичный. На этом моменте хочу закончить раздел. Со следующей главы начнётся кульминационная часть книги. Очистите магнитофон кнопочкой «*Eject*» и закройте *Spectaculator*.

Глава 41
История и классификация типов автозагрузок

Картинки и блоки данных, это конечно, прекрасно, но все нормальные программы после загрузки должны сами запускаться. С замиранием сердца, я подхожу к самой желанной главе. Именно в отчаянных попытках найти секрет автозагрузки «с 23755», спонтанно родилась эта книга, и я немного разобрался в структуре ПЗУ. Нетрудно догадаться, если вы читаете эту главу, значит, всё удалось. Поскольку эта больная тема у меня тянулась больше 31 года, предлагаю начать с истории.

Попытки разобраться в вопросах автозагрузки я предпринимал ещё с начала 1993 года. В один из зимних дней я рано вернулся из школы. Мама была на работе, папа ушёл на смену, а бабушка понесла кошку в ветеринарную клинику на операцию. В квартире воцарилась тихая и спокойная атмосфера.

Волнуясь за кошку, я врубил «Дубну-48» и начал исследовать очередную кассету с играми, которую недавно купил отец. Одна из успешно загружающихся игр называлась «Strip Poker», в которой во время игры раздевалась женщина.

Я начал грузить игру и по звукам услышал, что большая её часть создана на BASIC. В то время я не знал машинных кодов, и еще плохо программировал на BASIC, но часто работая с магнитофоном, я уже отличал по тональности и мелодии, какие данные загружаются. И тут я слышу и каким-то двести пятьдесят шестым чувством понимаю, что грузится сочный огромный блок BASIC, вскрыв который, можно поучиться секретам программирования. Но не тут-то было. При нажатии клавиш, загрузка прекращалась и всё сбрасывалось. Секретная команда MERGE "" тоже не помогала, потому, что блок «Program:» при загрузке таким способом перекашивало. Перепробовав все известные, на тот момент способы, я загоревал. Игра никак не поддавалась.

Проходило время. Вскоре мне попалась другая игра, в звуках которой, я заподозрил BASIC. Она называлась «Dominoes». Её дизайн был в желто-зеленых тонах. Каково было моё удивление, когда она вскрылась, и там обнаружилось 45 экранов отборного BASIC, но при этом, она записывалась «Бутэсом» (*блоком «Bytes» прим. авт.*). Потирая руки, я набрал свою программу и попытался записать её таким же блоком «Bytes:». Не зная устройства BASIC системы, я подбирал значения SAVE ""CODE 23xxx и так и сяк, но в итоге после загрузки у меня всё время портилась верхняя строка. В попытках её исправить компьютеру становилось только хуже. Вскоре по экрану начинали бегать знаки вопроса, потом грозно рычало, зависало или сбрасывалось. Что мне там помешало подобрать адрес,

я уже не помню. Возможно, меня не устраивала запись блока с области картинки или не нравилось оставлять в конце зазор в сотню байт, а подбирать точное количество наугад не хватило терпения. Это же после каждого байта нужно насиловать магнитофон, записывая программу на ленту, а потом проверять. Ну и самое главное, что такая программа отказывалась самозапускаться. Ведь любое условное «Рандомизе» в блоке «Program:» затиралось накладывающейся программой, а так хотелось, чтобы после загрузки «Bytes:» не повреждалась BASIC программа предыдущего блока. Отчаявшись как-то адекватно записать программу «Dice» из книжного самоучителя, настоящим блоком «Bytes:», я на долгое время оставил эту затею.

Проходили годы, складываясь в десятилетия. На смену Спектруму пришли эмуляторы и открыли второе дыхание. Я поверхностно научился программированию в машинных кодах и даже разобрался, как сделать свои уровни к паре программ из детства. Воодушевлённый этим небольшим опытом, в 2013 году я снова попытался разобраться в вопросе автозагрузки и сделать «программу мечты». В процессе исследований удалось открыть кое-что новое для себя, я написал простенькую книгу «*ZX-Spectrum-48K в эпоху windows и эмуляторов*» и успокоился. Еще немного проигравшись в ассемблер, я и вовсе позабыл про эмуляторы. А тем временем грянули апокалипсические 2020-е годы со всеми вытекающими…

Настала осень 2024 года. С лета этого года, с головой закопавшись в изучение ПЗУ, я предпринял очередную попытку в поисках алгоритма программы мечты и вот оно свершилось.

В эфире снова рубрика «Возвращаясь к напечатанному». В качестве предисловия к будущим автостартам, придётся опять возвратиться к одному из разделов своей «напечатанной» книги 2013 года, которую даже опубликовали на сайте «Virtual TR-DOS» (приятно, блин).

Предлагаю посмотреть на предпринимаемые попытки, и, проанализировав ошибки прошлого понять, почему в итоге тема так и не раскрылась до конца:

очень пригодиться. Вычисляем черновую длину 23850-23613=363 байта.

Но это еще не все, следует учесть, что после нажатия ENTER и ввода строки, пойдут процессы, которые отследить до записи не удастся. Программа расползется на какое-то количество байт, потому что выполнятся операторы в этой строке, например тот же RUN. При этом расширяется область переменных **VARS**, которая вклинивается в область между программой на бейсике и вводимой строкой. Поэтому нужно взять длину с небольшим запасом. Дополнительная длина может быть разной, в зависимости от стоящих операторов, через двоеточие после SAVE. Чем их больше, там дальше сползет конец программы вниз.

Если в строке после `SAVE "TEST-BASIC"CODE 23613,400`, через двоеточие стоит только RUN, то «индекс сползания» будет 12 байт, а если длина иная, то нужно вычислить индекс, и прибавить к его основной длине.

Давайте разберемся на простом примере, как это происходит. Сохраните программу текущим слепком памяти, в формате **.z80*, например, под именем «*test-basic.z80*». Для этого в эмуляторе откройте вкладку «*File → Save as…*» В строке «*Save as type*» выберите «*Z80 Snapshot (.z80)*»:

Введите строку своей программы на Spectrum. Появится надпись «`Start tape, then press any key`». Снова нажмите ENTER, и терпеливо подождите, пока выполнится вся загрузка. Затем еще раз нажмите любую клавишу для `PAUSE 0`, и на фоне желтой рамки выскочит сообщение `0 OK, 4:1`.

Теперь после прогона «сценария развития событий» сразу откройте *Debugger*, и посмотрите новое значение длины.

Обратите внимание, что переменная **WORKSP** теперь уже нам в этом не помощник. После выполнения строки в нижней части экрана, компьютер про нее забыл и передвинул все сзади стоящие области на пару десятков байт выше. Строка стала отработанным мусором, и поэтому при следующей операции начнет затираться, по мере надобности. В этом случае следует искать адрес только вкучную. Всегда стоит внимательным образом искать конец действующей строки, и не перепутать с остаточным мусором от предыдущих операций (если таковые были).

Как видите, данные после выполнения сползли на адрес 23862, то есть на 12 байт ниже. Вот это и есть тот самый «индекс сползания программы».

Рис. 374. Фрагменты из книги «ZX-Spectrum-48K в эпоху windows и эмуляторов».

Смещение от команды RUN… процессы до записи отследить не удастся… искать вручную конец программы…

Бля, я вот только сейчас от смеха вылез из под стола. Оказывается, мои мемы будут покруче, чем гоблины с рыцарями вместе взятые.

Если что, мне не слабо поприкалываться над собственными ошибками прошлого. Я не делаю исключение никому. Своё творчество я критикую точно так же, как и чужое. Ну а с другой стороны не ошибается тот, кто ничего не делает, поэтому критику стараюсь делать объективной и с юмором.

Не осилив, как функционирует BASIC с системными переменными, таким образом, я пытался разобраться в процессах 11 лет назад. О принципах работы SP-области и ПЗУ, которого я вообще не понимал, я скромно умолчу. Почитав ту книгу сейчас, я понял, что не знал об автозагрузке ничего. Лишь методом тыка с бестолковым захватом большей части области системных переменных, я подобрал количество байт, чтобы программа таки запустилась, но объяснить происходящее не смог. Да и не с 23755 загружалась программа в той главе, а с 23613. Таким образом, попытку-2013 считаю лишь частично удавшейся.

И ладно, в 1993 году я вообще не имел представления о машинных кодах и как устроен BASIC. Но ведь в 2013 году я даже написал пару простых редакторов на

ассемблере. Почему тогда не получилось вникнуть в ПЗУ и разобраться, как формируется «SP»? Да хрен его знает, я уже позабыл.

К стыду своему, про автозапуск в «SP» тоже идея была не моя. Я вычитал из пиратской книжки с нерабочим примером программы. Что-то подшаманив наугад, мне удалось нащупать автозапуск через SP-область, наворотив кучу лишнего хлама. Сколько смотрю в ту книгу, столько за голову хватаюсь, какую ахинею я там прогнал по поводу «Категорий» автозапуска. Вижу, что назрела категория «Ретро» как в последних номерах «ZX-Ревю», перед его исчезновением.

На самом деле, для вычисления точного количества байт в строке не обязательно так сложно «прогонять» сценарий, как это делалось в прошлой книге. Теперь вы знаете, что все эти «разбухания» происходят после возвращения с проверки синтаксиса (6935) на адрес 4791, где кроме всего прочего распаковываются числа. И происходит это из-за числовых значений в команде `SAVE`. Команда `RUN` тут вообще не при делах. Сейчас вы сами можете вычислить такого рода смещения, но я предлагаю и вовсе отказаться от записи блоков методом BASIC. Пусть это останется в прошлом поколении книжек.

А теперь к делу. Предлагаю посмотреть на загрузку данных не только свежим взглядом, но и под другим углом. В процессе освоения этой книги, вы много путешествовали со Стрелочкой по BASIC-блокам, генерировали и запускали программные строки без помощи команд. К этому моменту книги вы знаете о ПЗУ и «SP-столбике» в разы больше, чем я в 2013 году, когда написал «*ZX-Spectrum-48K в эпоху windows и эмуляторов*».

Помните цветные ♥ сердечки, которые попадались в главе «*Путешествие с командой LOAD*»? Для начала, предлагаю вернуться в ту главу и собрать все сердечки, попадающиеся по пути, в процессе загрузки данных. Затем взять ниточку, созданную из дорожки памяти с адреса 23755 по 65535, каждую цветную бусинку нанизать на эту ниточку, и добавить немного дизайнерских элементов:

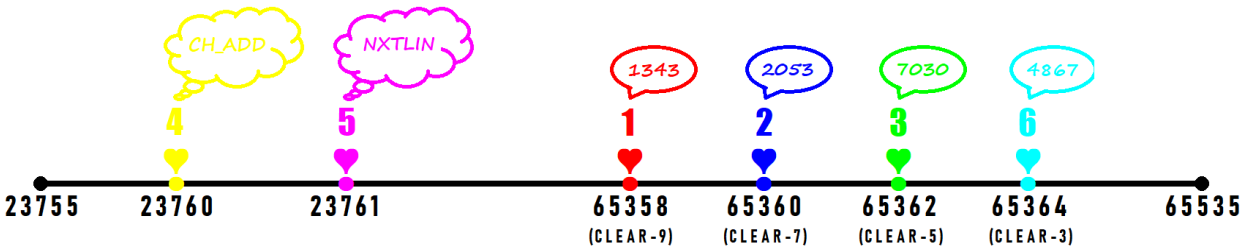


Рис. 375. Схема точек перехвата внесистемной автозагрузки и последовательность прохода по ним.

Такие получились бусы или станции метро. Вот эти все шесть точек, в которых вы можете перехватить управление и скорректировать адрес автозапуска без модификации ПЗУ. Именно в такой последовательности Стрелочка возвращается из программы считывания в свой «Дворец». Поскольку такой расклад справедлив для блоков загрузки с адреса 23755, то эти точки автостарта предлагаю назвать «Внесистемными». Под этим словом я подразумеваю, что блок данных, загружаемый в память, напрямую не модифицирует область системных переменных (23552-23754). В этом случае приходится подстраиваться под существующий расклад, сформировавшийся после сброса.

Естественно, блоки данных, загружаемые с адреса 23552 и ранее, будут иметь еще несколько дополнительных точек для возможностей перехвата, но все они будут расположены в системных переменных. Такие автостарты будут называться «Системными», но всё по порядку. О них расскажу позже.

Глава 42
Автостартующий блок «Ранний»

Краткое содержание: SA/LD-RET, точка 65358

Итак, самая первая красная точка перехвата после удачного считывания программы будет находиться в ячейках с адресами 65358/59 на голом компьютере. Этим числом является 1343 или начало подпрограммы SA/LD-RET.

Если до записи вы немного побаловались командой CLEAR и сместили SP-столбик (на фундамент столбика указывает RAMTOP (23730)), то этой красной точкой станет адрес CLEAR X-9, где X – значение, которое вы ввели с командой.

Надо с чего-то начинать. Как говорилось в рекламе страхового брокера «Энерджи Лайф», ТРАХНЕМ СТРАХ:

Рис. 376. Реклама «ТРАХНЕМ СТРАХ». Фото из интернета.

А теперь можно приступить к созданию первого или «Раннего» (*о, новый термин родился!*) автостартующего блока «Bytes: », но придётся учесть несколько особенностей. При таком раннем перехвате теряется отчёт о правильности считывания (галочка «С») и команда EI, которая стоит далее. Без нее отключится опрос клавиатуры по адресу 00056 (который фурычит благодаря IM 1).

Вариантов любых программ автозапуска в пределах одной разновидности почти всегда будет три:

– с окончанием в точке перехвата
– с проходом сквозь точку перехвата
– с началом в точке перехвата.

Естественно, самым сложным будет второй вариант, поэтому его и возьму в качестве первого примера. Кроме того, предлагаю автоматизировать процесс записи последующих блоков, чтобы упростить себе жизнь. Для этого нужно изменить алгоритм SAVE, так, чтобы после записи автоматически происходил сброс, путём перехода по адресу 0. Таким образом, после перезагрузки вы сразу сможете набрать LOAD ""CODE и проверить записанный блок «без пара и воза».

Наберите и введите следующий алгоритм записи автостартующего блока «Ранний»:

```
New File [Автозапуск 1-2.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 65220
65220 ← 253 54 0 255 49 84 255 237 115 61 92 241
65232 ← 251 253 203 2 134 17 225 254 1 39 0
65243 ← 205 60 32 195 3 19 22 11 0 18 1 16 2
65256 ←ТРАХНЕМ СТРАХ
65269 ← 32 80 65 72 72 144 77 32 65 66 84 79
```

```
65281 ←СТАРТОМ
65318 ← 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
65338 ← 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
65358 ← 65220 0 0 0
65366 ← 0 62 0 66 70 74 82 98 66 0

      ; запись блока с автосбросом

Go To 17996
17996 ← 65220 0              ; стартовый адрес для блока данных
18000 ← 3                    ; тип блока Bytes
18001 ← TPAX-CTPAX           ; имя заголовка
18011 ← 156 65220            ; длина и стартовый адрес для заголовка

IX ← 18000                   ; место размещения заголовка
SP ← 17996       ; установить SP в свободное место для автосброса
PC ← 2436                    ; переход на подпрограмму записи
Trace
Cassette Recorder [Stop]
```

После нажатия «*Trace*», по экрану пробежит пара чёрточек, как при загрузке «программы с помойкой». На экране отображается работа «SP», который для записи блока был временно перенесен в область экрана:

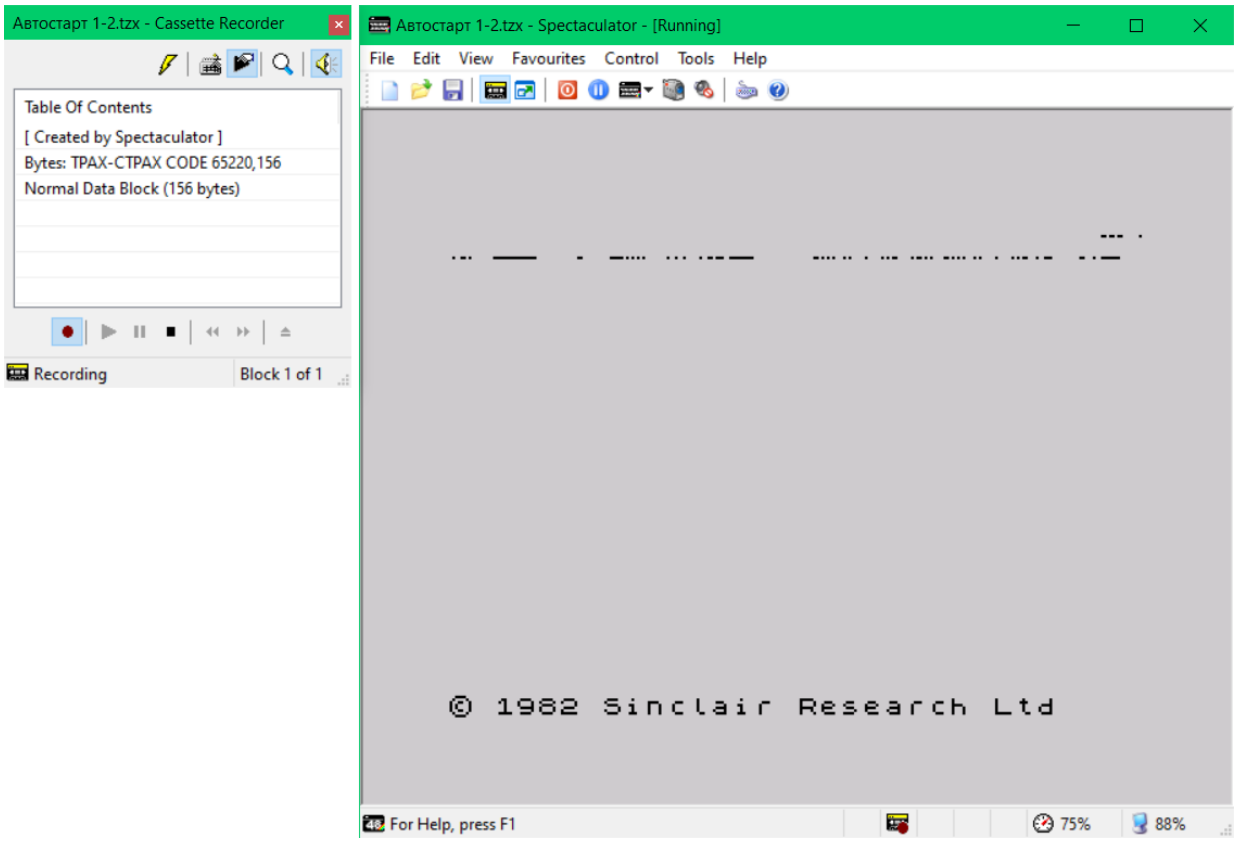


Рис. 377. Процесс записи блока «Bytes:».

Следом произойдёт сброс, а в магнитофоне появится автостартующий блок «`Bytes:`». Выключите запись магнитофона «*Stop Tape*». Лосьон «Огуречный», тьфу, блок автозагрузки «Ранний» готов. Обратите внимание, никакой мутотни с BASIC. Ведь для загрузки программы в эту область нужно сначала набрать `CLEAR`, потом долбиться с командой `SAVE`… А тут создали программу, она автоматом записалась и компьютер вернулся в исходное положение.

Наберите `LOAD ""CODE`:

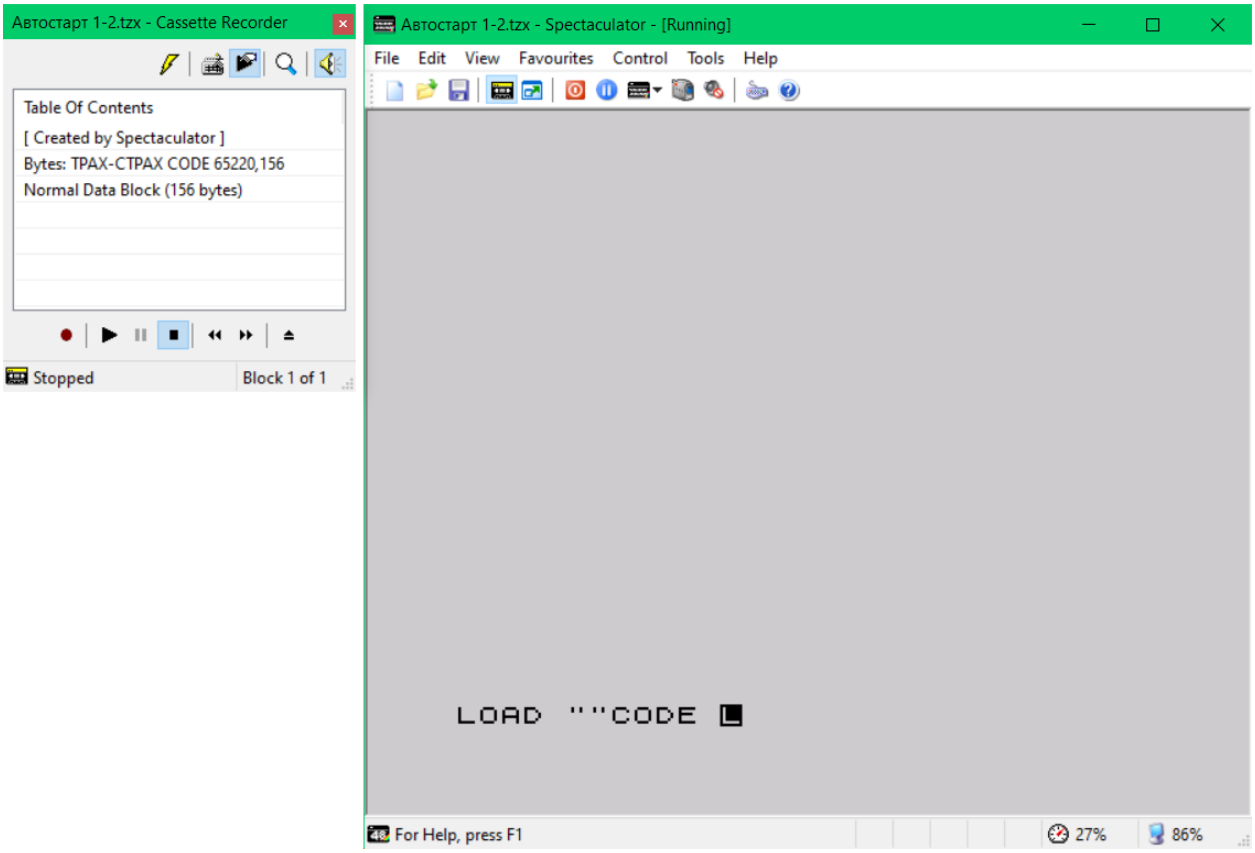


Рис. 378. Подготовка к загрузке сгенерированного автостартующего блока типа «Бутес».

Вводите и включайте «*Play*» магнитофона. После загрузки на экран выдастся тест и сообщение «0 OK»:

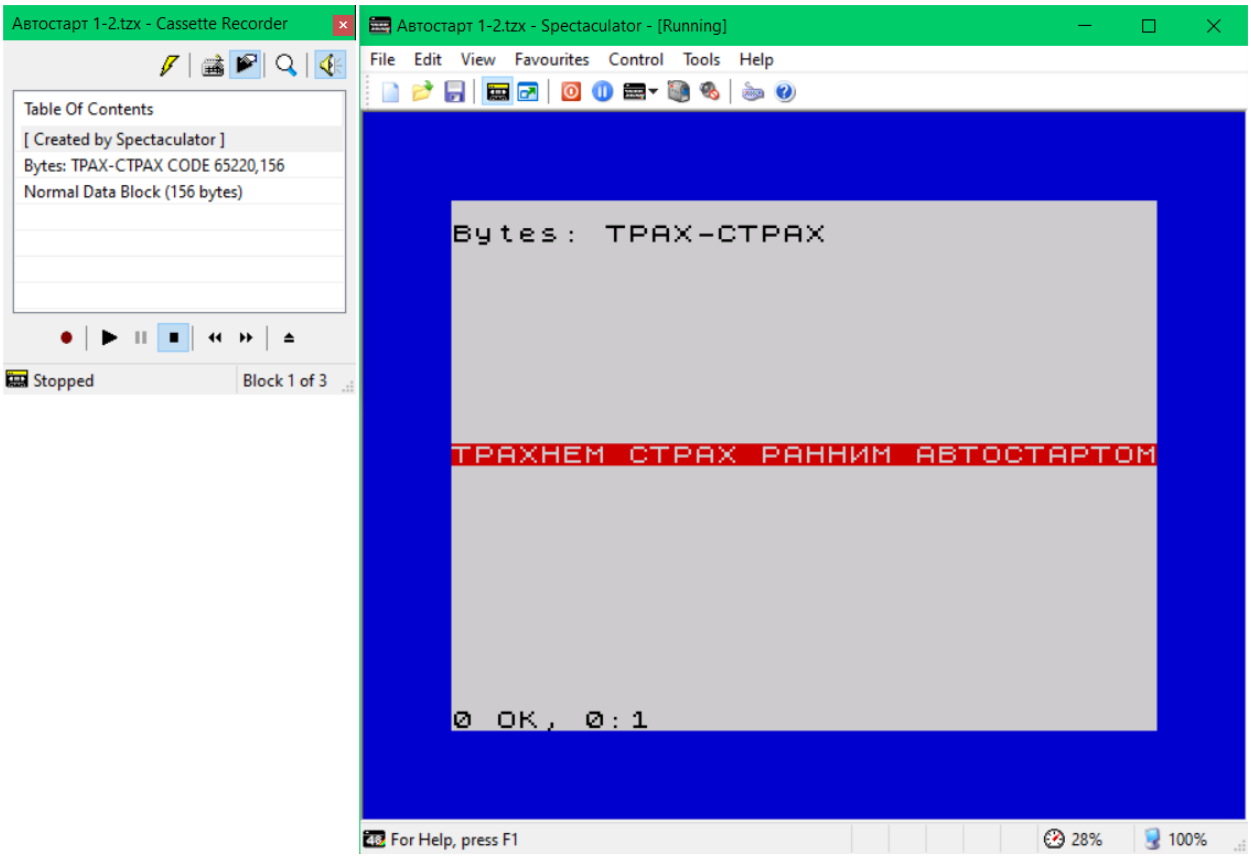


Рис. 379.

Синяя рамка осталась от последней полоски загрузки, поскольку пропустили подпрограмму, которая полностью восстанавливает экран, ограничившись установкой базовых настроек. По желанию вы можете сами ее добавить. Я её оставил, потому что она вписалась в цветовую гамму спародированной скандальной рекламы.

Нажмите ENTER и экран очистится. Вы можете повторно загрузить программу, и она снова корректно запустится, вернув жизненно важные настройки в первоначальное положение.

Вариант автозагрузки блока «Ранний» с окончанием в точке перехвата будет чуть проще:

```
New File [Автозапуск 1-1.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 65220
65220 ← 253 54 0 255 49 84 255 237 115 61 92 241
65232 ← 251 253 203 2 134 17 225 254 1 34 0
65243 ← 205 60 32 195 3 19 22 11 2 18 1 16 2
65256 ←ТРАХНЕМ СТРАХ АВТОСТАРТОМ 1
65318 ← 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
65338 ← 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
65358 ← 65220

Go To 17996
17996 ← 65220 0
18000 ← 3
18001 ← TPAX-CTPAX
18011 ← 156 65220

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

После набора и ввода программы, произошёл плановый «добрый сброс» (словосочетание то какое). Следом по команде LOAD ""CODE [ENTER] загрузилась и автостартовала программа:

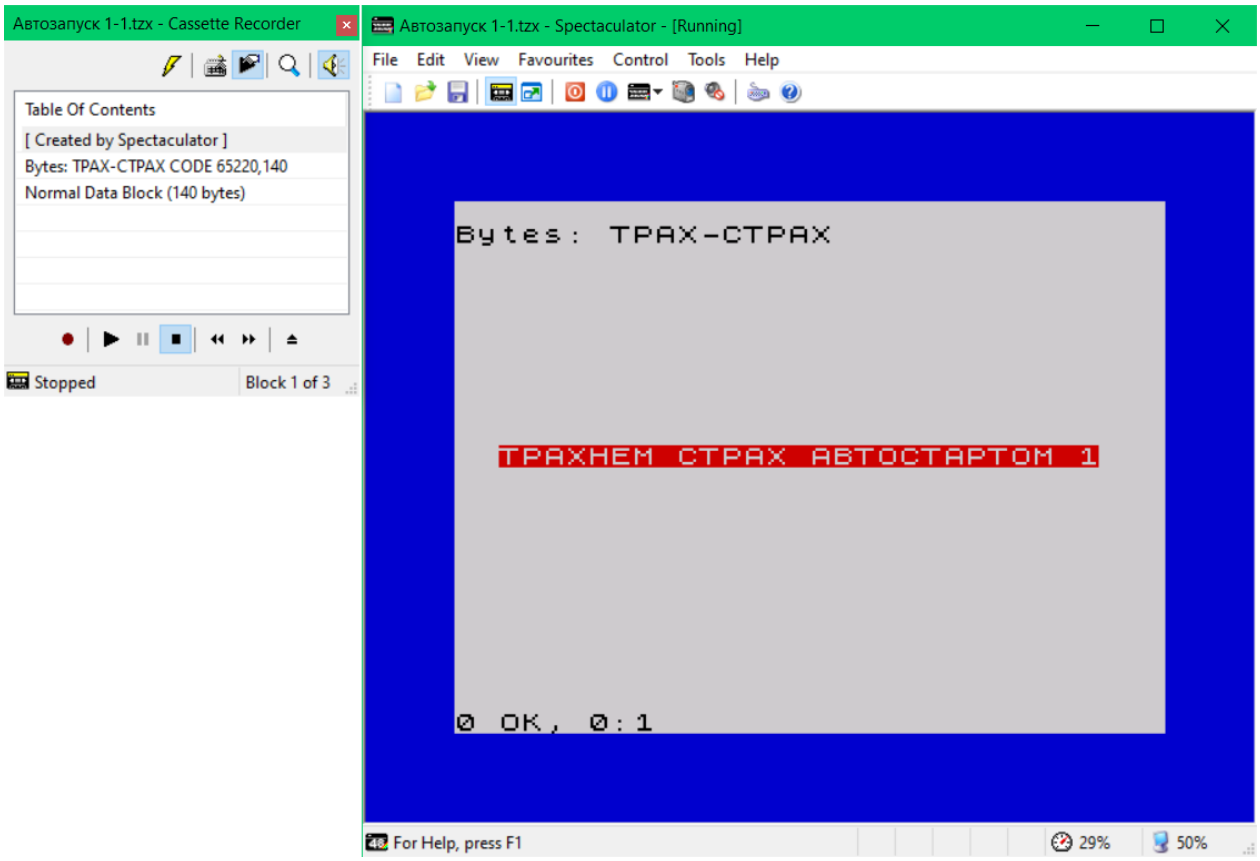


Рис. 380. Вывод текста после загрузки автостартующего блока «Bytes:».

С этого момента я уже не буду писать включить или остановить магнитофон, а выводить картинку с конечным результатом для сверки. Все эти действия уже включены в алгоритм программы. Если что-то непонятно, всегда можете нажать поиск по тексту и вернуться к первым главам, где это всё подробно объясняется. Мало того, в конечной редакции, все эти примеры я точно также вместе с вами копирую из этой книги, компилирую, проверяю работоспособность и кладу в папку, которая должна будет войти в комплект.

Возвращаюсь к программе. Этот вариант автозапуска без UDG области, поэтому пришлось изменить текст, так как не задать дополнительные символы с кириллицей.

Заключительный вариант этого автостарта с началом в точке перехвата 65358. Основной недостаток заключается в том, что для компактности программу нужно размещать в UDG. Нет, конечно, можно сделать блок размером 65533 с переходом через ПЗУ, и разместить выполняемую программу в 40000, но для коротких программ такой подход не совсем рациональный.

Итак, вариант программы с началом в точке перехвата:

```
New File [Автозапуск 1-3.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 65358
65358 ← 65220 0 0 0
65366 ← 0 62
65368 ← 253 54 0 255 49 84 255 237 115 61 92 241
65380 ← 251 253 203 2 134 17 117 255 1 34 0
65391 ← 205 60 32 195 3 19 22 11 2 17 0 16 2
65404 ←MATPOC HA ABPOPE MOET OTCEK

Go To 17996
```

```
17996 ← 65358 0
18000 ← 3
18001 ← 17 0 16 2 65 66 80 79 80 65
18011 ← 73 65358

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

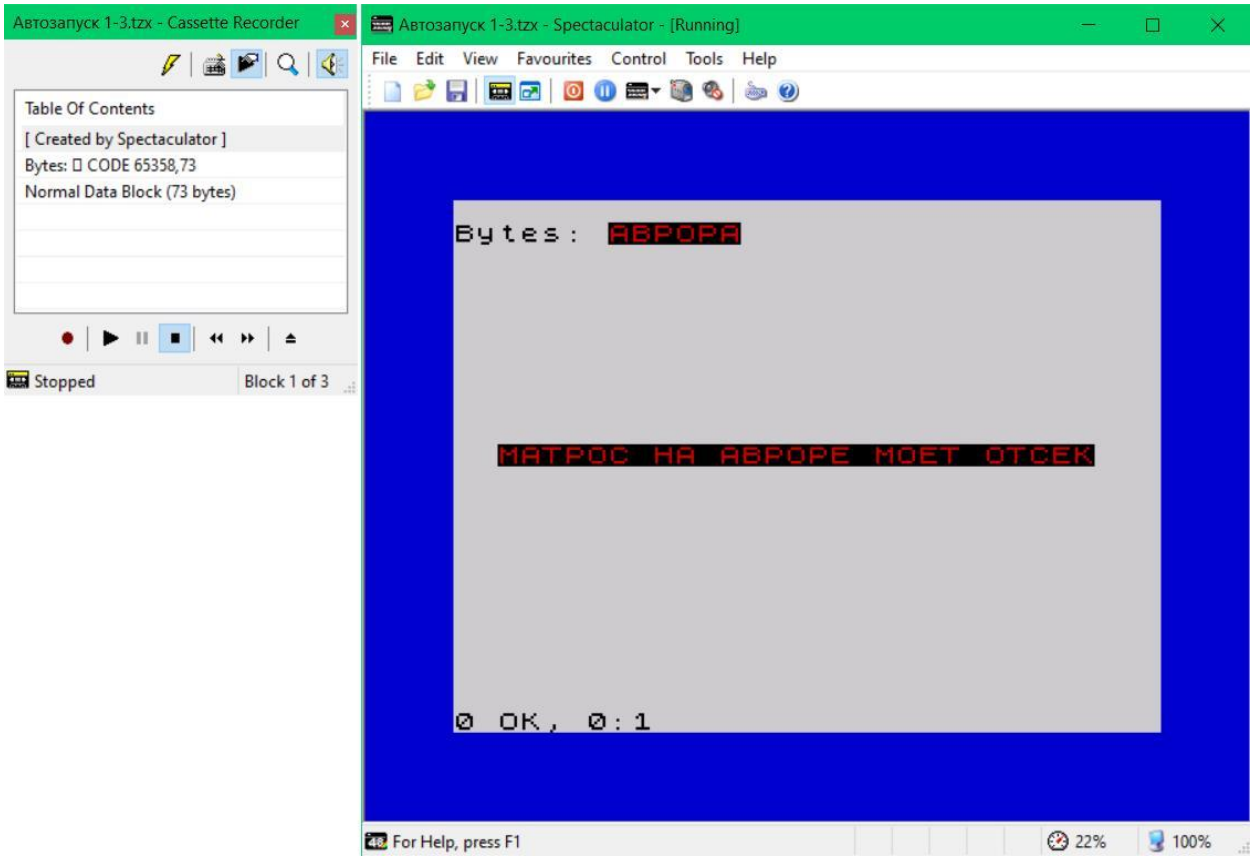


Рис. 381. Вывод текста после загрузки автостартующего блока «Bytes:».

Этот вариант получился самым компактным, как компактный алкоголь из одноимённого видео на моём YouTube-канале. Подборка цветов не совсем удачная, но как бы красный цвет исторический для крейсара Аврора.

Глава 43
Автостартующий блок «Безошибочный»

Краткое содержание: LD-BLOCK. точка 65360

Если с первой понятно, можно приступить ко второй точке по адресам 65360/61, которая отмечена синим цветом. В этой точке по команде `RET` из `SA/LD-RET (1365)` происходит возврат в `LD-BLOCK (2053)`. На этом месте уже активируются прерывания и рамка восстанавливает свой цвет. С этим местом проблем меньше, но добавляется необходимость сохранить адрес возврата предыдущей точки.

```
New File [Автозапуск 2-2.tzx]
Cassette Recorder [Record]
```

```
Debugger
Dec
Go To 65220
65220 ← 253 54 0 255 49 84 255 237 115 61 92 241
65232 ← 253 203 2 134 17 224 254 1 41 0
65242 ← 205 60 32 195 3 19 22 11 0 17 1 16 4
65255 ← 19 1 66 32 84 65 80 88 79 66 75 69 32 72
65269 ← 65 32 144 69 67 75 69 32 80 65 67 84 69
65282 ← 84 32 79 67 79 75 65
65318 ← 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
65338 ← 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
65358 ← 1343 65220
65362 ← 0 0 0 0 0 62
65368 ← 0 126 66 66 66 66 66 0

17996 ← 65220
18000 ← 3
18001 ← 16 1 84 65 80 88 79 66 75 65
18011 ← 156 65220

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

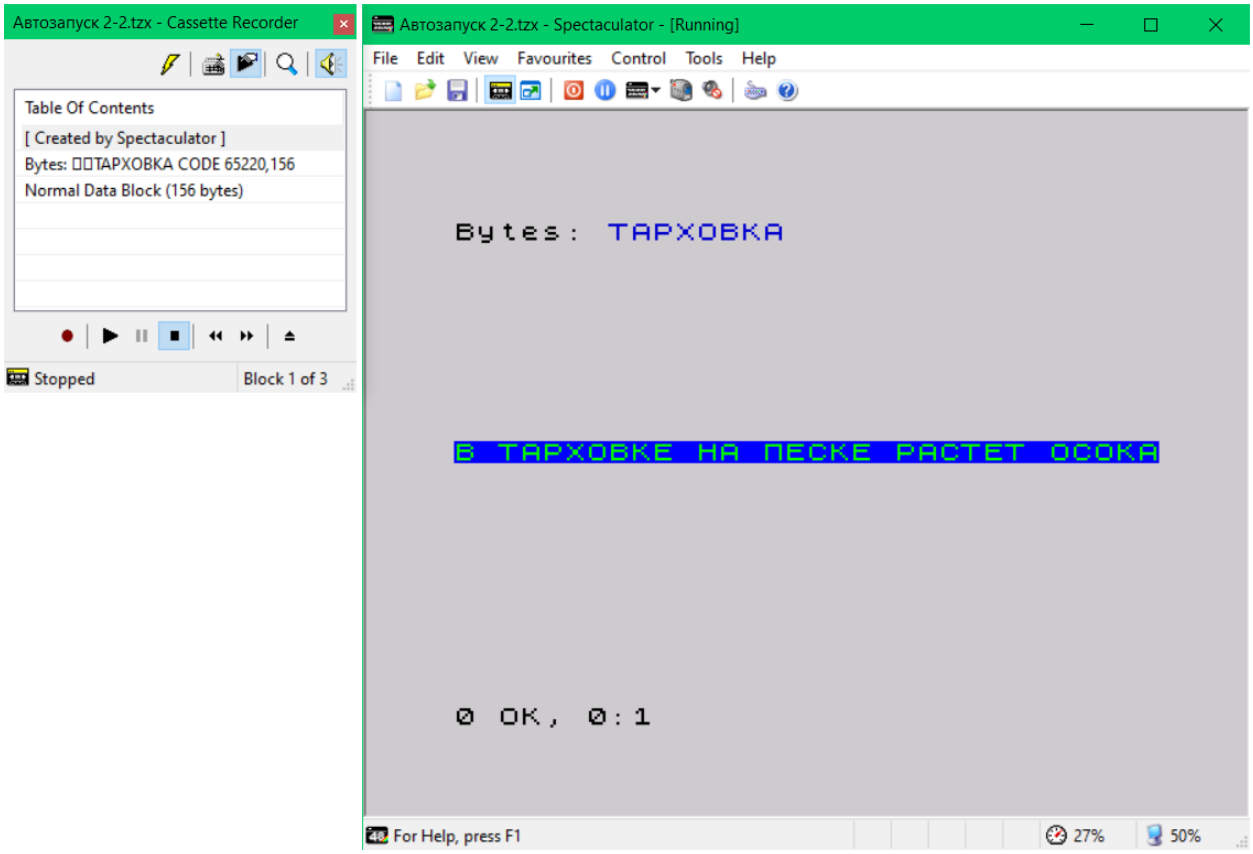


Рис. 382. Вывод текста после загрузки автостартующего блока «Bytes:».

…что ни говори, а побережье Финского залива, у платформы Тарховка в Сестрорецком районе, очень живописное. Шум волн и желтый песок, из которого пробивается бирюзово-серебристая осока…

Рис. 383. Серебристая осока на побережье Финского залива в Зеленогорске. Фото 13 июня 2021 года.

На этот раз и рамка восстановлена, но по-прежнему, потерян отчёт о правильности считывания данных. Именно поэтому к данному автостарту приклеилось прозвище

«Безошибочный», потому что на этой точке перехвата теряется отчёт об ошибке считывания. Звучит трояко, как предложение: «Ехал на велосипеде и разбил яйца О(А)браму». И непонятно какие именно яйца разбил чувак: свои, куриные из магазина или Абрама, которого вёз на багажнике. Не удивлюсь, если найдётся и четвёртая версия.

Вариант программы с окончанием в точке 65361 будет следующим:

```
New File [Автозапуск 2-1.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 65220
65220 ← 253 54 0 255 49 84 255 237 115 61 92 241
65232 ← 253 203 2 134 17 224 254 1 40 0
65242 ← 205 60 32 195 3 19 22 11 0
65251 ← 17 6 19 1 16 2
65257 ← В КОСТРОМЕ РАССВЕТ КАК В СКАЗКЕ
65318 ← 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
65338 ← 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
65358 ← 1343 65220

17996 ← 65220
18000 ← 3
18001 ← 16 2 75 79 67 84 80 79 77 65
18011 ← 142 65220

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

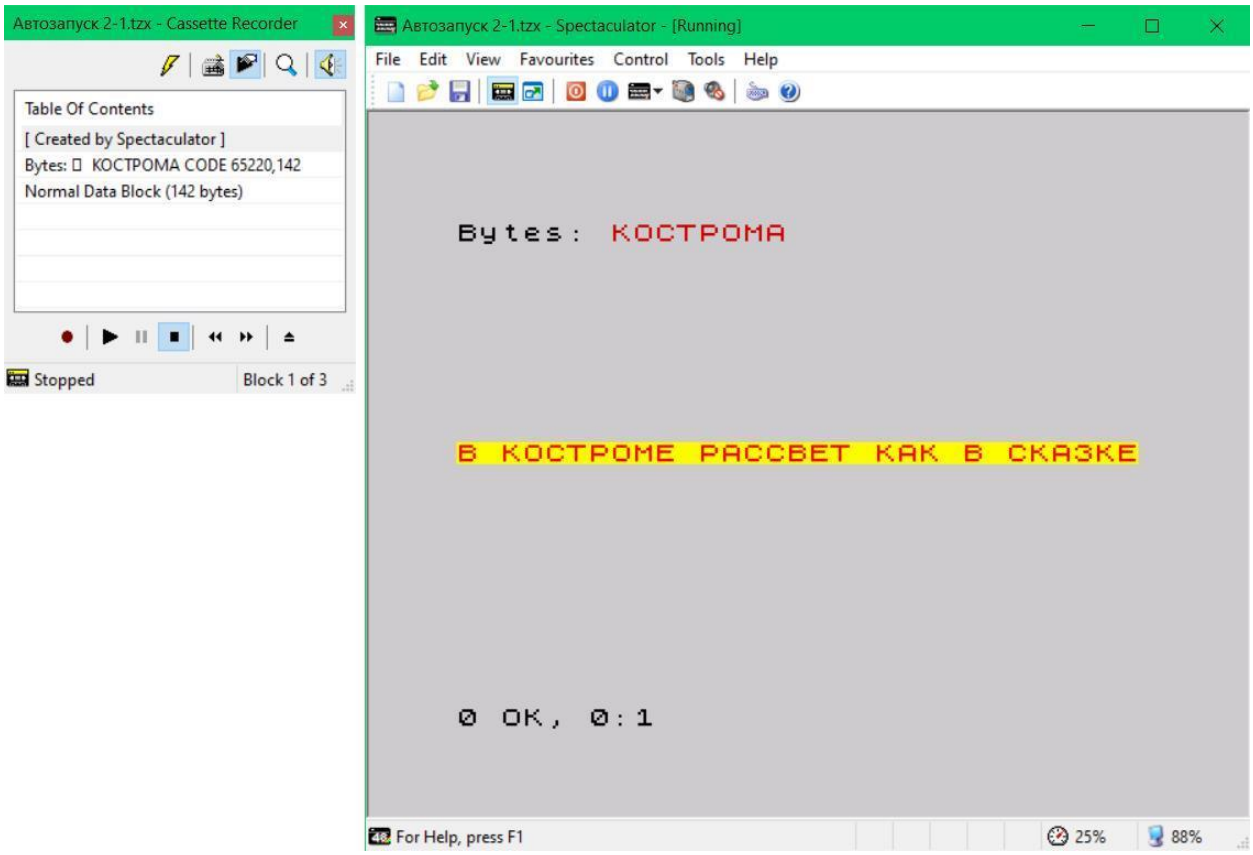


Рис. 384. Вывод текста после загрузки автостартующего блока «Bytes:».

И заключительный вариант с началом в точке 65360:

```
New File [Автозапуск 2-3.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 65360
65360 ← 65368 0 0
65366 ← 0 62
65368 ← 253 54 0 255 49 84 255 237 115 61 92 241
65380 ← 253 203 2 134 17 116 255 1 41 0
65390 ← 205 60 32 195 3 19 22 11 0
65399 ← 16 6 19 1 17 2
65405 ← В НЕВСКОМ ВОЕНКОМАТЕ СТРЕМНОВАТО

17996 ← 65360 0
18000 ← 3
18001 ← ВОЕНКОМАТ!
18011 ← 77 65360

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

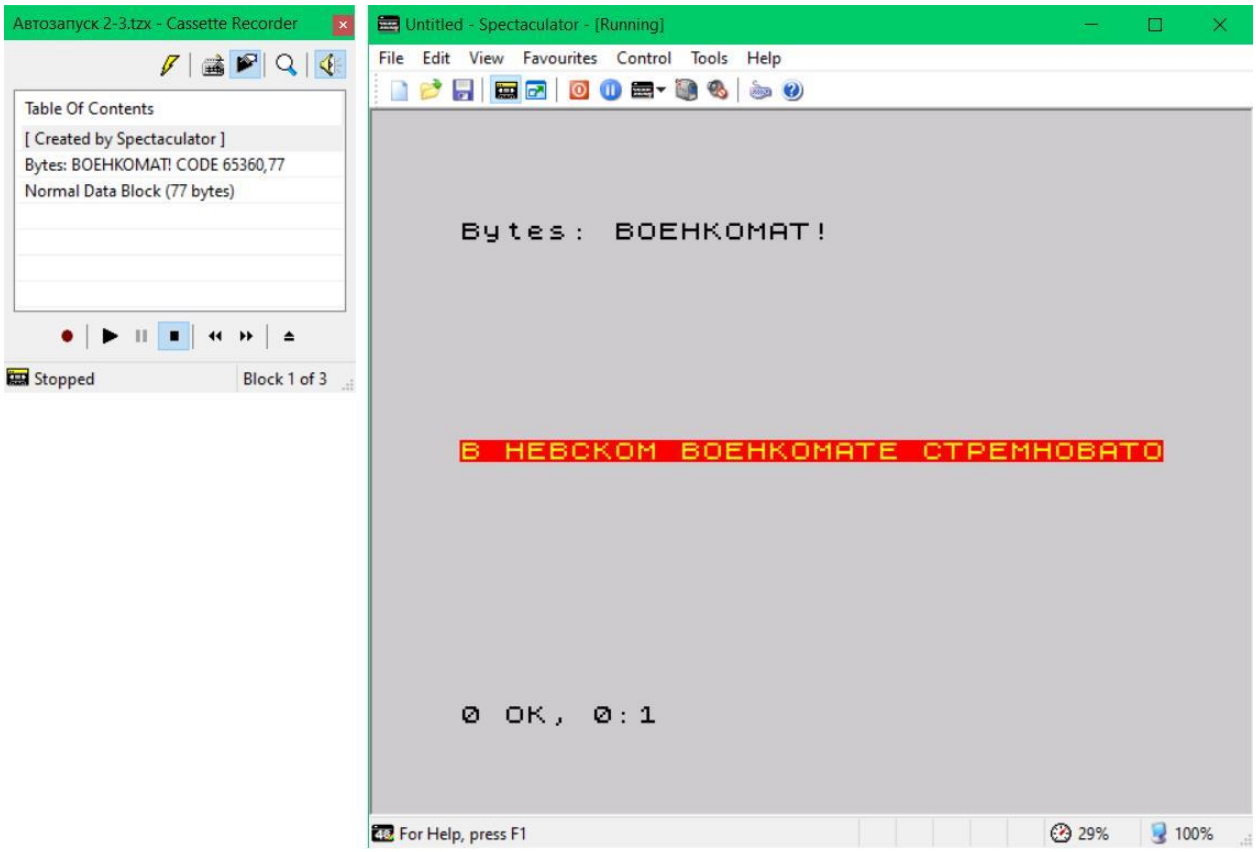


Рис. 385. Вывод текста после загрузки автостартующего блока «Bytes:».

Однажды жизнь заставила продать квартиру, естественно со всей вытекающей бюрократией с перепрописками. Настал момент, когда осталась только перепостановка на воинский учёт по новому адресу проживания. В один из утренних дней я приехал в Невский военкомат сниматься с учёта. Захожу в проходную, а там, на входе охрана, каждого мурыжит по десять минут, чуть ли не с металлоискателем. Запись по паспорту, куда к кому, цель визита и всё такое. Не хватает только анкеты, как заполняют туристы при пересечении границы. Да и само здание какое-то мрачное и стрёмное внутри. Кругом снуют люди, освещения мало…

После полутора часов стояния, я выписался и облегчённо вышел на улицу. Ну, ещё час мучений и дело сделано. Приезжаю в Купчинский военкомат на Южное шоссе. А тут атмосфера противоположная. Маленький уютный домик, светлые коридоры и пустота. Зашел на проходную, показываю паспорт и открываю рот, чтобы сказать цель визита, а тётка охранница машет рукой.

«Идите без записи, кто просто так в здравом уме сюда придёт» – пошутила она.

Как в другой мир попал, а это всего лишь соседний район. И настроение моё улучшилось.

Глава 44
Автостартующий блок «Чистый»

Краткое содержание: STMT-RET, точка 65362

Третья зелёная точка автозапуска происходит по команде `RET C` с адреса 2053 в `STMT-RET (7030)`. За этот переход ответственны адреса 65362/63. Это последняя точка, где можно перехватить программу перед возвращением в BASIC область. Для использования этого адреса в качестве автостарта, нужно просто сохранить два предыдущих: `SA/LD RET (1343)` и `LD-BLOCK (2053)`.

Так уж повелось с позапрошлой главы, автостартам даются имена. Этот можно назвать «Чистый», потому что по нему стартует только при условии полной загрузки программы. Соответственно и тематика текста чистая и природная.

Вариант с проходом сквозь «SP»:

```
New File [Автозапуск 3-2.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 65220
65220 ← 253 54 0 255 49 84 255 237 115 61 92 241
65232 ← 253 203 2 134 17 224 254 1 39 0
65242 ← 205 60 32 195 3 19 22 11 0 19 1 17 4
65255 ← 72 65 32 144 80 79 67 69
65263 ←KE PACTET MOX C BEPECKOM
65318 ← 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
65338 ← 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
65358 ← 1343 2053 65220
65364 ← 0 0 0 62
65368 ← 0 126 66 66 66 66 66 0

17996 ← 65220
18000 ← 3
18001 ← 19 1 17 4 66 69 80 69 67 75
18011 ← 156 65220

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

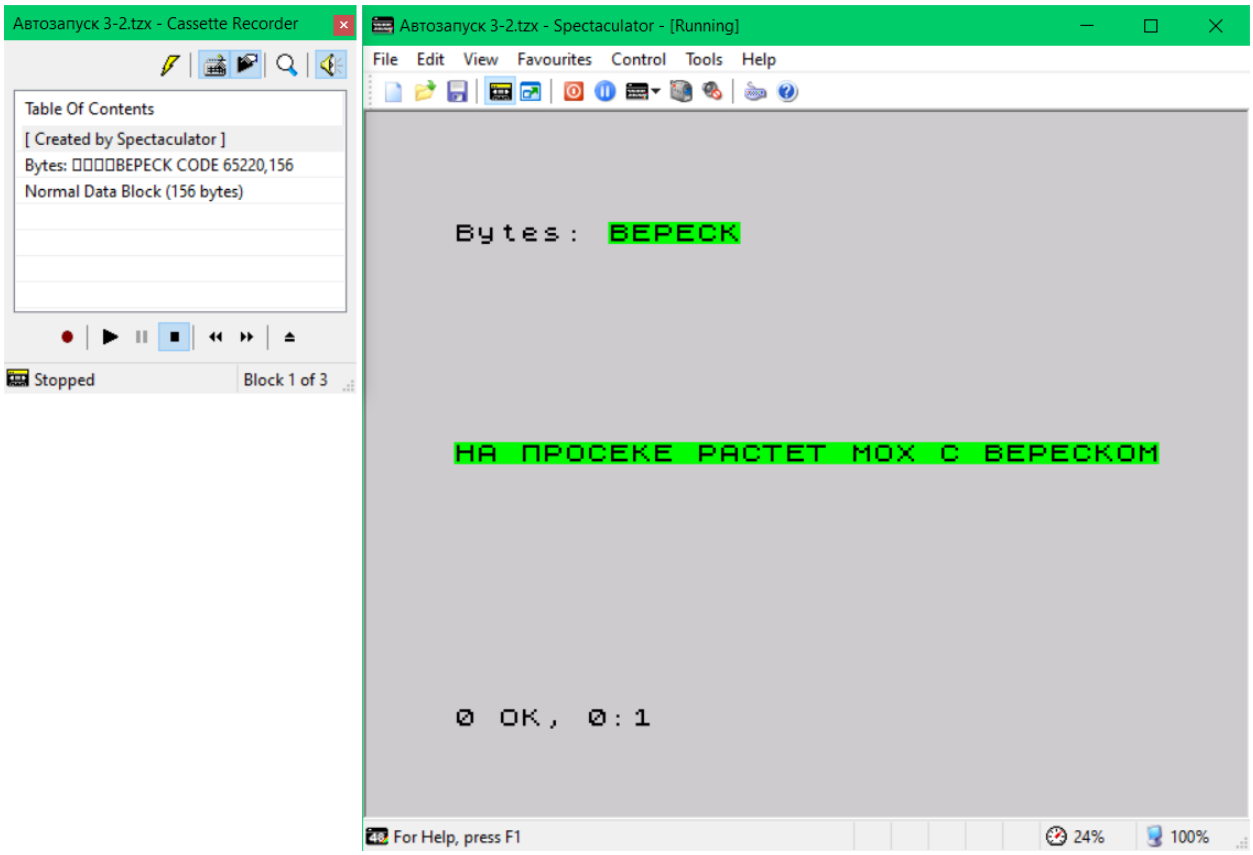


Рис. 386. Вывод текста после загрузки автостартующего блока «Bytes:».

Высоковольтная просека в сосновом лесу. Одноцепная линия, напряжением 35 кВ. Ряд бетонных столбов уходит вдаль за горизонт. На ковре из голубого мха растёт вереск и кустики голубики с ягодами. Кое-где торчат оранжевые шляпки подосиновиков.

Версия с окончанием в 65363:

```
New File [Автозапуск 3-1.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 65220
65220 ← 253 54 0 255 49 84 255 237 115 61 92 241
65232 ← 253 203 2 134 17 224 254 1 39 0
65242 ← 205 60 32 195 3 19
65248 ← 22 11 1 19 1 17 1 16 4
65257 ← КОТОВСКОЕ ОЗЕРО ЗАРАСТАЕТ МХОМ

65318 ← 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
65338 ← 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
65358 ← 1343 2053 65220

17996 ← 65220 0
18000 ← 3
18001 ← 16 4 17 1 75 79 84 79 66 79
18011 ← 144 65220

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

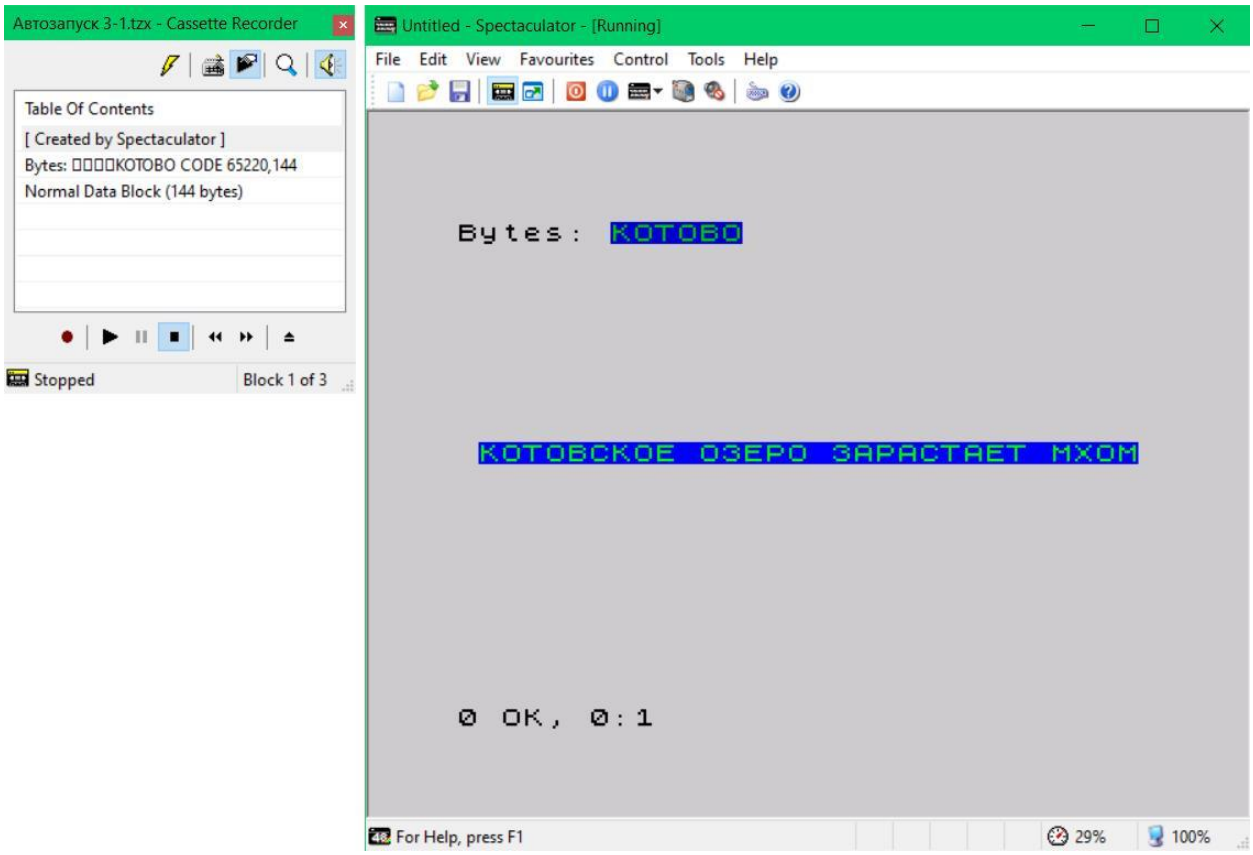


Рис. 387. Вывод текста после загрузки автостартующего блока «Bytes:».

И с началом в 65362:

```
New File [Автозапуск 3-3.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 65362
65362 ← 65368 0
65366 ← 0 62
65368 ← 253 54 0 255 49 84 255 237 115 61 92 241
65380 ← 253 203 2 134 17 116 255 1 39 0
65390 ← 205 60 32 195 3 19
65396 ← 22 11 0 19 1 17 1 16 7
65405 ←ТВЕРСКОЕ МОРЕ РАЗМЕРОМ С ОКЕАН

17996 ← 65362 0
18000 ← 3
18001 ← 19 1 17 1 16 7 77 79 69 80
18011 ← 73 65362

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

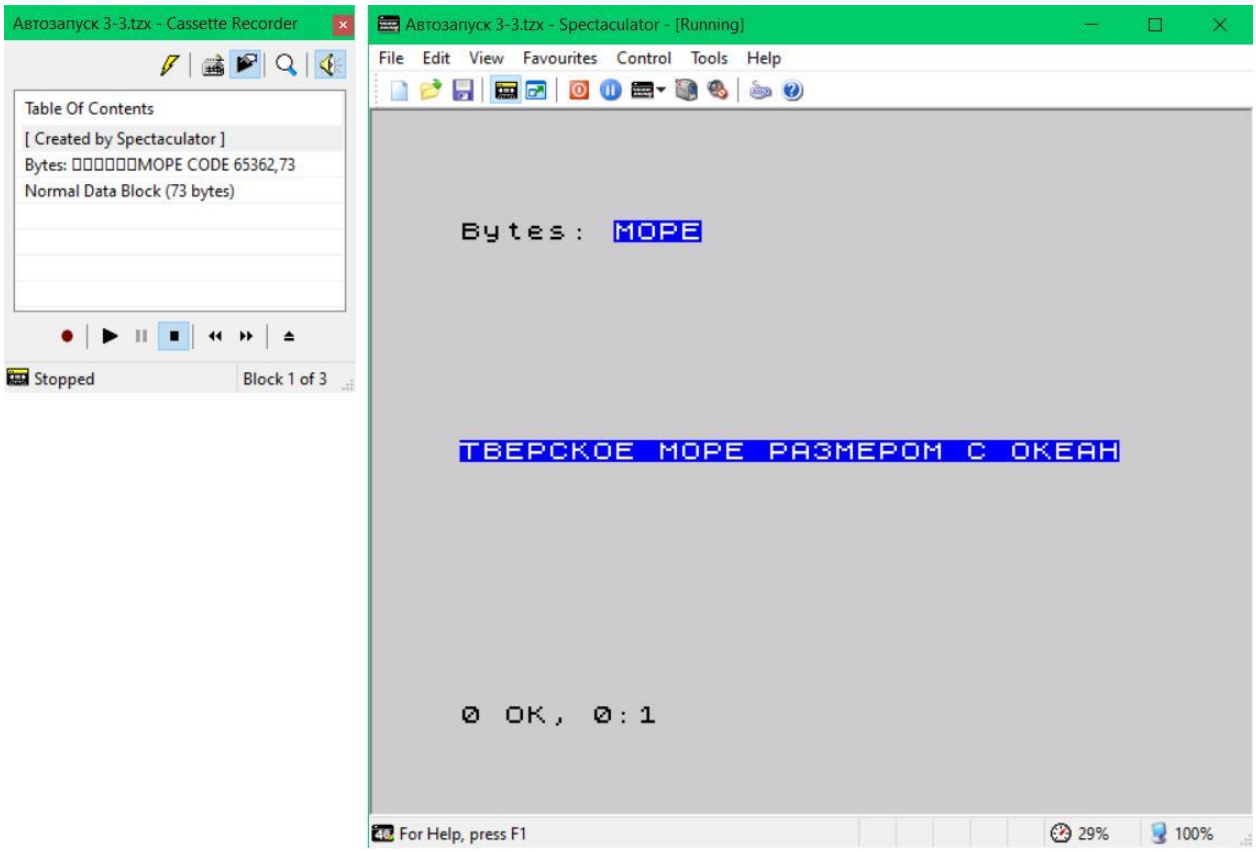


Рис. 388. Вывод текста после загрузки автостартующего блока «Bytes:».

И это правда. Особенно это хорошо ощущается, если ехать на Сапсане из Петербурга в Москву по Конаковскому району Тверской области. После платформы Московское Море поезд долго едет вдоль бескрайнего озера.

Глава 45
Автостартующий блок «Диана»

Краткое содержание: CH-ADD, точка 23760

И вот начинаются самые заковыристые и сложные автостарты, а это значит, что я вплотную подобрался к реализации «программы мечты». Без детального анализа работы BASIC системы с переменными, создать методом тыка то, о чем пойдёт речь, фактически невозможно. Именно поэтому больше тридцати лет ничего подобного не получалось.

Вопреки логике, четвёртая или жёлтая точка перехвата будет не по следующему, и самому нижнему значению из столбика SP, а по ячейке 23760.

Выполняемая BASIC строка автозагрузки будет выглядеть следующим образом:

: RANDOMIZE : [4 пробела] : PRINT "[AT 10, 12]НА ОХТЕ[AT 11, 3]ЗАСРАНА ОСТАНОВКА ТРАМВАЕВ"[ENTER]

Первый вопрос, который может возникнуть: На XYZ такая укуренная конструкция?

Реакцию понимаю. В общем, я пару раз переписывал эту главу, не зная как начать. После долгих раздумий, решил зайти сразу с примера, а уже на его анализе устроить разбор. Для создания желтого автозапускного блока №4 введите следующий алгоритм:

```
New File [Автозапуск 4-1.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 23760
```

```
23760 ← 58 249 58 32 32 32 32
23767 ← 58 245 34 22 10 12 17 6 19 1 72 65 32 79
23781 ← 88 84 69 22 11 3
23787 ←ЗАСРАНА ОСТАНОВКА ТРАМВАЕВ
23813 ← 34 13

17996 ← 23760 0
18000 ← 3
18001 ← 17 6 19 1 43 79 88 84 65 43
18011 ← 55 23760

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

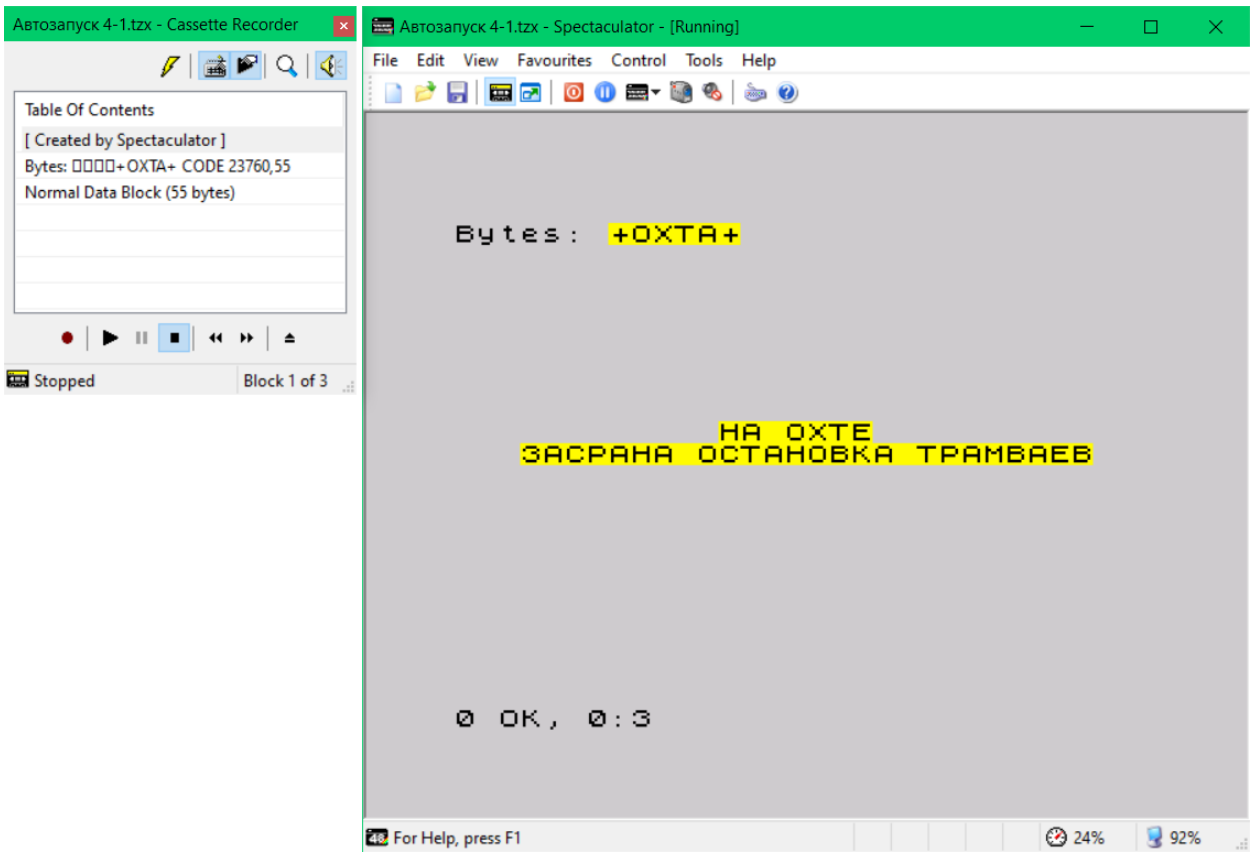


Рис. 389.

В главе «*Путешествие с командой LOAD*» подробно разбирался общий механизм команды. Предлагаю повторить, но уже с позиции только что созданного жёлтого автостарта, или как я его успел прозвать «ДИАНА».

По команде `LOAD` загрузились последние биты данных только что созданного «`Bytes: +OXTA+`». Начался каскадный выход на поверхность по красной (65358), затем по синей (65360), а следом по зеленой (65362) точкам.

Вынырнув по зелёной точке в программу `STMT-RET (7030)`, выполненная строка возвращается во владения BASIC и снова рассматривается с позиции синтаксиса команд:

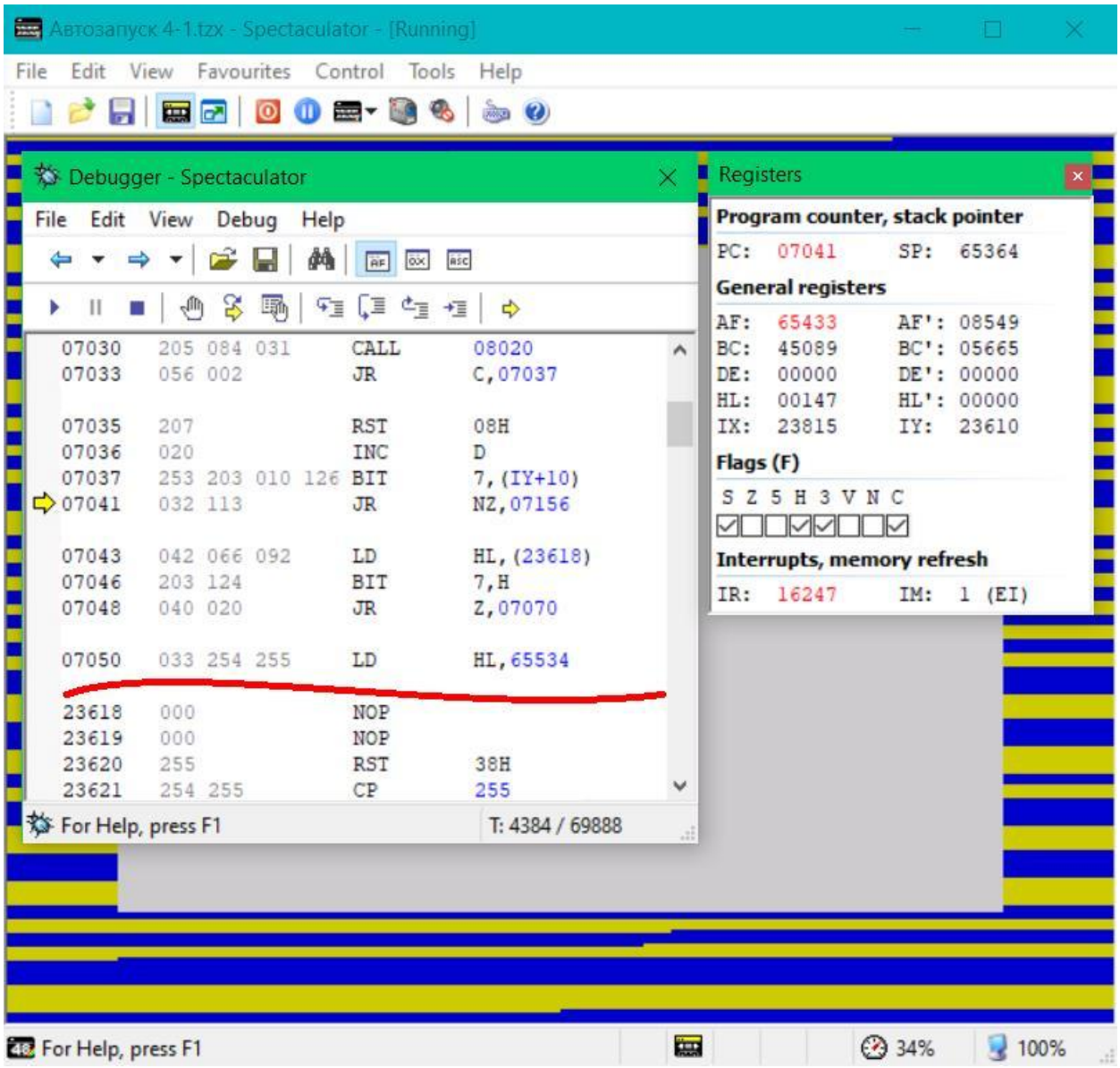


Рис. 390.

Первым делом опрашивается клавиатура на предмет нажатия комбинации «CAPS SHIFT+BREAK». По адресу 7037 проверяется номер следующего выполняемого оператора NSPPC (23620). Там стоит заглушка «255», которая всяко больше «128» (BIT 7, IY+10).

Следовательно, после проверки перспективных к выполнению строк, Стрелочка попадает на STMT-NEXT (7136) и смотрит символ, на который указывает CH_ADD (23645):

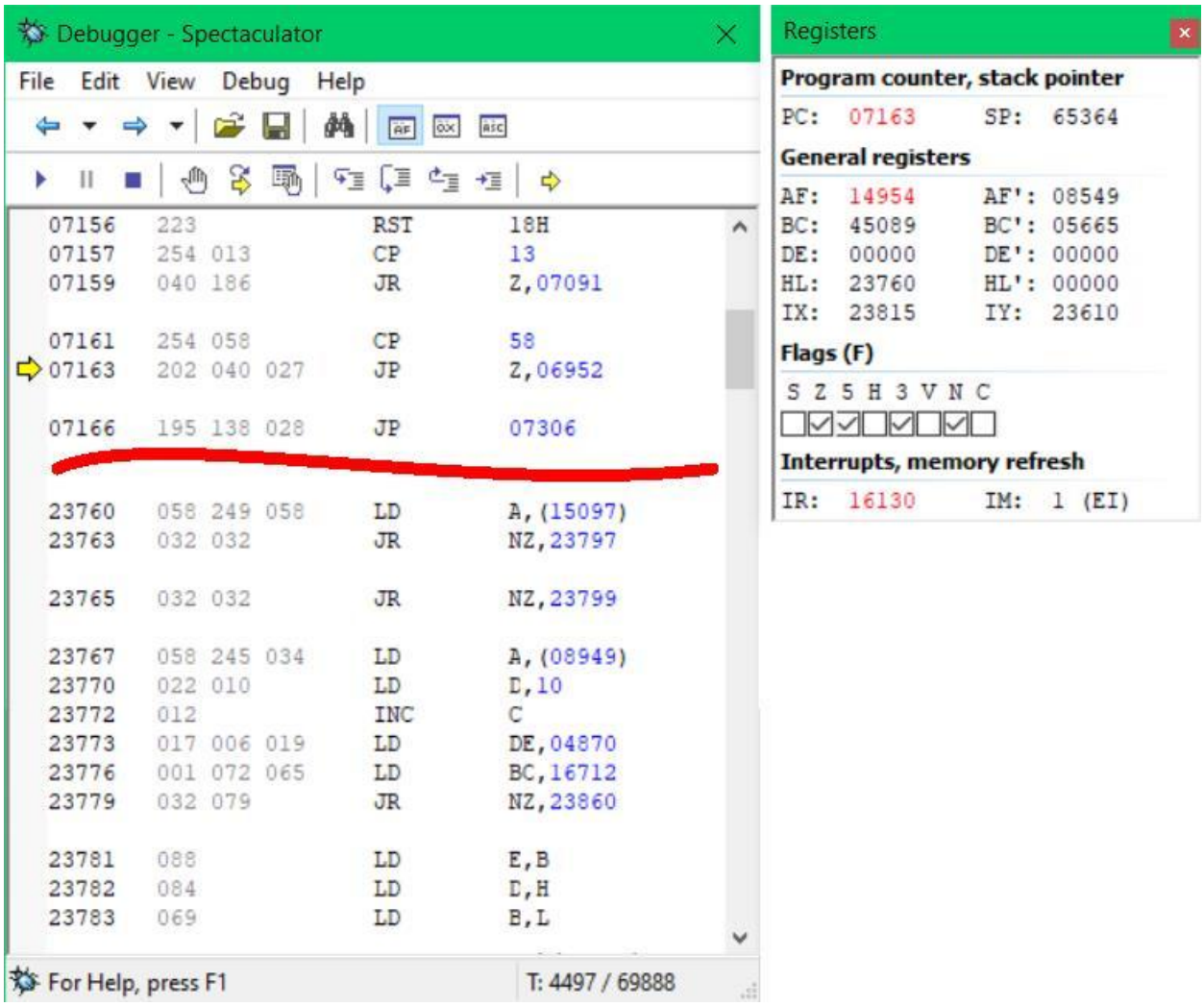


Рис. 391.

А после LOAD ""CODE он указывает на ячейку 23760, в которую сейчас записалось двоеточие. Таким образом, увидев там символ с кодом «58», Стрелочка отправляется по адресу 6952 в операторный цикл STMT-LOOP и вместо выхода, идёт на сверхурочные работы, решив продолжать выполнение программы.

Это единственный оптимальный вариант для дальнейшей зацепки и выполнения строки. Остальные два будут провальными. В случае обнаружения ENTER продолжится дальнейший выход на поверхность. При любом другом символе произойдёт переход на REPORT-C (7306), где и закончится выводом сообщения №11 (C Nonsense in BASIC).

Итак, внепланово попав в STMT-LOOP (6952), Стрелочка переключает курсор CH_ADD (23645) на ячейку 23761 и производит вторую проверку на тот же самый набор: ENTER, двоеточие и «всё остальное»:

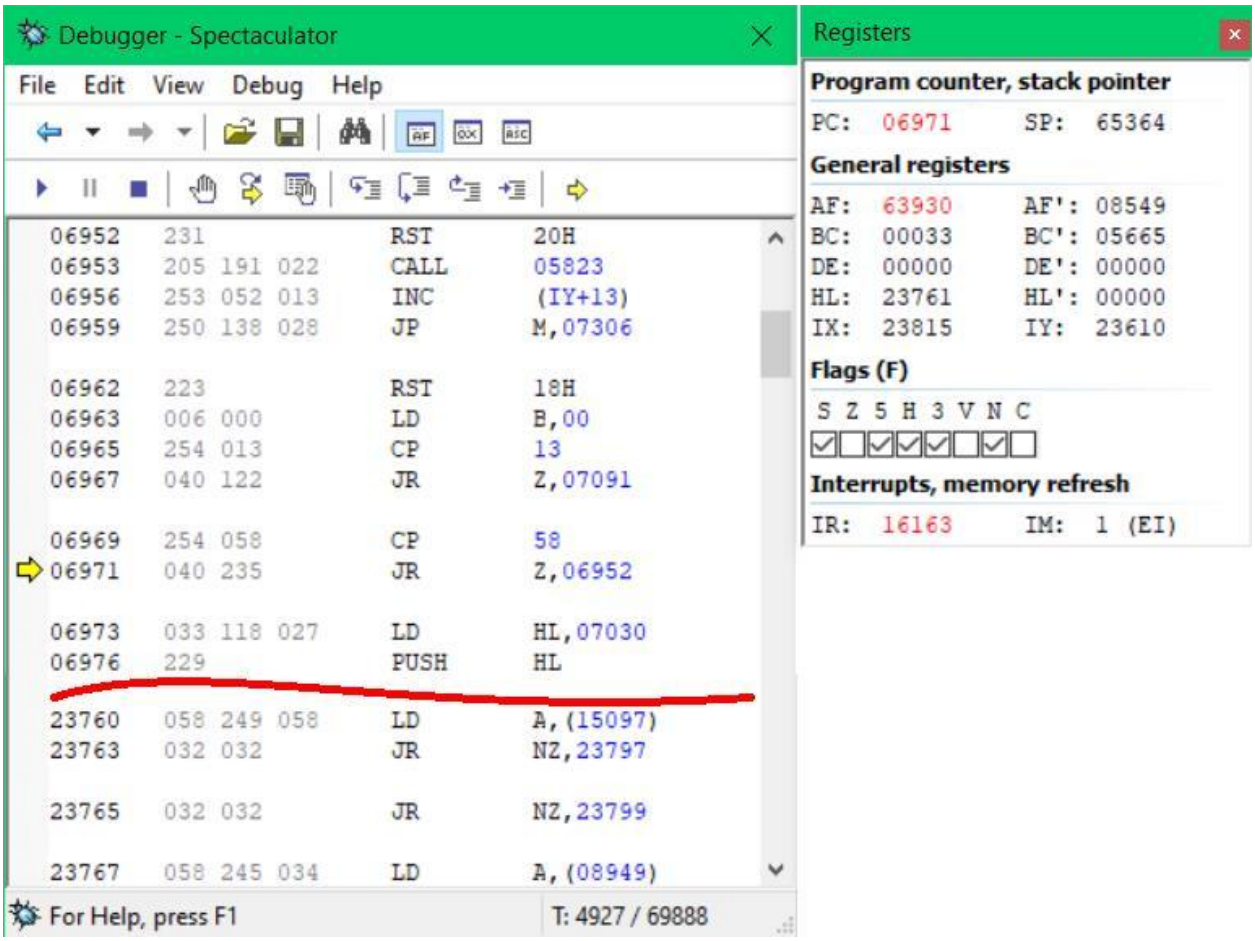


Рис. 392.

На первый взгляд, кажется, что Стрелочка попала на предыдущую программу или её полный клон, но посмотрите внимательно. В случае ENTER снова предлагается продолжить выход из программы, вынырнув в LINE-END (7091). По двоеточию произойдёт возврат к началу программы, и курсор CH_ADD (23645) перещёлкнется на следующую ячейку, а если там будет не то и ни другое…

В текущем случае в ячейке 23761 стоит код команды RANDOMIZE (248), поэтому Стрелочка проходит дальше и начинает выполнение команды. Узнаёте программу идентификации и опознания команды? Да, это она, родимая. Заманив Стрелочку двоеточием на предыдущей программе (23760=58), следующим символом ставится выполняемая команда (23761=248) и уже в этой программе (STMT-LOOP (6952)) строка пристыковалась и продолжает выполняться, забыв, что перед загрузкой тут стоял ENTER и простирались голые поля.

Вроде бы всё хорошо, строка подклеена, сейчас выполнится и всё отлично, а вот хренушки. Любое хорошее дело редко обходится без говнеца. Текущий случай не исключение.

Разметка области WORKSP, в чьи владения заехала новая программа, осталась прежней. Указатели в системных переменных никто не переписывал, поэтому они не изменились (WORKSP=23762). И вот при выполнении присоединенного куска, с этого адреса происходит самопоедание фрагмента данных уже запуска с первой команды:

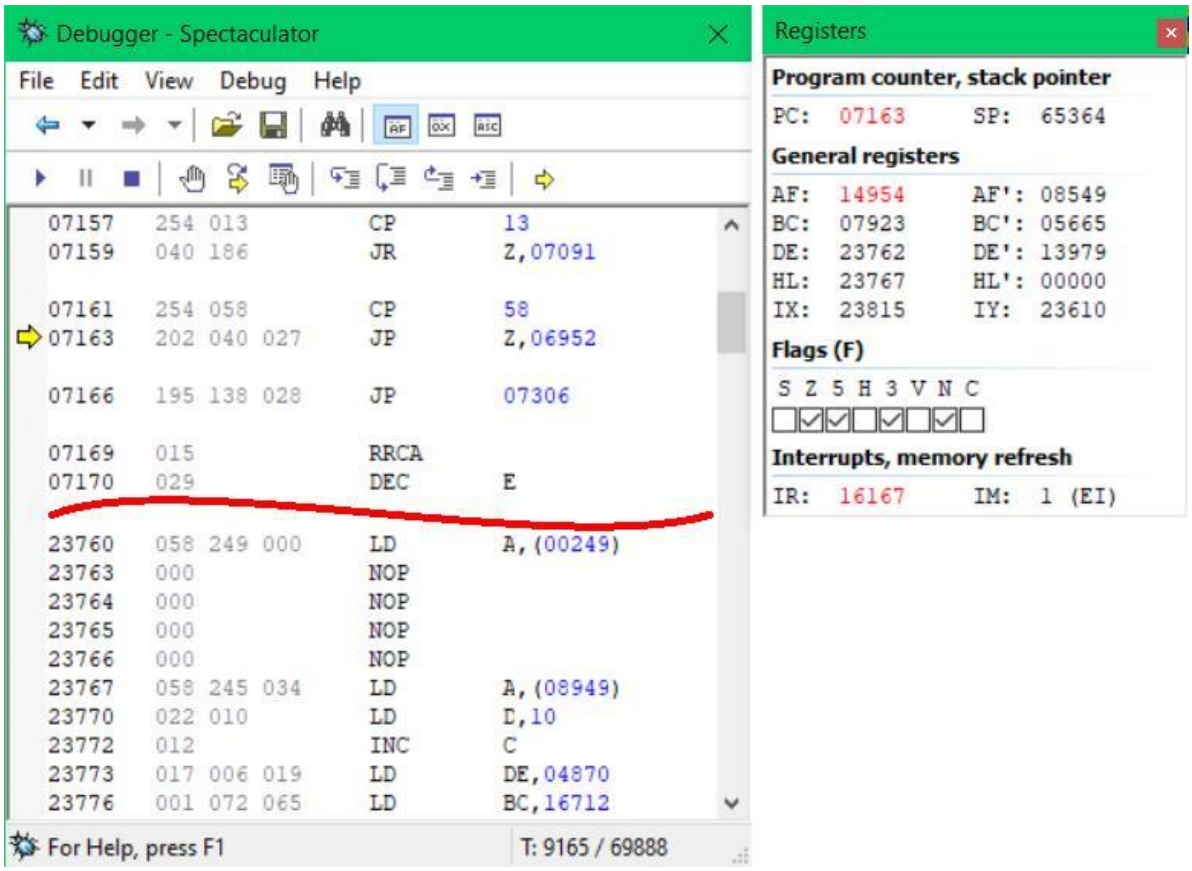


Рис. 393. Искажения в процессе выполнения команды RANDOMIZE.

Первой сработает пустая команда `RANDOMIZE`, и затрёт пятью нулями ячейки с 23762 по 23766. Следом своё дело сделает команда `PRINT` с текстом:

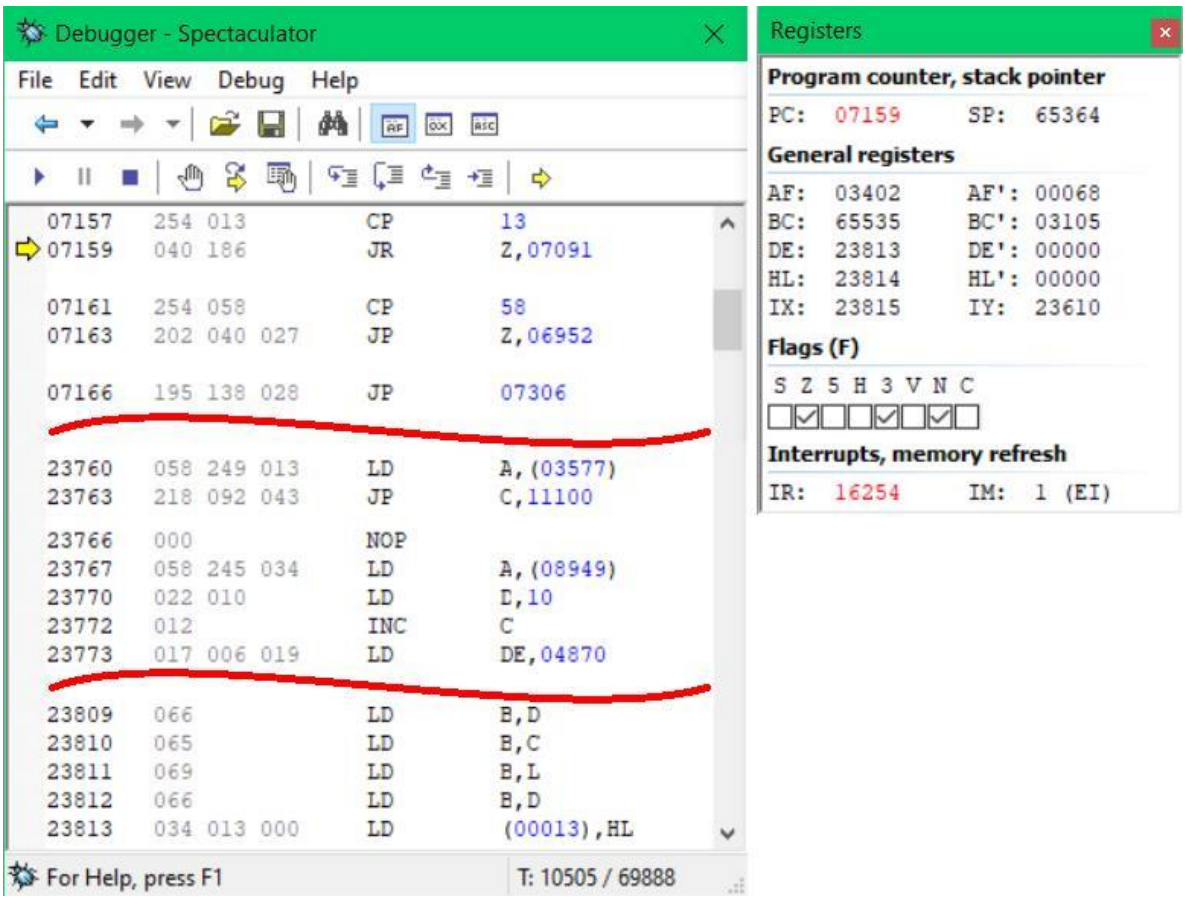


Рис. 394.

Но опять в те же пять ячеек. В зависимости от данных после команды, размер повреждения будет немного разниться. Как правило, для вывода символьной информации это 5 байт. В тоже время одиночные команды типа двоеточия (да, оно тоже считается за полноценную команду), пробела, `CLS` и `PRINT` не вносят искажения. А пустые команды `RANDOMIZE` и `RESTORE` затирают нулями впереди себя. Именно для этого следом за командой `RANDOMIZE` установлено двоеточие и четыре пробела.

Может возникнуть вопрос: если происходит искажение, то, как выполнится дальнейший кусок программы?

Искажения вносятся уже в процессе выполнения команды или группы. Указатель `CH_ADD (23645)` к этому времени авансом уходит к следующему двоеточию или на `ENTER` в конец строки. На запуск команды эти искажения не повлияют, но, к сожалению, вывод на экран догонят. Таким образом, противопоказано первым ставить `PRINT` с текстом. Именно для этого перед первым оператором поставлена холостая команда `RANDOMIZE`, а следом четыре пробела.

После пробелов, которые программа синтаксического анализа не считает за ошибку, стоит третье двоеточие и полноценная выполняемая строка, которая заканчивается кодом 13 для корректного выхода в BASIC. В данном случае следом не требуется ставить даже «128».

Вообще в качестве балласта для искажения, подойдут любые коды от 24 до 32. По таким символам комплекс «`RST 20`» вхолостую двигает `CH_ADD (23645)` до тех пор, пока не найдёт что-то существенное.

И снова есть в этом что-то странное и недосказанное. А почему вместо `RANDOMIZE` по адресу 23761, и второго двоеточия сразу было не поставить пробелы?

А вот этот момент совсем жёсткий. В том то и дело, что не поставить пробел. И не только пробел, а вообще, любой код, меньше «64»:

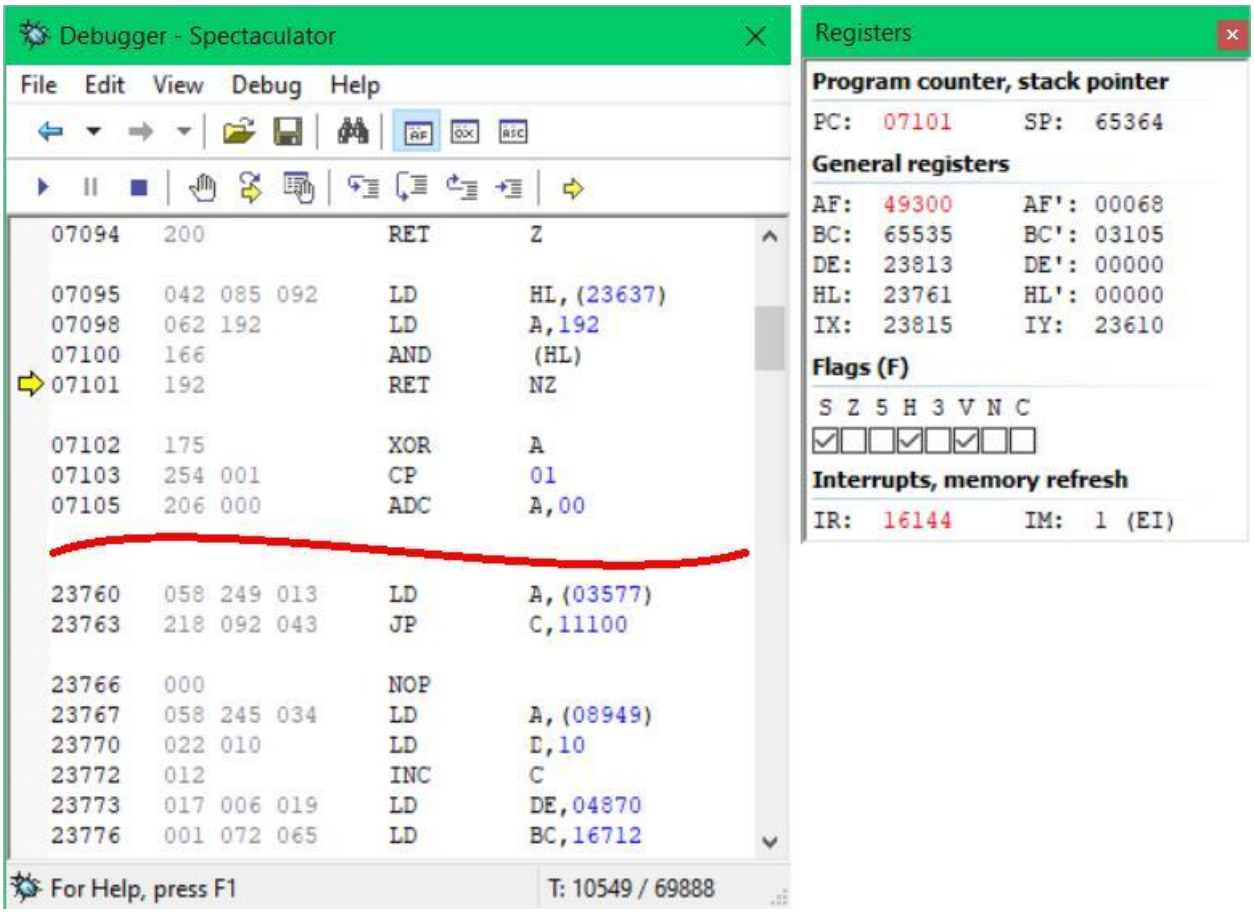


Рис. 395.

Нет, программа к этому моменту уже запустилась, но корректного выхода в BASIC можно не ждать.

Подумав, что в 23761 лежит номер строки, Стрелочка проваливается в LINE-USE (7103) а потом на NEXT-LINE (7121) и дальнейшую цепочку данных начинает рассматривать в формате BASIC-строки, пытаясь вычислить номер и длину. В системные переменные попадает искаженная информация. Соответственно, всё это окончится сообщением «C Nonsense in BASIC».

Таким образом, для успешного автозапуска, символ, стоящий в ячейке 23761 должен соответствовать сразу нескольким требованиям:

1) Быть командой.
2) Иметь код выше «63».
3) Не выдавать видимых эффектов на экран.

В идеале это должен быть одиночный оператор. Если пошарить по всем кодам, таких операторов будет несколько. Однако и они не все подойдут. Например, REM будет игнорировать после себя оставшийся хвост строки, несмотря на двоеточие. Использование LIST, LLIST и LPRINT тоже сомнительно.

Наиболее оптимальные кандидаты с нулевым искажением это CLS и PRINT. Однако, первая очистит экран, а вторая передвинет выводимый текст на позицию вниз от надписи «*Bytes:*». Если задумывается очистка экрана, это вариант беспроигрышный.

В данном примере я использовал пустую RANDOMIZE. Выступив в роли приманки для программы LINE-END (7091), она хоть и очистила впереди себя 5 байт (двоеточие и 4 пробела), но не затронула параметров экрана.

Общая схема автозагрузки №4 будет такой:

:[команда>63]:[4+ пробела]:[своя автостартующая программа][ENTER]

Конечно, зазор из пробелов можно дать на запас, но в конкретном примере достаточно минимального пакета.

До программы мечты остаётся два шага. Следующий вариант более лёгкий, но очень нужный для уверенного предпоследнего рывка. Если предыдущая версия была BASIC программой, то сейчас помощью подклеивания строки, предлагаю вывести текст из машинного кода, и возвратиться в режим ожидания BASIC:

```
New File [Автозапуск 4-2.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 23760
23760 ← 58 249 192 48 14 0 0 218 92 0    ; RANDOMIZE USR 23770
23770 ← 49 84 255 237 115 61 92 241      ; Универсальный выход в BASIC
23778 ← 253 203 2 134 17 242 092 1 43 0  ; Вывод текста
23788 ← 205 60 32 195 3 19 22 10 12
23797 ← 17 6 19 1 72 65 32 79 88 84 69 22 11 3
23811 ← ЗАСРАНА ОСТАНОВКА ТРАМВАЕВ

17996 ← 23760 0
18000 ← 3
18001 ← 17 6 19 1 43 79 88 84 65 43
18011 ← 77 23760

IX ← 18000
SP ← 17996
PC ← 2436
Trace
```

```
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

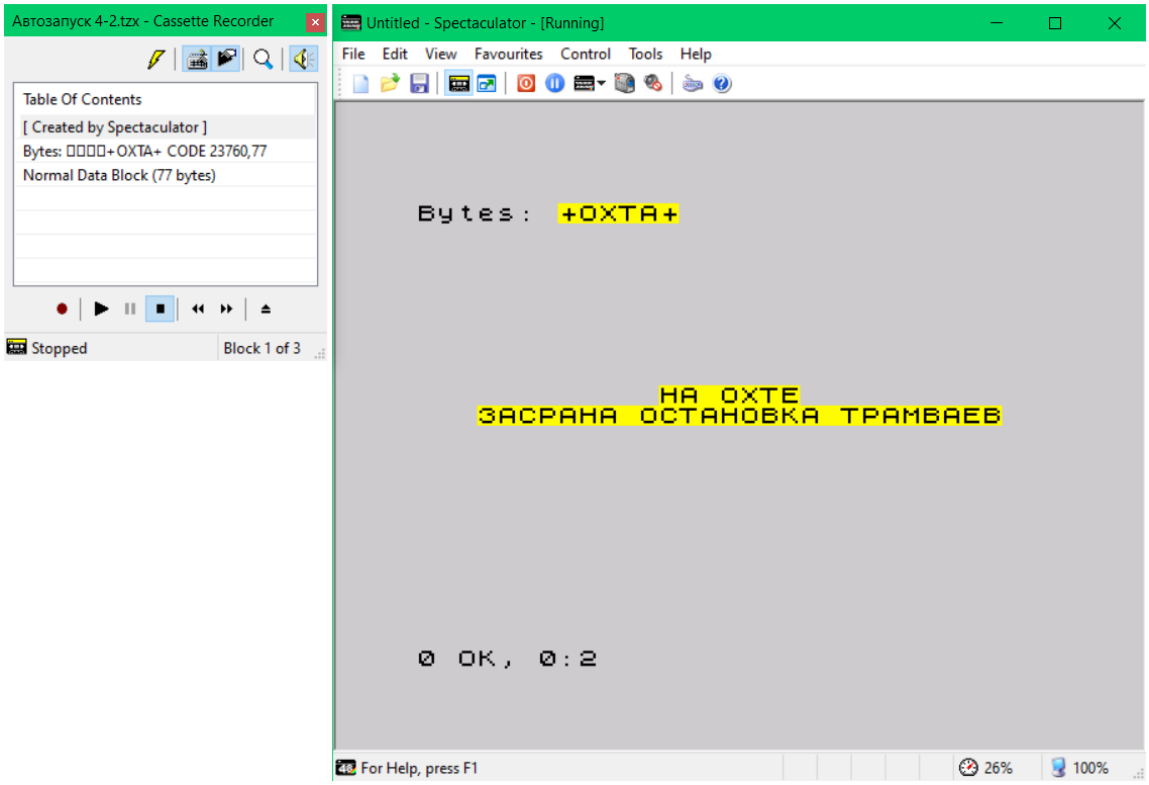


Рис. 396.

Обратите внимание, что `RANDOMIZE USR 23770` успевает запустить программу и только потом начинает затираться. В конце строки даже не понадобилось добавлять `ENTER`.

Из этого варианта можно получить комплексную модификацию программы. Добавив в конце BASIC-строки `ENTER`, можно упростить возврат из машинной программы до обычного `RET` по всем книжным правилам:

```
New File [Автозапуск 4-3.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 23760
23760 ← 58 249 192 48 14 0 0 219 92 0 13      ; RANDOMIZE USR 23771
23771 ← 253 203 2 134 17 233 092 1 43 0       ; Вывод текста
23781 ← 205 60 32 201 22 10 12
23788 ← 17 6 19 1 72 65 32 79 88 84 69 22 11 3
23802 ← ЗАСРАНА ОСТАНОВКА ТРАМВАЕВ

17996 ← 23760 0
18000 ← 3
18001 ← 17 6 19 1 43 79 88 84 65 43
18011 ← 68 23760

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]
```

```
BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

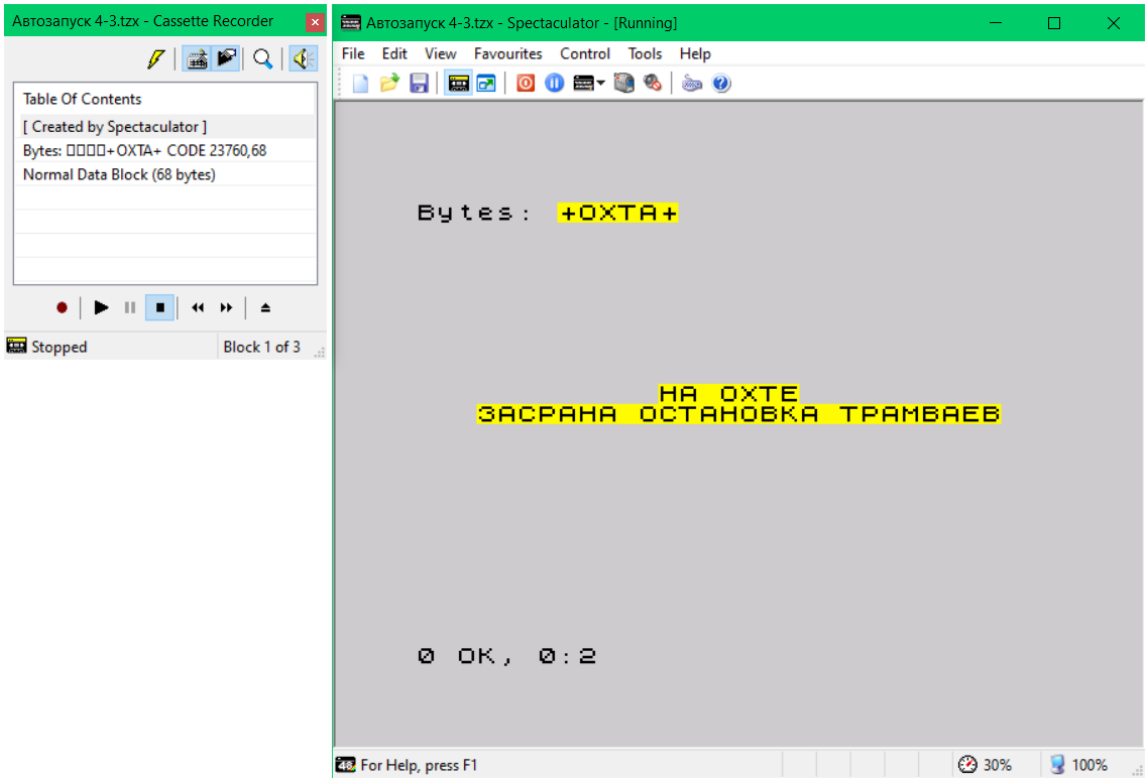


Рис. 397. Вывод текста после загрузки автостартующего блока «Bytes:».

Настал кульминационный момент. Впереди первый вариант «программы мечты 1993-2024». Именно из-за вот этого алгоритма начала зарождаться данная книга. Над подобным результатом я безуспешно бился десятилетия.

Итак, загрузка и автозапуск BASIC программы даже не с 23755, а с 23760:

```
New File [Автозапуск 4-4.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 23760
23760 ← 58 249 192 48 14 0 0 43 93 0     ; RANDOMIZE USR ремонтная
программа
23770 ← 0 1 29 0 245 173 49 50 14 0 0 12 0 0 59
23785 ← 34 19 1 17 6 72 65 32 79 88 84 69 19 8
23799 ← 17 8 34 13
23803 ← 0 2 41 0 245 34 32 32 32 19 1 17 6
23816 ←ЗАСРАНА ОСТАНОВКА ТРАМВАЕВ
23842 ← 17 8 19 8 34 13 128 13 128
23851 ← 1 81 0 17 203 92 33 218 92 237 176 235
23863 ← 43 43 34 89 92 43 34 75 92 205 176 22
23875 ← 33 1 0 34 66 92 253 54 10 1 201

17996 ← 23760 0
18000 ← 3
18001 ← 17 6 19 1 43 79 88 84 65 43
18011 ← 126 23760

IX ← 18000
SP ← 17996
PC ← 2436
```

```
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

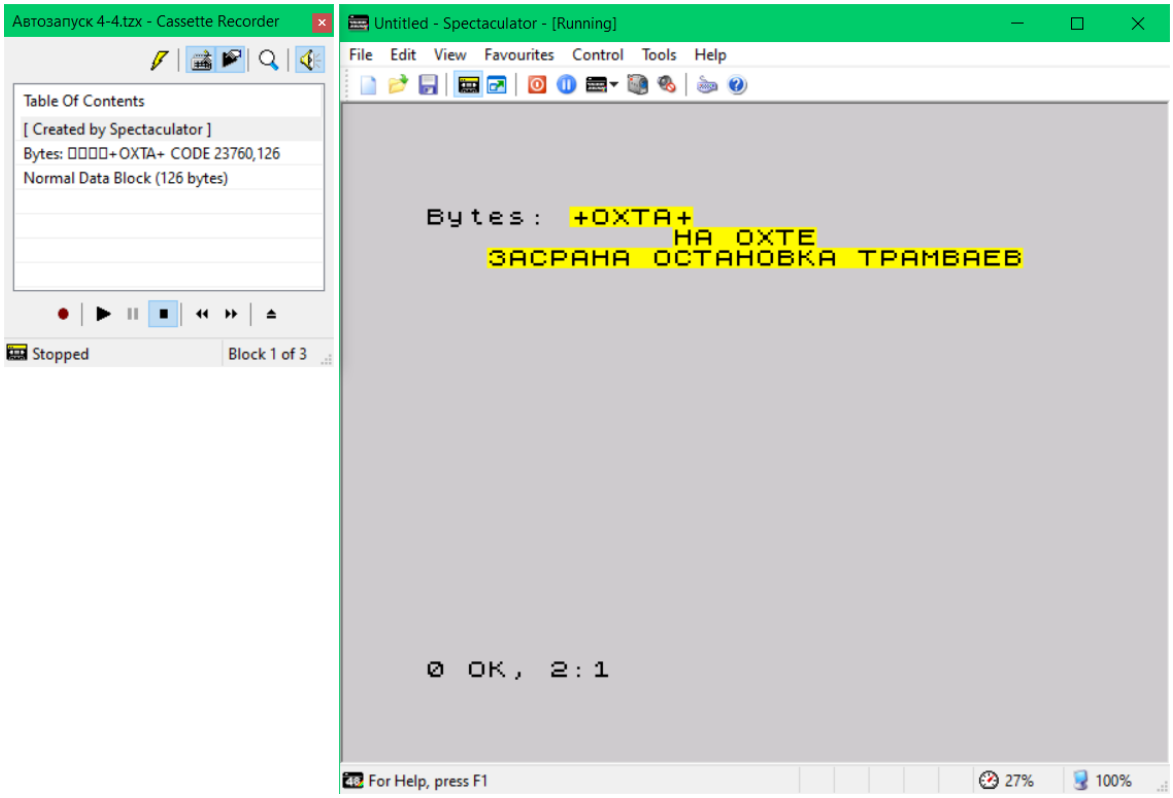


Рис. 398. Вывод текста после загрузки автостартующего блока «Bytes:».

А теперь нажмите ENTER:

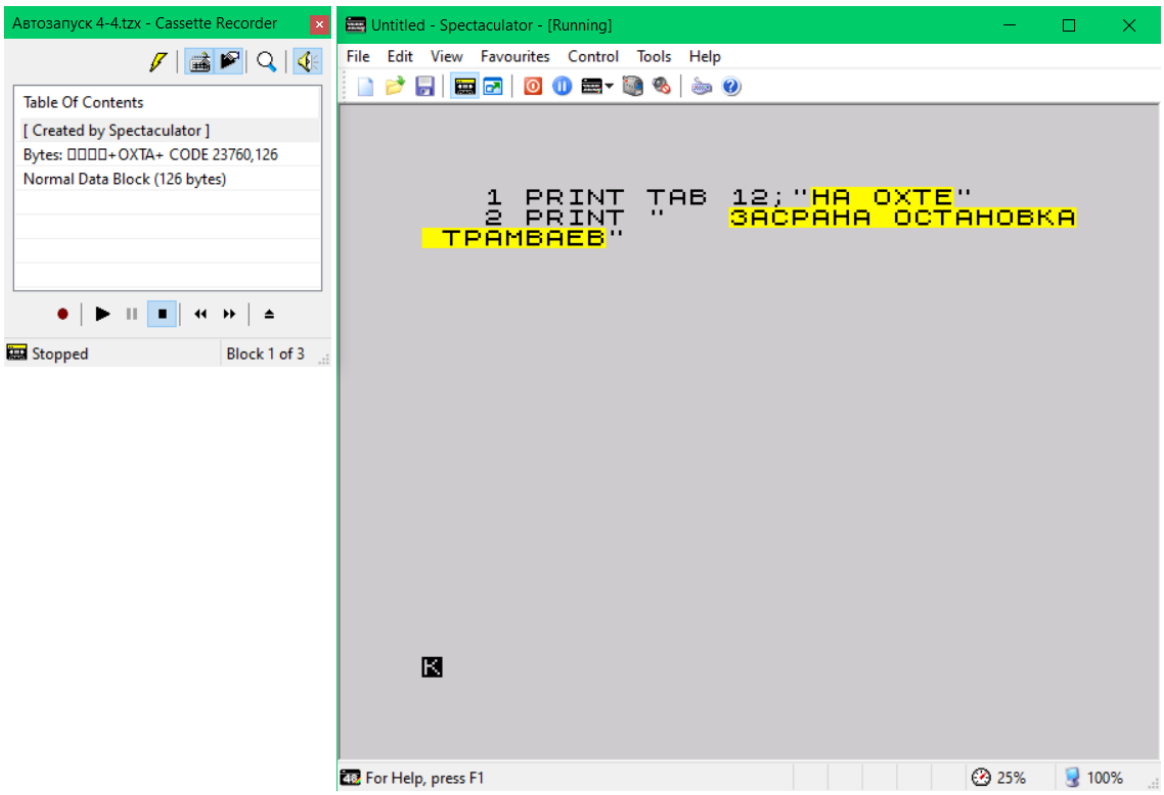


Рис. 399.

У меня непередаваемые эмоции. Неужели это свершилось? Вот она программа на BASIC целая и неповреждённая, которая записалась блоком «Bytes:» без помощи системных переменных, да еще и запустилась!

Да, случайности тут недостаточно. Неудивительно, что в 1993 и 2013 годах, методом тыка, не понимая, как работает BASIC, мне не удалось корректно даже записать программу под адрес от 23755 и выше.

Полагаю, остался один вопрос: почему «Диана»? Да фиг знает. Слово «CH_ADD» вызывает какие-то странные и непередаваемые чувства. Как будто я гуляю по Охте, от бывшей спектрумской фирмы НТК «Композит» (на Большеохтинском проспекте 15 корпус 2) в сторону Полюстровского парка. И вот там на «Тухачах» прохожу и любуюсь легендарным продуктовым магазином «Диана» (Маршала Тухачевского 5 корпус 1). Короче иду мимо «Дианы» и радуюсь жизни, примерно, как эта бабушка, попавшая попала в кадр моей фотографии:

Рис 400. Магазин «Диана». Улица Тухачевского 5 корпус 1. Фото 5 сентября 2010 года.

Ну еще почему-то слово «CH_ADD» ассоциируется с треугольником (Golden Triangle) на заставке игры ТЕТЯIS-2. В общем, трава в комнате у мальчика, из главы «*Ввод командной строки или ваш первый забористый ПРИНТ*», оказалась слишком забористая. Нужно повтыкать, полумать о жизни, да приступить к следующей главе.

Глава 46
Автостартующий блок «Никита»

Краткое содержание: NXTLIN, точка 23761

И это ещё не всё. Одна кульминация порождает другую. Поэтому перехожу к следующей, фиолетовой точке автозапуска. Она находится по адресу 23761. Да, эта та самая ячейка, в которую для прошлого автостарта приходилось ставить псевдозагрушку

из команды с кодом выше «63». А теперь представьте обратную ситуацию. Предыдущая жёлтая точка перехвата была проигнорирована. Для заглушки в ячейке 23760 остался стоять ENTER.

Увидев это, с точки 7159 Стрелочка переходит на подпрограмму LINE END в 7091. В прошлой главе эту подпрограмму рассматривал вскользь в качестве вспомогательной к точке CH_ADD. Ещё раз внимание на рисунок:

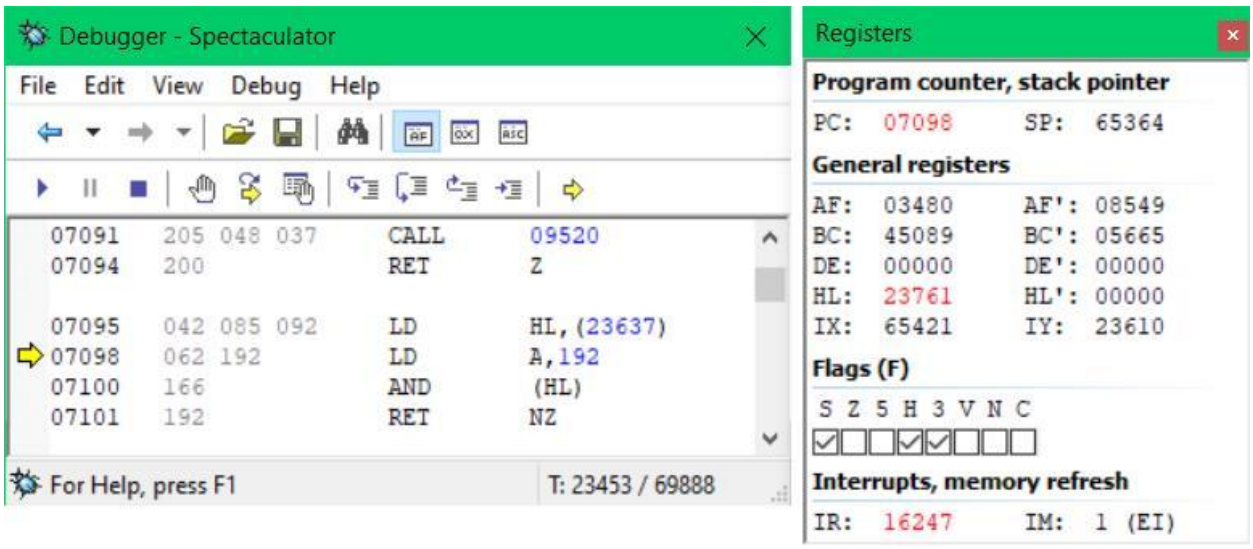


Рис. 401.

В «HL» из указателя NXTLIN (23637), забирается адрес текущей ячейки. При загрузке на чистый компьютер, этим адресом всегда будет 23761, который забивается еще на этапе подготовки к выполнению команды в ячейке 7121 *(см. главу «Путешествие с командой LOAD»)*. В обычном состоянии там перегородка «128», которая отгораживает E_LINE от WORKSP. В прошлой главе в качестве обманки я ставил команду RANDOMIZE.

Путём гашения первых шести битов числа ([23761] AND 192), выясняется характер содержимого ячейки. Если там значение 64 и более, значит, строк нет и можно выходить из программы.

Автостарт «CD_ADD» пропущен и не создаст помех. Теперь никто не мешает с 23761 подклеивать полноценные BASIC-строки с номерами и устанавливать их друг за другом.

Введите следующую программу:

```
New File [Автозапуск 5-1.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 23761
23761 ← 0 0 2 0 32 13              ; пустая строка в жертву затирания
23767 ← 0 0 34 0 245 34 22 11 0 16 3
23778 ← Hukuma XABAET Cocucky C Kanycmou
23810 ← 16 8 34 13 128

17996 ← 23761 0
18000 ← 3
18001 ← 16 3 75 97 110 121 99 109 107 97
18011 ← 54 23761

IX ← 18000
SP ← 17996
```

```
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

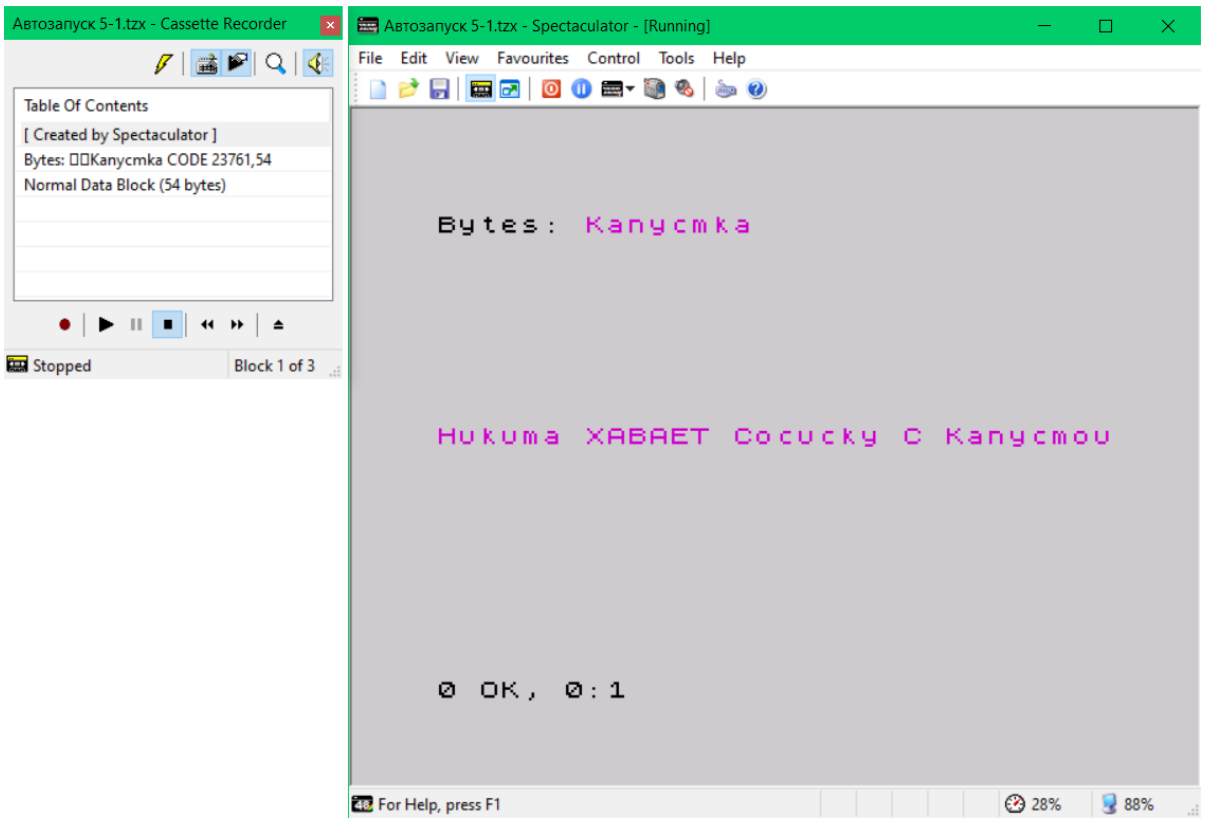


Рис. 402. Вывод текста после загрузки автостартующего блока «Bytes:».

Тут был хороший борщик. С капусткой, но не красный. Так… Сосисочки, ну ещё есть какой-то непонятный салат, куда крошат морковку, капусту и яблоки с ананасами. Вообще он меня бесит. Вот, ещё чего. Вкусный чай. Он так утоляет жажду, я чувствую себя человеком. Вот. Всё.

Имя переменной «NXTLIN» при беглом просмотре напоминает имя «Никита». Именно поэтому она моментально получила своё прозвище в честь легендарного мема про школьную столовую с YouTube. Следом за ней получила название и сама точка автостарта.

Характер затирания байт точно такой же, как и в прошлом примере. В данном случае, при выводе текста затирается 5 байт, но первая повреждённая строка успевает выполниться. Есть и отличия от прошлой точки: для корректного выхода в BASIC после `ENTER`'а последней строки обязательно нужен маркер «128».

Следующий вариант будет смешанного плана. С адреса 23761 запускается строка `RANDOMIZE USR 23776`, которая активирует программу в машинных кодах, а затем снова произойдёт выход в BASIC:

```
New File [Автозапуск 5-2.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 23761
23761 ← 0 0 10 0 249 192 48 14 0 0 224 92 0 13 ; RANDOMIZE USR 23776
23775 ← 128 253 203 2 134 17 238 092 1 37 0
```

```
23786 ← 205 60 32 201 22 11 0 16 1
23795 ←СВЕТА К ВЕСНЕ НАСОСЕТ НА САМОКАТ

17996 ← 23761 0
18000 ← 3
18001 ← 16 1 67 66 69 84 79 75 65 84
18011 ← 66 23761

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

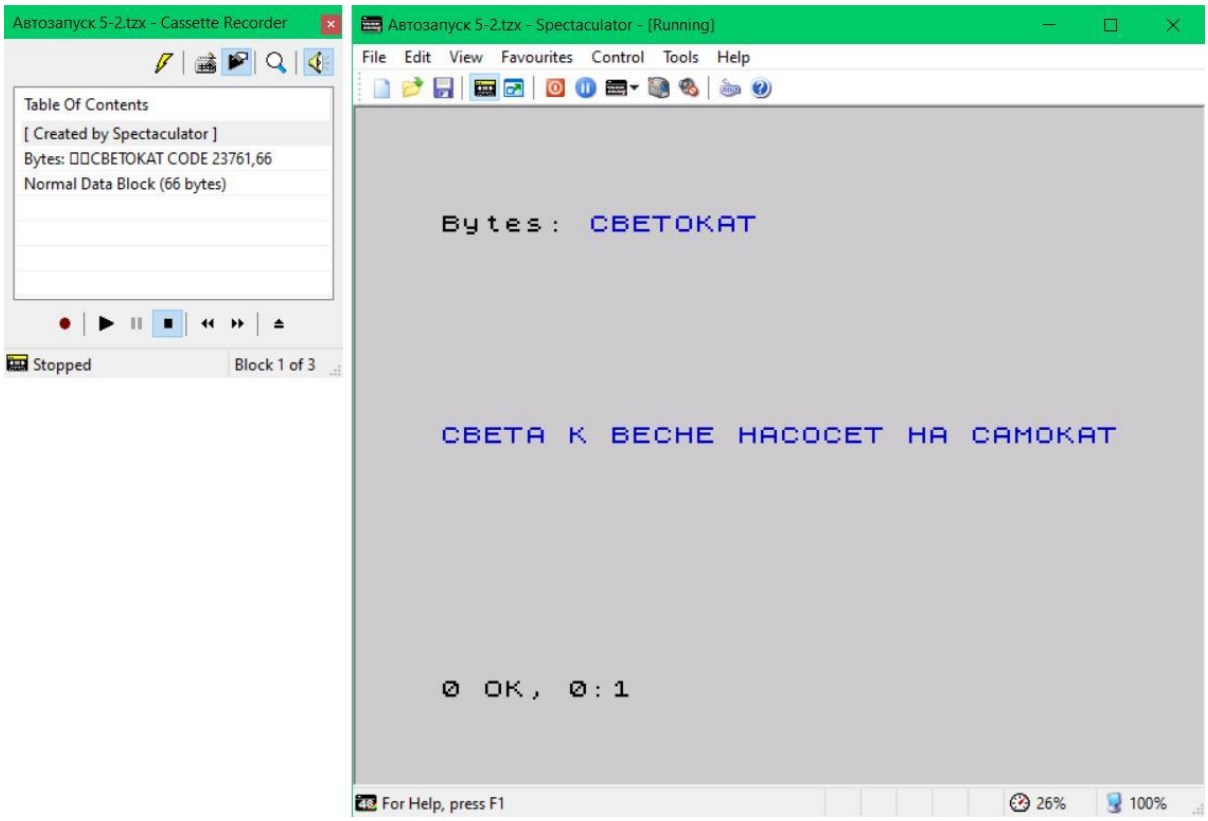


Рис. 403. Вывод текста после загрузки автостартующего блока «Bytes:».

Ну что можно сказать… у каждого своя цель в жизни. Вопрос в том, какой именно самокат. Если электро, то еще куда ни шло.

И теперь второй вариант программы мечты. По аналогии с предыдущей главой, с адреса 23761 можно точно также записать, подтащить и запустить BASIC программу:

```
New File [Автозапуск 5-3.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 23761
23761 ← 0 1 10 0 249 192 49 14 0 0 32 93 0 13
23775 ← 0 1 58 0 245 172 49 49 14 0 0 11 0 0 44
23790 ← 48 14 0 0 0 0 0 59 34 16 2
23801 ←В РОСТОВЕ КОРОНОВАН ВОР В ЗАКОНЕ
23833 ← 16 8 34 13 128 13 128
```

```
23840 ← 1 65 0 17 203 92 33 223 92 237 176 235
23852 ← 43 43 34 89 92 43 34 75 92 205 176 22
23864 ← 33 1 0 34 66 92 253 54 10 1 201

17996 ← 23761 0
18000 ← 3
18001 ← РОСТОВСКОЕ
18011 ← 114 23761

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

После запуска на экран выведется контрольный текст. Нажав ENTER, откроется сама BASIC программа.

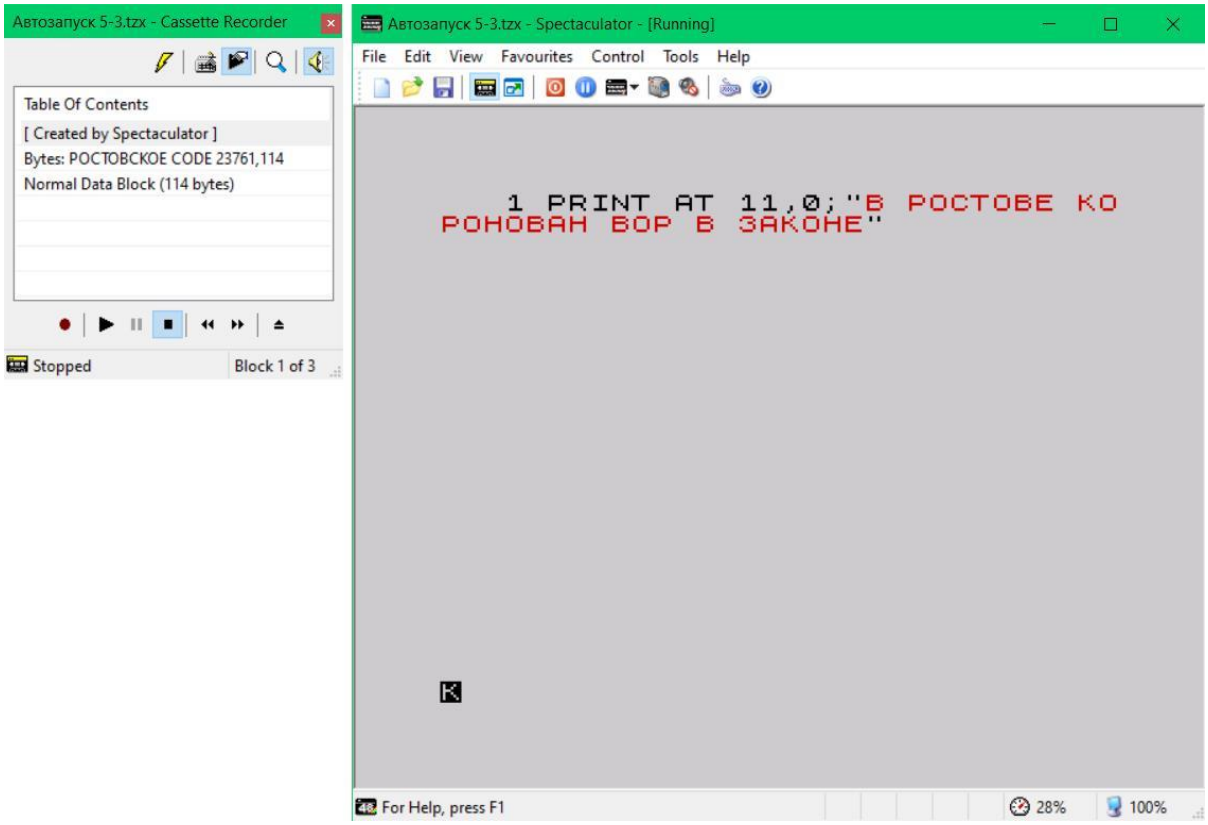


Рис. 404. Вывод текста после загрузки автостартующего блока «Bytes:».

Тоже жизненный пример. Как говорится, из песни слов не вынинешь и историю не перепишешь. Ведь эта книжка про мои любимые и счастливые девяностые.

Глава 47
Автостарт «Общий» или «Последний шанс»

Краткое содержание: MAIN-4, точка 65364

А теперь заключительная точка автостарта. Предположим, что в предыдущеё точке перехвата 23761 заглушка «128» осталась нетронутой. В таком случае, отработанная

строка покидает программу LINE-END, и по RET Z из 7101 выходит на поверхность в 4867 через два заключительных значения SP-столбика 65364 и 65365.

Что уникального будет в этой бирюзовой точке? А то, что перехват произойдёт после выполнения BASIC программы перед процессом выдачи сообщения «0 OK». Ну и сюда стекаются все ручейки с приготовленными сообщениями об ошибках и аварийных выходах по прерыванию записи и нажатию клавиши BREAK. Таким образом, бирюзовая точка это последний шанс перехватить управление, прежде чем Стрелочка уйдёт в свой «Дворец».

Для создания этого позднего автостарта, выставлять особых настроек не требуется. Для варианта автозагрузки сквозь SP-столбик, достаточно воссоздать оригинальные адреса всех предыдущих точек, и по окончанию своей программы просто оставить JP 4867.

Наберите и введите следующий алгоритм:

```
New File [Автозапуск 6-2.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 65234
65234 ← 253 203 2 134 17 226 254 1 37 0
65244 ← 205 60 32 195 3 19 22 11 1 19 1 17 5 66
65258 ← 144 84 69 75 32 145 79 32 72 65 75 146
65270 ←PKE OPET B TEATPE
65318 ← 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
65338 ← 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
65358 ← 1343 2053 7030
65364 ← 65234
65366 ← 0 62
65369 ← 66 70 74 82 98 66 0 0 126 66 66 66 66 66
65384 ← 0 66 66 66 62 2 60 0

17996 ← 65234 0
18000 ← 3
18001 ← 32 19 1 17 5 84 69 65 84 80
18011 ← 158 65234

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

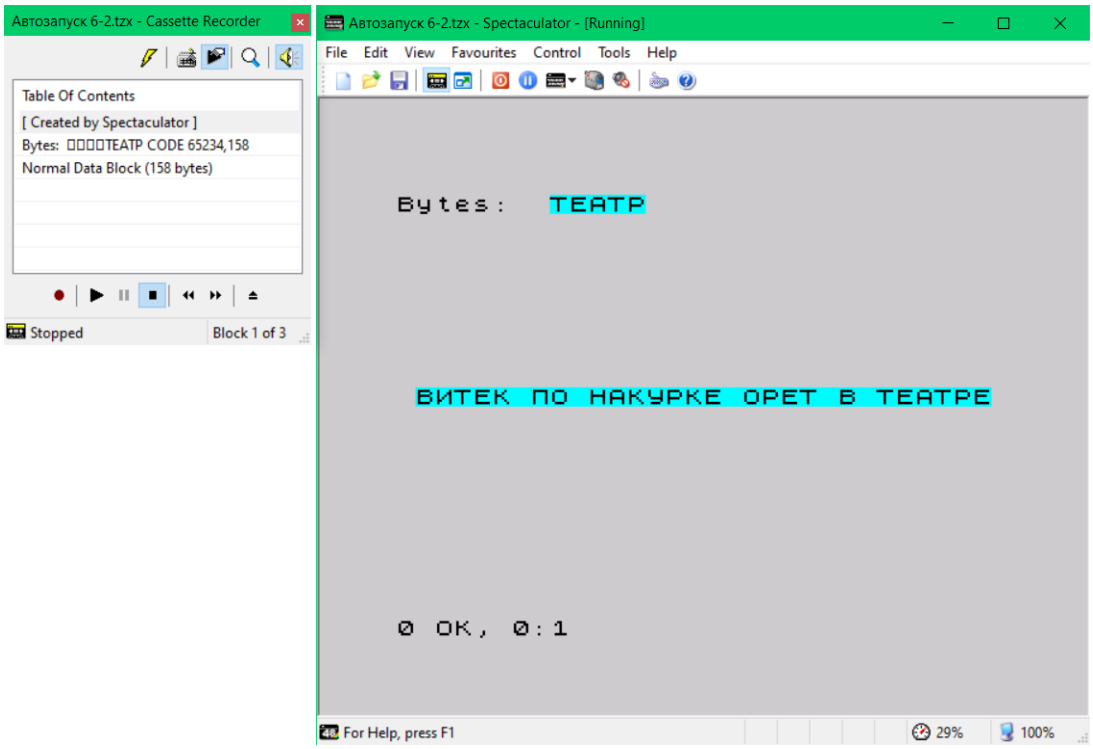


Рис. 405. Вывод текста после загрузки автостартующего блока «Bytes:».

И правда, что это я о каких то ютубных видосиках? Вот, пожалуйста, пример об интеллигенции и культурной жизни. Походы во всякие Мариинские театры это не фигли-мигли!

В автостарте с окончанием на точке перехвата, придётся восстановить три адреса возврата, кроме последнего:

```
New File [Автозапуск 6-1.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 65234
65234 ← 253 203 2 134 17 226 254 1 37 0
65244 ← 205 60 32 195 3 19 19 1 22 11 1 17 5
65257 ← Bumek no Hakypke Opem B Teampe
65318 ← 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
65338 ← 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
65358 ← 1343 2053 7030
65364 ← 65234
65366 ← 0 62

17996 ← 65234 0
18000 ← 3
18001 ← 32 19 1 17 5 84 69 65 84 80
18011 ← 134 65234

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

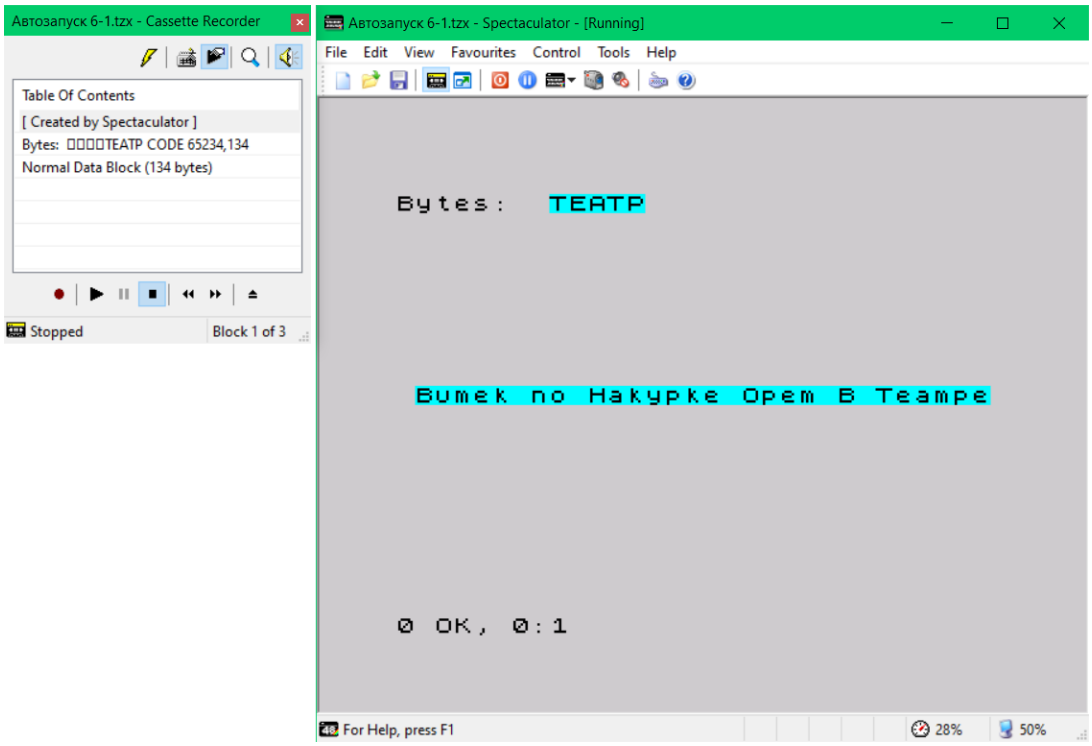


Рис. 406. Вывод текста после загрузки автостартующего блока «Bytes:».

Вариант с началом в точке прерывания самый простой:

```
New File [Автозапуск 6-3.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 65364
65364 ← 65368
65366 ← 0 62
65368 ← 253 203 2 134 17 104 255 1 37 0
65378 ← 205 60 32 195 3 19 19 1 22 11 1 17 5
65391 ←Bumek no Hakypke Opem B Teampe

17996 ← 65364 0
18000 ← 3
18001 ← 32 19 1 17 5 84 69 65 84 80
18011 ← 57 65364

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

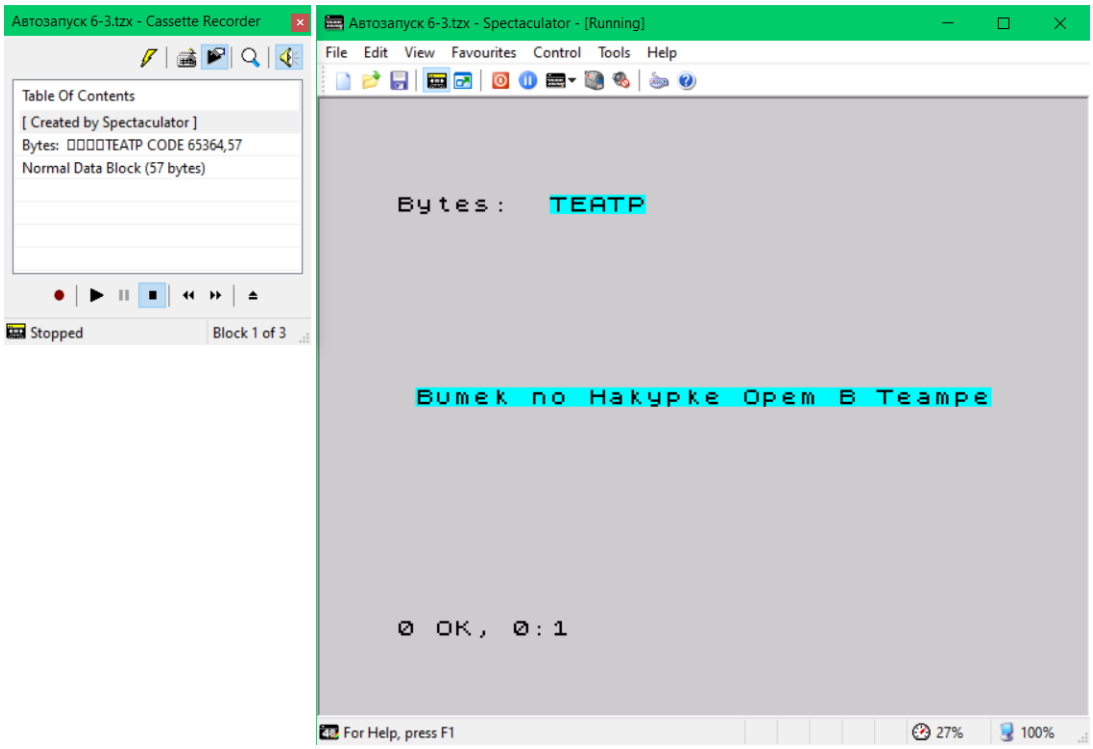


Рис. 407. Вывод текста после загрузки автостартующего блока «Bytes:».

Тут кроме фундамента восстанавливать уже нечего.

Глава 48
Читатель-писателю: Автозагрузки из старых книжек

Краткое содержание: НеZX-Forum, тайники ZX-Spectrum, анализ ошибок

КСЗ: После выхода в свет прошлых глав с цветными автостартами, к нам в редакцию пришло много благодарственных писем. Среди них хочется выделить послание с очень важным вопросом, который, увлекшись автостартами, совсем упустили из виду. Итак, нам пишет товарищ Гамнюков А. А. из города Мухосранска.

КОРР: Здравствуйте дорогая редакция! Я очень давно мечтал освоить автостартующие программы. К сожалению, во всех версиях книг и сборниках «Тайники ZX-Spectrum», именно в этой программе кучи ошибок, поэтому реализовать мечту никак не удавалось. Благодаря вашим статьям мой мир перевернулся. Мы с друзьями взахлёб читали и изучали секреты автостартов. Благодаря Вам, у нас в городке повысилась производительность предприятия, и в разы улучшился социальный климат. Поясню этот момент: я работаю на градообразующем предприятии по переработке помёта и навоза «ПомётСтрой». Наверняка вы слышали о нем, предприятие то известное. Так вот, после того, как пол завода прочитало про запуск BASIC строк с адреса 23760 и 23761, производительность резко увеличилась. Теперь наше предприятие превратилось в концерн, с собственным научно-исследовательским институтом. Мы стали лидерами не только по переработке навоза любой категории, но и сделали ряд важных открытий в отрасли по передвижению на пуканной тяге. В знак благодарности наш директор готов выслать Вам большую оптовую партию продукции отборного качества.

КСЗ: Акакий Акакиевич! Большое спасибо за тёплые слова и предложенную спонсорскую помощь! Мы очень тронуты, но пока вынуждены отказаться от Вашего заманчивого предложения. К сожалению, у нас пока переизбыток своего …овна. На складах завалялся голубиный, воробьиный, соловьиный и даже кукушачий помёт. Сами не знаем, куда девать, но как только возникнет потребность, мы обязательно обратимся к Вам.

И еще: Акакий Акакиевич ищет единомышленников и попросил опубликовать свой адрес. С удовольствием выполняем просьбу товарища Гамнюкова. Вот обратный адрес для писем:

Куда: г. Мухосранск, улица Навозная 5
Кому: Гамнюкову Акакию Акакиевичу

Итак, возвращаюсь к вовремя поднятой теме. Предлагаю посмотреть тот самый пример автостарта из клонированных изданий «*Тайники ZX-Spectrum*»:

В этом разделе был описан способ запуска блоков машинного кода путем считывания в область машинного стека. Чтобы такой блок запустился, достаточно ввести инструкцию LOAD""CODE, которая загрузит его. Чтобы убедиться в этом на практике, введите и запустите программу с листинга 2:

ЛИСТИНГ 2

```
10 CLEAR 65361: LET S=0
20 FOR N=65362 TO 65395: READ A: POKE N,A: LET S=S+A:
   NEXT N
30 IF S3437 THEN PRINT "OSHIBKA V STROKE DATA": STOP
40 SAVE "BEEP" CODE 65362,34
50 DATA 88,255,3,19,0,62,221,2,29,6,8,197,17,30,0,33,112,5,205
60 DATA 181,3,33,0,8,43,124,18,1,32,251,193,16,235,221,225,201
```

Она запишет на ленте короткий блок машинного кода, который будет запускаться самостоятельно. После записи этого блока освободите память с помощью RANDOMIZE USR 0 или RESET (по возможности переставьте RAMTOP на нормальное значение с помощью CLEAR 65367) и прочтите его, вводя LOAD""CODE. Цель этого блока - проинформировать о том, что он запустился: он это делает несколькими звуковыми сигналами. Вы услышите их сразу после считывания программы, как только с рамки исчезнут гранатово-желтые полосы.

ГЛАВА 7

ДЕКОДИРОВАНИЕ ЗАКОДИРОВАННЫХ БЛОКОВ ТИПА «BYTES»

Что означает то, что программа или блок закодированы? Кодирование это род весьма простого шифра, делающего невозможной правильную работу программы. Для её запуска служит специальная декодирующая процедура.

151

6 этом разделе был описан способ запуска блоков машинного кода путем считывания а область машинного стека. Для запуска такого блока достаточно ввести инструкцию LOAD""CODE, которая и запустит его.Чтобы убедиться в этом на практике, запустим программу, приведенную ниже:

```
10 CLEAR 65361: LET S=0
20 FOR N=65362 TO 65395: READ A
  :POKE N,A: LET S=S+A: NEXT N
30 IF S<>3437 THEN PRINT "ОШИБКА В СТРОКЕ DATA"
  :стоп
40 SAVE "BEEP" CODE 65362,34
50 DATA 88,255,3,19,0,62,221,2,
        29,6,8,197,17,30,0,33,0,8,43,124,18
60 DATA 181,3,33,0,8,43,124,18,
        1,32,251,193,16,235 221,225,201
```

Она запишет на ленте короткий блок машинного кода, который будет запускаться самостоятельно. После записи этого блока освободим память с помощью RANDOHIZE USR 0 или RESET (по возможности, нужно переставить RAMTOR в нормальное значение с помощью CLEAR 65367) и прочтем его, введя LOAD""CODE. Задача этого блока - проинформировать, что он запустился - что и делается несколькими звуковыми сигналами, которые появятся сразу после считывания программы, как только с рамки исчезнут гранатово - желтые полосы.

Рис. 408. Вырезки из пары книг «Тайники ZX-Spectrum».

В этих вырезках из двух книжек разных издательств шикарно всё: от кода программы до элементарной арифметики и описания. Роднит их одно: чей-то кривой перевод с польского, о чём нетрудно догадаться по информации из предыдущих глав и меткам из программ.

Предлагаю начать анализ с машинного кода. Прежде чем попробовать скопировать это в память посмотрите внимательно на обе цепочки:

Книга 1:

65362 ← **88 255 3 19 0 62 221 2 29 6 8 197 17 30 0 33 112 5 205 181 3 33 0 8 43 124 18 1 32 251 193 16 235 221 225 201**

Книга 2:

65362 ← **88 255 3 19 0 62 221 2 29 6 8 197 17 30 0 33 0 8 43 124 18 181 3 33 0 8 43 124 18 1 32 251 193 16 235 221 225 201**

Синим цветом выделены одинаковые фрагменты, красным с подчёркиванием несовпадающие. Рассинхрон значений начинается с 17-го числа. Потом числа в цепочках снова становятся одинаковыми. Кроме того, в первой на два числа меньше, чем во второй.

По логике вещей, во второй версии при вёрстке книги, в середину цепочки два раза вставили один и тот же фрагмент «**0 8 43 124 18**». Это понятно, что наборщик увидел число «33» второй раз, запутался и по ошибке повторил предыдущий кусок.

Дальше интереснее. В обеих книгах, BASIC-строка для записи одинаковая:

```
SAVE "BEEP"CODE 65362,34
```

И всё бы ничего, но в первой цепочке 36 чисел, а во второй 38. Что с арифметикой? Где тут 34? Можно сделать вывод, что в книгу закралась еще пара лишних цифр.

Теперь касаемо программы. Общая задумка хорошая, если после `LOAD` нет выполняемых команд. Поскольку блоки кодов будут записываться в память с 65362, то согласно терминологии этой книги, автостарт должен происходить по зелёной точке, подменив возврат на проверку окончания строки `STMT-RET (7030)`. Перехватить по «Чистому» автостарту, не дав уйти в BASIC, решение логичное. На подмену стоит адрес 88 255, что равно (255*256)+88=65368. Это еще лучше, потому что программа запустится. Следующим стоят 3 19, и означают знакомый адрес 4867, который там и должен быть для возврата на печать сообщений. Последним угадывается основание 0 и 62. Таким образом, к реставрированной конструкции SP-столбика и первым 6-ти значениям вопросов нет. А вот дальше… тихий ужас. Посмотрите на введённую в «*Debugger*» цепочку:

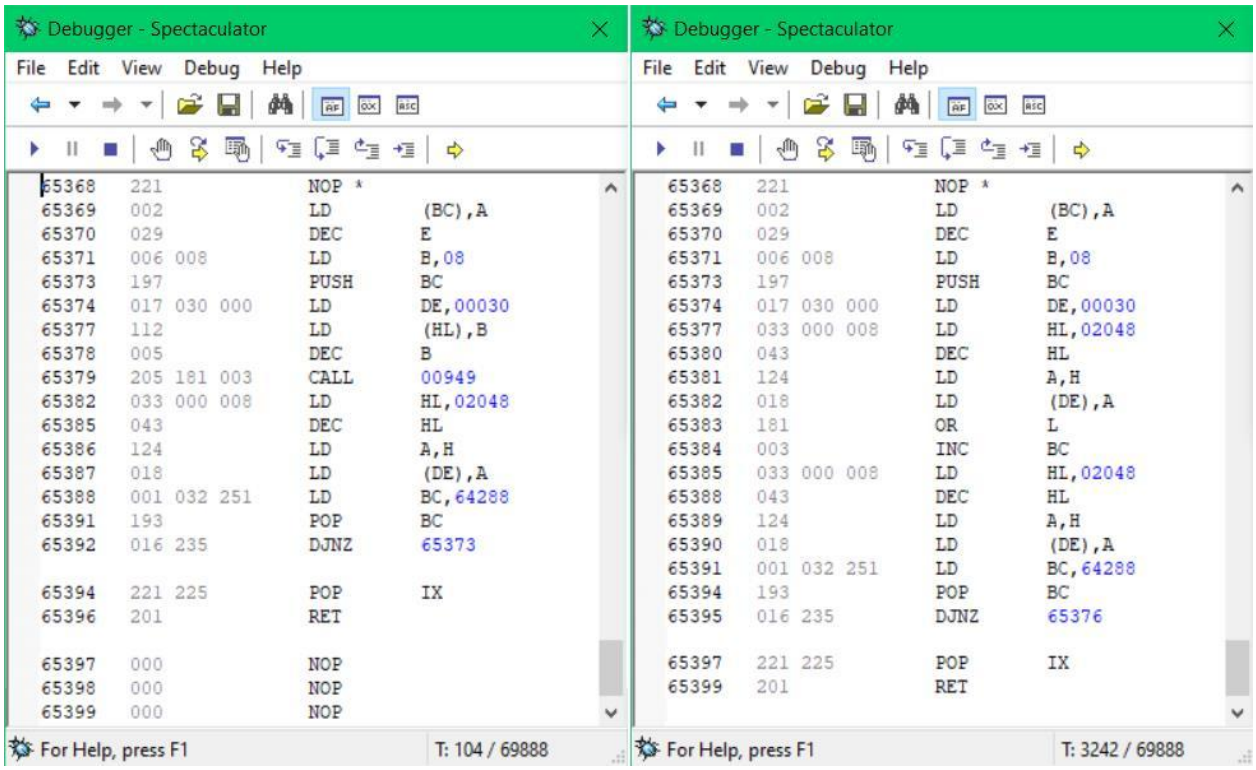


Рис. 409.

О!!! Что это было??? Начало предполагаемой программы на локальную типографскую ошибку уже никак не тянет. В обоих изданиях косяк один. Судя по команде «POP IX» перед «RET», в начале кусок недописанной команды из набора «IX». Как вариант «PUSH IX», но нахрена? Понятно только то, что, стоящее в конце «RET» намекает на планируемый возврат назад через 65364/65, чтобы по взятому адресу 4867 вернуться в BASIC, обогнув контрольные точки 23760 и 23761.

На самом деле, идея и каркас программы сделаны знающим человеком. Это хорошо прослеживается. Но вот начинка не выдерживает никакой критики. После «Испорченного телефона» эту программу уже не отреставрировать. Виден вызов подпрограммы BEEPER (949), но вот этот окружающий мусор, мне непонятен. Что подразумевалось автором оригинала, теперь уже не расшифрует никто. Зная качество передачи информации через бумажные издания, лучше бы для примеров оставляли вывод одиночной буквы.

А прототип этой программы с другой начинкой вы сможете создать элементарно:

```
New File [Автостарт Пиратский.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 65362
65362 ← 65368 4867
65366 ← 0 62
65368 ← 253 203 2 134 17 105 255 1 37 0
65378 ← 205 60 32 195 3 19 201
65385 ← 18 1 22 11 0
65390 ←ТРАКТОР В СОВХОЗЕ ТОНЕТ В НАВОЗЕ

17996 ← 65362 0
18000 ← 3
18001 ← 32 18 1 84 80 65 75 84 79 80
18011 ← 60 65362
```

```
IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

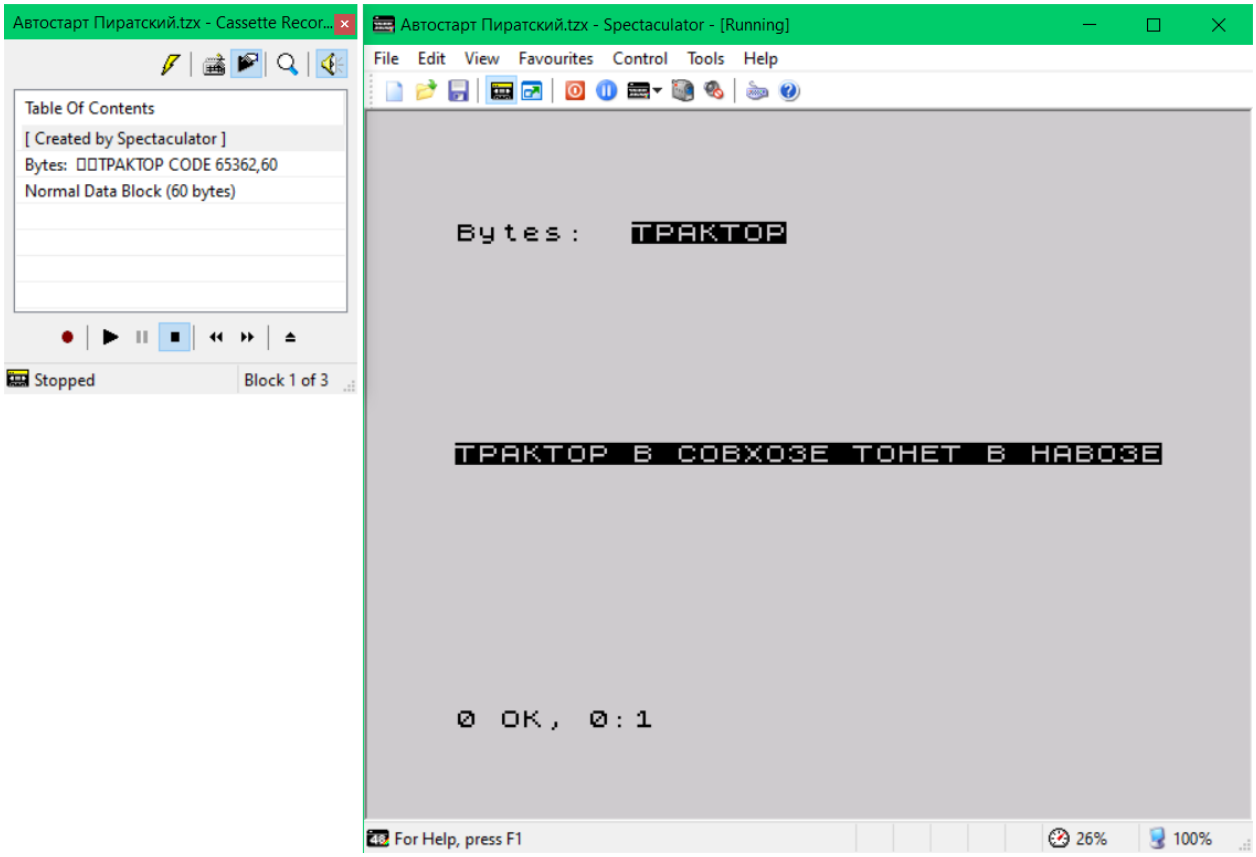


Рис. 410. Вывод текста после загрузки автостартующего блока «Bytes:» по мотивам старой книги.

Еще минус одна проблема юности. В 1996 году я бился в бессилии над этой программой из книжки, наугад перебирал разные значения, но она всё время сбрасывалась. Та самая книга, которая появилась у меня осенью 1996 года на закате эпохи Спектрума:

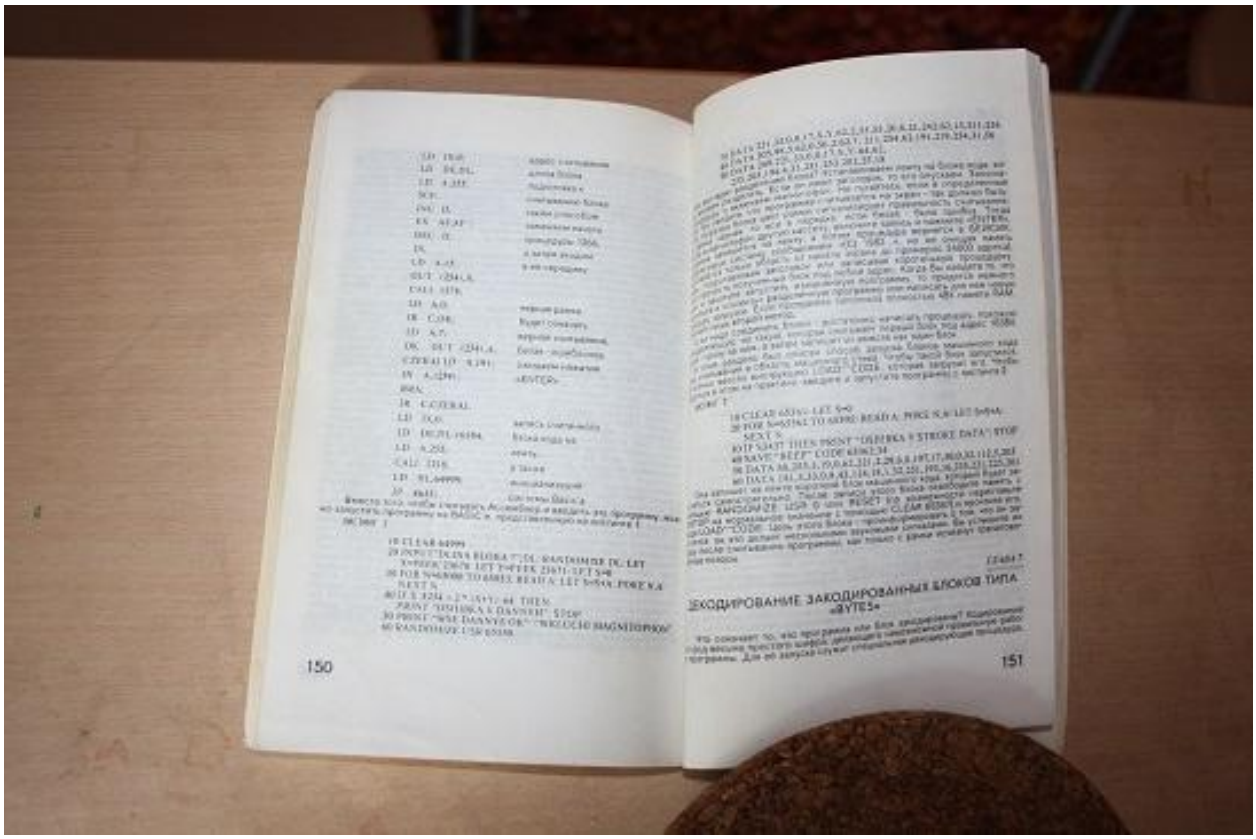


Рис. 411. Легендарная книга лежит на столе. Страница с программой автозапуска. Фото 4 мая 2017 года

Ну а уважаемому читателю будет приятно воссоздать примеры на тему родной стихии.

Глава 49
Системные автостарты. Блок «Простой»

Краткое содержание: NEWPPC, NSPPC, номера строк и операторов, автостарт

Автостартующие блоки, которые содержат в себе фрагмент области системных переменных, будут называться системными автостартами. Именно до такой конструкции я догадался в 2013 году, когда описывал легендарное «Смещение от команды RUN».

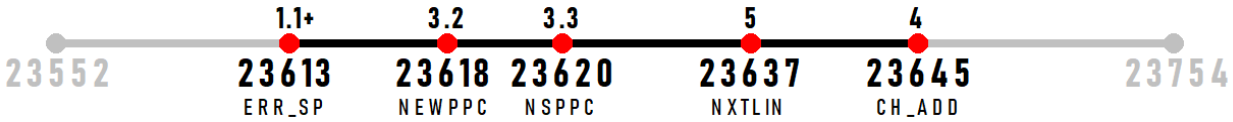


Рис. 412. Линия значимых адресов системных автостартов.

Тут ещё больший простор для творчества. Кроме привычных точек CH_ADD (23645) и NXTLIN (23637) добавляются промежуточные NSPPC (23620) и NEWPPC (23618). Заменяя значения переменных в процессе загрузки, можно перехватывать выходы по ошибкам и размещать BASIC строки в произвольных адресах, заставляя выполняться с автозапуска фрагменты и целые блоки программ.

Однако, последовательность выполнения будет частично завязана на базовых цветных точках автостарта. Поскольку программы записываются в память искусственным способом на чистый компьютер, для работоспособности придётся изменять комплекс из нескольких переменных. Например, для BASIC программы требуется отодвинуть области E_LINE, WORKSP, STKEND и STKBOT. Выше по книге это всё изучалось, поэтому к текущей странице такие дополнительные действия не должны вызвать трудностей.

Начну с самого простого и полезного автостарта. Его можно использовать для создания самозапускных блоков адаптированных игр. С комплексом точек 3.2 и 3.3 вы также хорошо знакомы по главе «*Закольцовка безномерной строки*». Предлагаю на их основе изобрести автостартующий блок:

```
New File [Автозапуск 23618-20.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 23618
23618 ← 1 0 1                ; запустить с 1-й строки 1-го оператора
23627 ← 23828
23641 ← 23829
23649 ← 23831 23831 23831
23755 ← 0 1 34 0 245 172 167 42 167 44 167 43
23767 ← 167 59 34 32 32 16 2
23774 ← НАРКОМАН РАЗНЕС
23789 ← 16 8 34 13 0 2 31 0 245 173 167 43
23801 ← 167 59 34 32 16 3
23807 ← РЕСТОРАН В АВТОВО
23824 ← 16 8 34 13 128 13 128

17996 ← 23618 0
18000 ← 3
18001 ← 19 1 16 2 65 66 84 79 66 79
18011 ← 213 23618

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

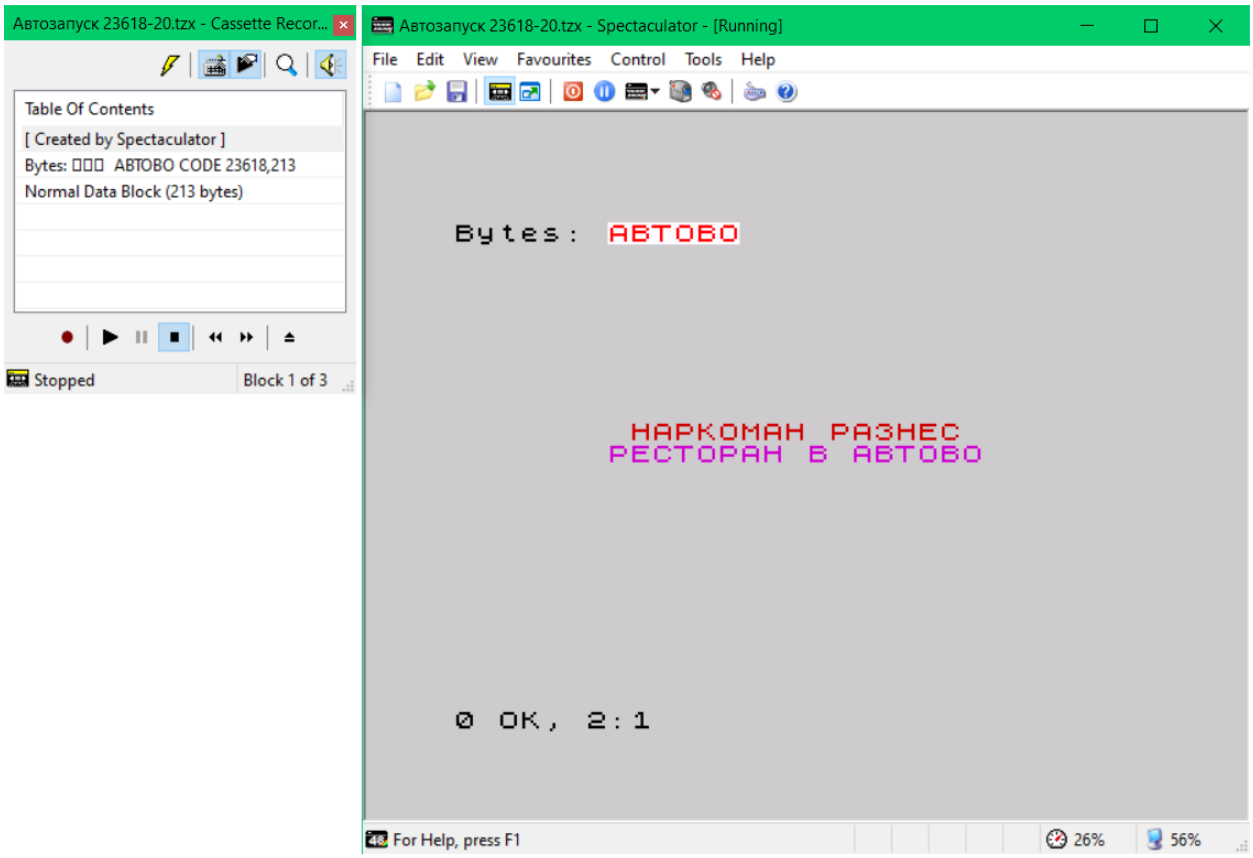


Рис. 413. Вывод текста после загрузки автостартующего блока «Bytes:».

Пусть так и называется «Простой». По адресу 7037 проверяется номер оператора `NSPPC (23620)` и если он меньше «128», то дополнительно смотрится `NEWPPC (23618)`. Поскольку во время загрузки туда записались значения 1 и 1, то вместо выхода происходит переход к выполнению программы с 1-го оператора строки №1. Обратите внимание, что для запуска не понадобилось никаких команд в буфере редактора. Синтезированная на чистый компьютер BASIC программа запустилась сама, не оставив грязи в области, на которую указывает `E_LINE (23641)`.

Позволю немного зайти вперёд. Метод этого автостарта широко используется в блоках «`Program:`». Номер строки автостарта, задаваемый по команде `LINE`, извлекается из заголовка и заносится в эти переменные. Следом идущий блок данных с BASIC программой автостартует благодаря подмене значений переменных.

Глава 50
Системный автостарт «CH_ADD» . Диана-2

Краткое содержание: CH_ADD, точка 23645

На очереди системный автостарт «CH_ADD» по ячейке 23645. На этот раз будет полная свобода действий:

```
New File [Автозапуск 23645-2.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 23508
23508 ← 58 245 34 22 10 0 19 1 18 1
23518 ←ХОХМА ХАЗАНОВА РАЗОРВЕТ ОТ СМЕХА
23550 ← 34 13
23610 ← 255      ; Подготовить код успешного выполнения OK
23611 ← 128      ; Включить тумблер 7 для отгона программы синтаксиса.
```

```
23613 ← 65364    ; Скорректировать для нормального выхода в BASIC
23620 ← 255      ; Не рассматривать операторы в НОМЕРНОЙ BASIC-строке
23645 ← 23508    ; Задать адрес продолжения выполнения новой строки

17996 ← 23508 0
18000 ← 3
18001 ← 18 1 19 1 32 67 77 69 88 32
18011 ← 139 23508

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

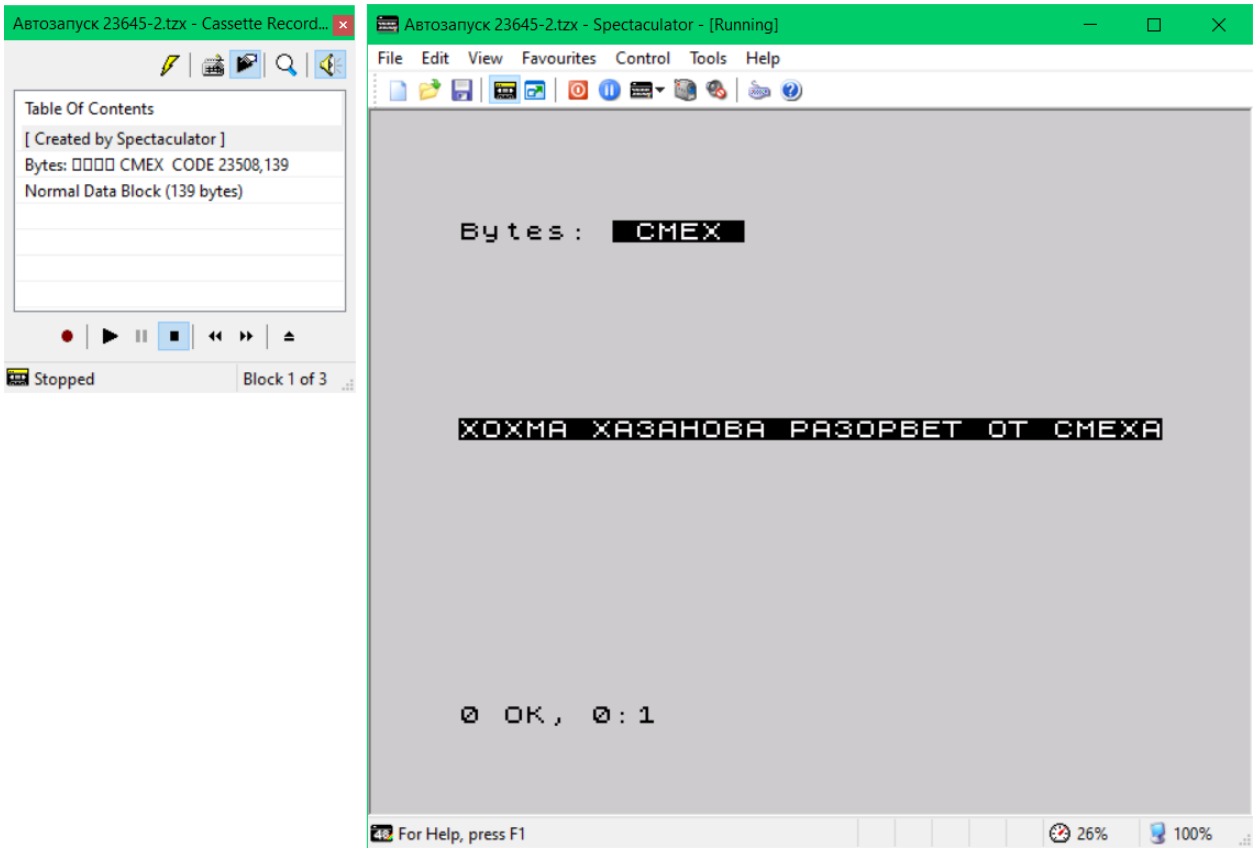


Рис. 414. Вывод текста после загрузки автостартующего блока «Bytes:».

Преимущества и недостатки этого метода хорошо видны на примере. Из недостатков это то, что приходится донастраивать парочку незапланированных переменных и грузить паразитные участки области с 23552 по 23646.

Преимуществ в разы больше. Автозапуск происходит не добравшись до области 23755. Фрагмент BASIC строки можно расположить в произвольном участке. На выбранном месте строка не искажается результатом выполнения работы и можно с самого начала, после двоеточия, ставить выполняемую программу, так как рабочие области остались на своих местах с адреса 23762.

На этот раз выставлять заглушку в `NXTLIN (23637)` не требуется. При записи с чистого компьютера она не выставлена и указывает на ячейку «00000» в которой «176», что вполне достаточно для срабатывания защиты и выхода в `MAIN-4 (4867)`.

Следующий вариант с началом в адресе 23645:

```
New File [Автозапуск 23645-1.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 23645
23645 ← 23800
23649 ← 23762 23762 23762
23761 ← 128
23800 ← 58 245 34 22 10 0 19 1 16 1
23810 ←B СНТ ОРЕХОВО ОТ ВЕТРА НЕТ СВЕТА
23842 ← 34 13

17996 ← 23645 0
18000 ← 3
18001 ← 16 1 32 79 80 69 88 79 66 79
18011 ← 199 23645

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

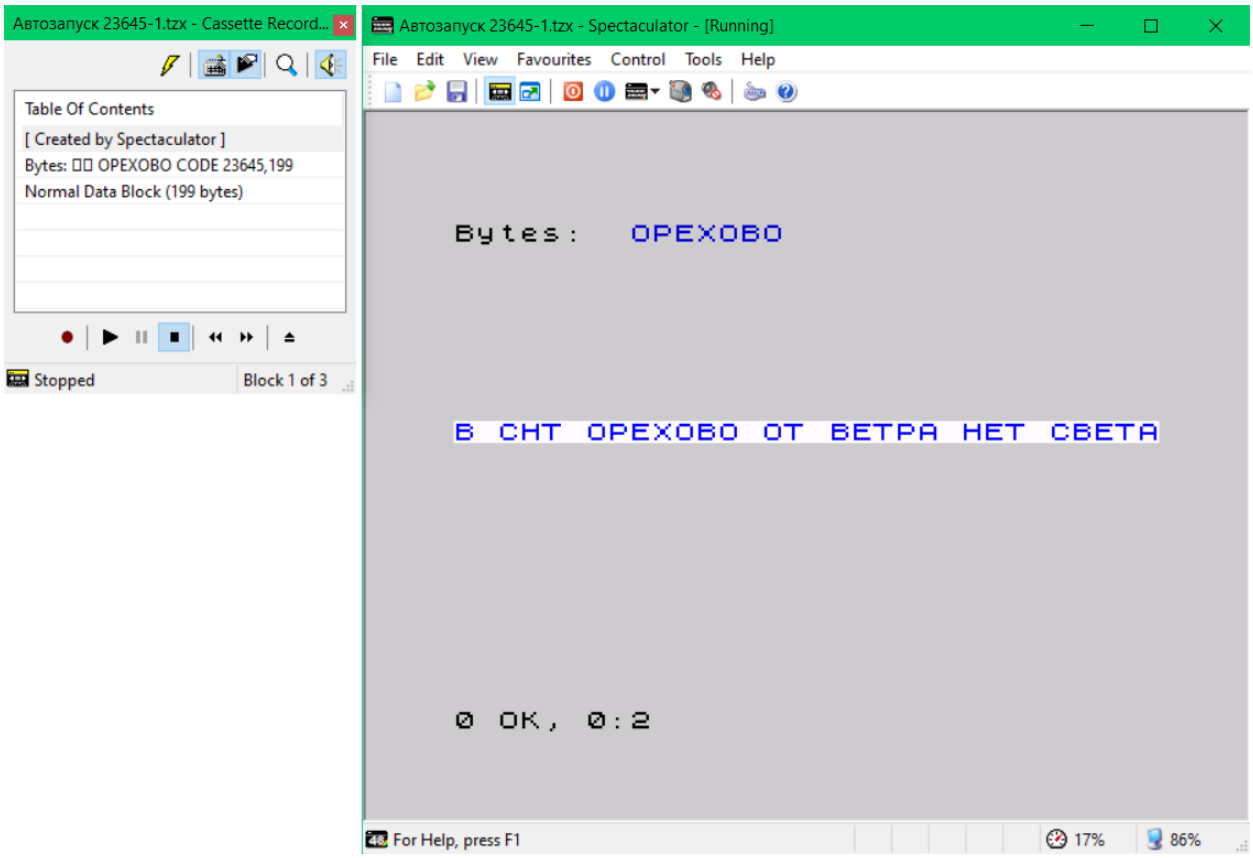


Рис. 415. Вывод текста после загрузки автостартующего блока «Bytes:».

Жаркий июльский день в южной части Приозерского района Ленинградской области. Садоводческие массивы в районе посёлка Орехово живут своей жизнью. Тётки с бабками копаются в огородах. Дети катаются на великах и тайком покуривают найденные бычки.

После полудня на небе появляется дымка. Солнце тускнеет и вскоре становится пасмурно. Внезапно небо начинает темнеть, а из-за леса появляется кусок чернущей тучи, которая каждую секунду извергает молнии. Птицы замолкают и наступает предгрозовое затишье…

Сильный порыв ветра пригибает к земле молоденькие берёзки на шестисоточных участках. В такт ветру на столбах искрятся провода. Свет померк, и весь посёлок погрузился во тьму…

Глава 51
Системный автостарт «NXTLIN». Никитос возвращается

Краткое содержание: NXTLIN, точка 23637

А теперь продвинутая версия автостарта с окончанием в указателе `NXTLIN (23637)`. Только сейчас предлагаю попробовать поместить строки в произвольной области и запустить:

```
New File [Автозапуск 23637-2.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 23470
23470 ← 0 1 35 0 245 172 167 42 167 44 167 59
23482 ←""  Вору На guckomeke прем
23507 ← 34 13 0 2 38 0 245 34 32 17 6 111
23519 ←m АНТОНОВА, СЕРОВА и АЗАРОВА
23547 ← 17 8 34 13 128
23610 ← 255
23611 ← 128
23613 ← 65364
23620 ← 255
23637 ← 23470

17996 ← 23470 0
18000 ← 3
18001 ← nonca-CCCP
18011 ← 169 23470

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

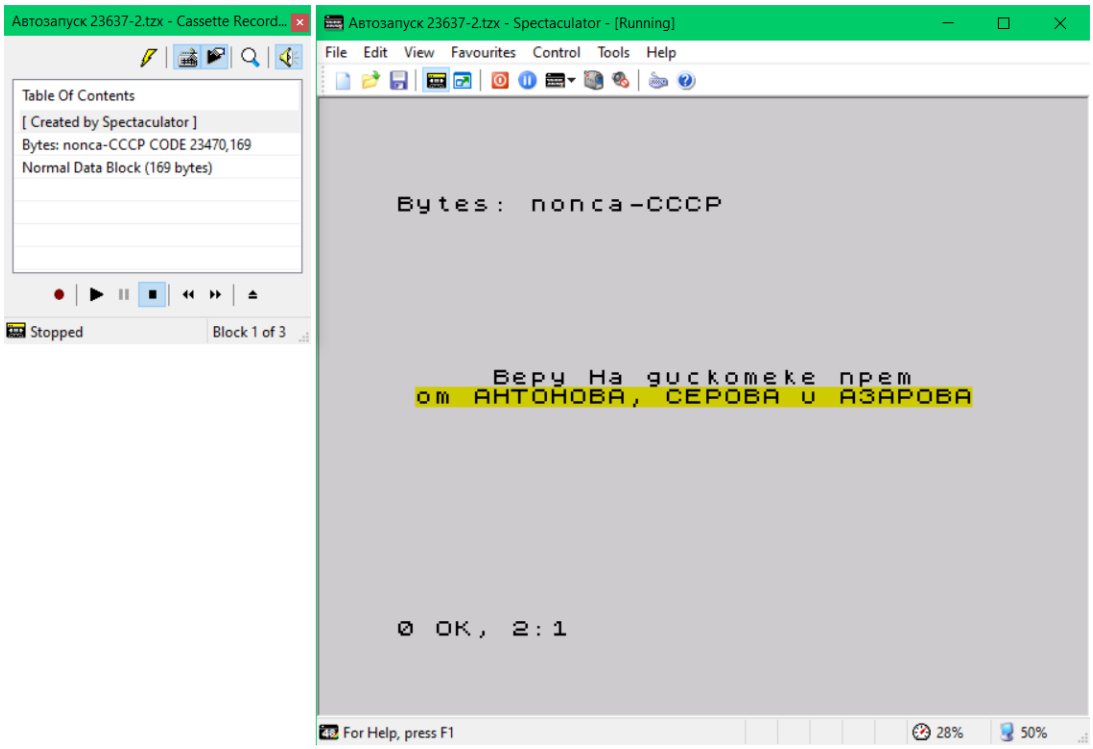


Рис. 416. Вывод текста после загрузки автостартующего блока «Bytes:».

А теперь нажмите ENTER, чтобы очистился экран и наберите:

```
POKE 23637,174: POKE 23638,91
```

Строка повторно выполнилась, а на экране снова появилось…

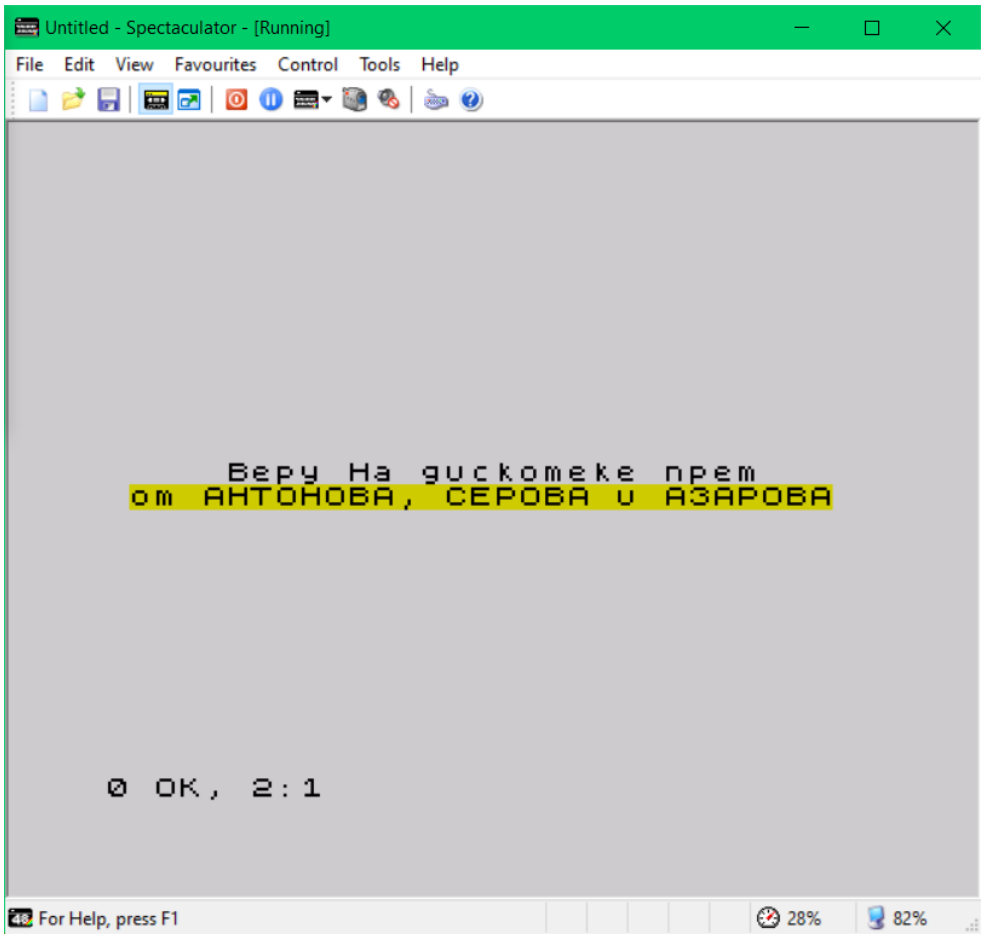


Рис. 417. Повторное выполнение строки текста из загруженного автостартующего блока.

Вновь восходит луна над моею судьбой
И идёт по извечному кругу
И покажется мне, снова шепчет прибой
То, что мы говорили друг другу...
...
А музыка венчальная
Такая беспечальная
Я холоду случайному
Её не уступлю...
...
Полной, полной луны сила!
Полной, полной луны сила!
Полной, полной луны сила!
О-о-о-о...

Всё, меня уже поперло! Вот она, сочная горбачёвская попса, которая пробирает до глубины души! Пойду послушаю музон, и обязательно вернусь со свежими идеями:

Рис. 418. WinAmp. Подборка песен Юрия Антонова, Александра Серова и Игоря Азарова.

Возвращаюсь с музыкальной паузы и сходу предлагаю версию с началом в точке 23637:

```
New File [Автозапуск 23637-1.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 23637
23637 ← 23800
```

```
23645 ← 23756
23800 ← 0 0 29 0 245 172 167 42 167 44 185 187
23812 ← 167 43 167 59
23816 ←""ЗАВХОЗ СЕМЕНОВ
23831 ← 34 13 0 1 32 0 245 34 32 32 32 32
23843 ←Воруем Tucku С ВЕРСТАКОВ
23867 ← 34 13 128

17996 ← 23637 0
18000 ← 3
18001 ← 19 1 16 1 51 65 66 88 79 51
18011 ← 233 23637

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

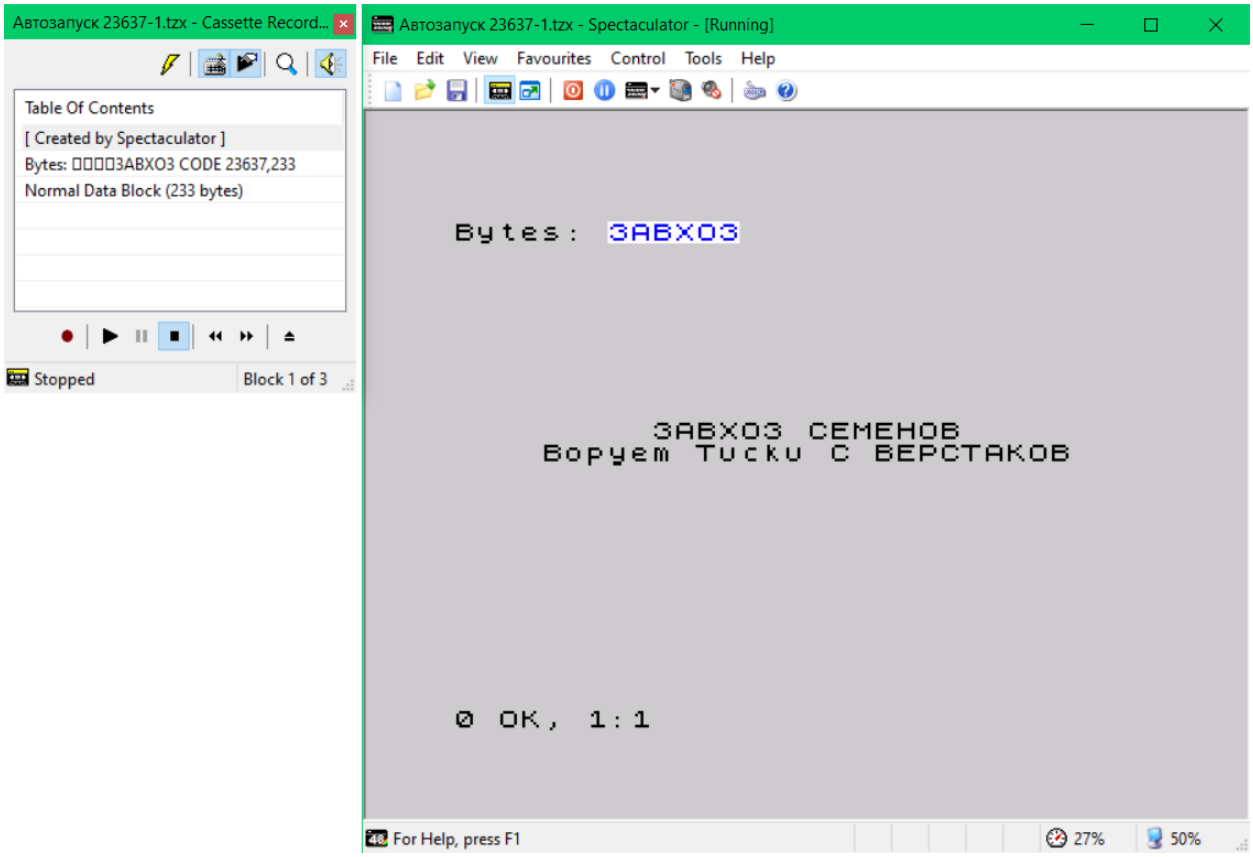


Рис. 419. Вывод текста после загрузки автостартующего блока «Bytes:».

Тут уже чуть проще, поскольку имеется возможность заглушить указатель CH_ADD нужным значением и заодно очистить BASIC область, путем загрузки.

А завхоз, он и в Африке завхоз. Одно время сам работал завхозом, поэтому знаю все тонкости этой должности.

Глава 52
HeZX-Forum. Сюжеты загрузки блоков данных

Краткое содержание: письма читателей, маршруты типов загрузки

КСЗ: А письма от благодарных читателей продолжают поступать косяками ~~и папиросами~~ в огромном количестве. Мы рады, что тема автозагрузки нашла отклик у нашей многомиллиардной аудитории подписчиков. И вот еще один корреспондент из посёлка Балаболово затронул жизненно важную проблему. Собственно, он усомнился в некоторых моментах, и допускает, что существует более ранняя точка, которая может быть задействована для автостартов. Приводим текст его лаконичного письма

КОРР: С удовольствием прочитал прошлые главы. Всё чики-пуки, но у меня закрались сомнения. По-моему, вы что-то недоговариваете. Ведь по клавише BREAK можно выйти раньше, чем сработает первая точка автостарта.

КСЗ: К сожалению, имя и отчество читателя было неразборчиво написано, поэтому заранее приносим извинения. Уважаемый гражданин Пургогонов П. Д. Вы не поверите, но автостартов раньше красной точки не существует. Дорожки аварийных выходов в процессе загрузки всё равно пролегают через точку 65358/59 и ранее нигде не соприкасаются с памятью в диапазоне 23755-65535. Учитывая сложившуюся ситуацию, придётся этому моменту посвятить главу, чтобы окончательно разобраться в данном вопросе.

Представьте себе такую картину. Стрелочка попадает на адрес 1516. Значение SP-столбика в этот момент 64354. До окончания загрузки остаётся дождаться и принять один неделимый сигнал, сделав полтора круга по программе ловли битов. Пусть это будет последней общий точкой, с которой дальнейшая судьба загрузки будет протекать по одному из 4-х вариантов или сюжетов:

1) Нажатие «BREAK» во время загрузки с последующей выдачей сообщения «D BREAK – CONT repeats»

2) Внезапный обрыв сигнала при остановке воспроизведения с выдачей сообщения «R Tape Loading Error»

3) Несовпадение псевдоконтрольной суммы после загрузки с выдачей сообщения «R Tape Loading Error»

4) Успешная загрузка с выдачей сообщения «0 OK»

Все остальные ситуации, вроде несовпадения количества данных, которые заявлялись в заголовке, с реально загруженными, являются частным случаем вариантов №2 и №3.

Вариант первый: Внезапно нажимается клавиша BREAK. Вместо сигнала данных из порта приходит значение нажатия клавиши «127». Галочка «C» не выставляется ☐. Распознав отмену загрузки, Стрелочка прерывает ловлю сигнала и немедленно начинает каскадный выход, собирая по дороге все попавшиеся команды «RET NC»: (1524 RET NC (65354) → 1510 RET NC (65356) → 1485 RET NC (65358)). Следующим шагом она попадает в навязанную подпрограмму SA/LD-RET (1343), где после прерванной загрузки начинает приводить в порядок атрибуты экрана.

Поскольку для компьютера с момента прохода по этим командам прошло много времени, а для человека тысячные доли секунды. То палец по инерции еще держит зажатой клавишу BREAK, а в это время по адресу 1354 происходит повторный опрос и из порта 254 снова прилетает «127». В 1362 выбирается сообщение №12 и далее по команде «RST 8», Стрелочка попадает в самый верх на адрес 00008. Там происходит запись кода сообщения в ERR_NR (23610), после чего командой «LD SP,(23613)» срезаются все лишние значения SP-столбика, оставляя пенёк из одного значения для выхода. Из 5843, по этому значению, происходит возврат на бирюзовую точку в MAIN-4 (4867). Там происходит вывод сообщения «D BREAK – CONT repeats».

Внезапный обрыв сигнала будет вторым вариантом. Согласно заявлениям, в заголовке ещё остаются несчитанные байты, а по факту в процессе ловли очередного бита

уже стоит мёртвая тишина. Стрелочка продолжает ждать, но однажды её терпение лопается. Не дождавшись продолжения загрузки, она выставляет Z ☑ галочку «Z». По команде «RET Z», с точки 1518, начинается каскадный выход. На этот раз в подпрограмму SA/LD-RET (1343) она попадает по пути: 1518 RET Z (65354) → 1510 RET NC (65356) → 1485 RET NC (65358).

Корректно выполнив подпрограмму, Стрелочка выходит по синей точке автостарта на LD-BLOCK (2053) следующей же командой «RST 8» она отправляется по адресу 00008, где подготавливает к выводу сообщение: «R Tape Loading Error».

Третий сюжет с ошибкой контрольной суммы может произойти в двух случаях:

– Когда заявленное количество данных в заголовке кончилось, а сигналы продолжают поступать.

– Когда количество заявленных байт совпало с реально загруженными из блока данных, но по каким-то причинам подсчитанная псевдоконтрольная сумма не совпала с последним проверочным байтом.

В этом случае цикл ловли битов успешно завершается, однако по адресу 1503-1504 в результате проверки снимается C ☐ галочка «C». Минуя красную и синюю точки выхода, Стрелочка снова попадает на команду «RST 8» и проходит путь, аналогичный прошлому варианту. Всё это точно также оканчивается сообщением «R Tape loading error».

Последний вариант сценария правильного считывания обсуждается на протяжении последних глав книги, и вы его хорошо знаете. С установленной C ☑ галочкой «C» происходит возврат через все цветные точки автостарта.

Чтобы понять взаимодействие этих сценариев, достаточно представить их в виде маршрутов, а в ключевых местах поставить точки и наложить на рисунок:

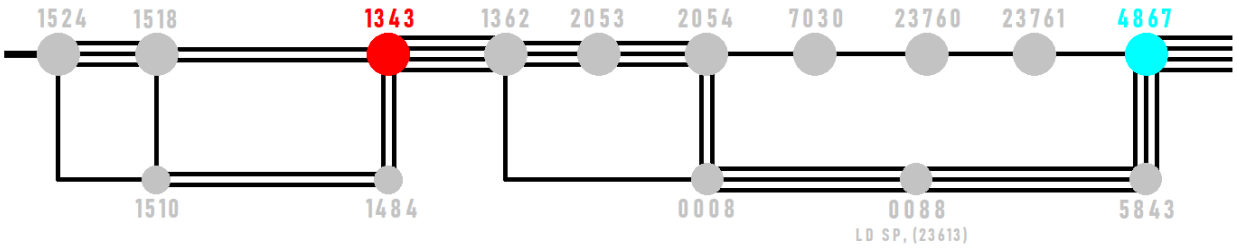


Рис. 420. Схема пересечения путей сценариев загрузки.

Наглядно видно, что все четыре варианта, в какой-то момент пересекутся в первой красной точке (1343), потом разойдутся (1362) и снова встретятся в последней бирюзовой (4867).

В любом случае адрес 1343 будет первой точкой соприкосновения, причём во всех четырёх сценариях. Как известно из главы «*Путешествие с командой LOAD*» ячейки 65354-65357 непригодны для записи значений, потому что после каждого принятого бита переписываются.

Естественно, при таком раскладе любой автостарт по прерыванию записи сработает, если для этого успеют загрузиться новые данные. Поэтому создание повреждённых блоков имеет смысл с началом в точке перехвата.

Глава 53
Дефектоскопические автостарты «ERR_SP»

Краткое содержание: ERR_SP, точка 23613

Вот и дошла очередь до переменной ERR_SP (23613). Автостартующие блоки на её основе будут универсальными и сработают от первой встретившейся команды перехода «RST 8». Пробежав по маршруту 8 → 14 → 83. в адресе 88 Стрелочка встретит команду «LD SP,(23613)». Ничего не подозревая, она переставит SP-столбик в место, которое вы зададите и попытается дальше выполнять рутинную программу вывода ошибки по адресу 5829. Добравшись до адреса 5843, Стрелочка встречает «RET» и попадает… совсем не туда, куда она думала (4867), а по адресу, записанному в основании нового SP-столбика.

Почему этот автостарт не имеет фиксированной очереди? Да потому что в зависимости от типа ошибки это может быть любая промежуточная точка, начиная от 1.1 до условной 5.9. Автостарт сработает как от имитированного повреждения записи, так и от случайной ошибки в BASIC строке. Если включить фантазию, выбор вариантов достаточно разнообразный.

Вопреки всем предрассудкам, серия системных автостартов «ERR_SP», будет одна из самых простых. Вариант с началом в точке 23613 получится таким:

```
Reset
New File [Автозапуск 23613-1.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 23613
23613 ← 23770
23768 ← 0 23772
23772 ← 253 54 0 255 49 84 255 237 115 61 92 241
23784 ← 253 203 2 134 17 248 92 1 32 0 205 60 32
23797 ← 195 3 19 22 11 5 19 1 17 6 16 2
23809 ← В КАНАВЕ КТО-ТО КВАКАЕТ

17996 ← 23613 0
18000 ← 3
18001 ← 16 2 17 6 75 66 65 75 69 80
18011 ← 219 23613

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

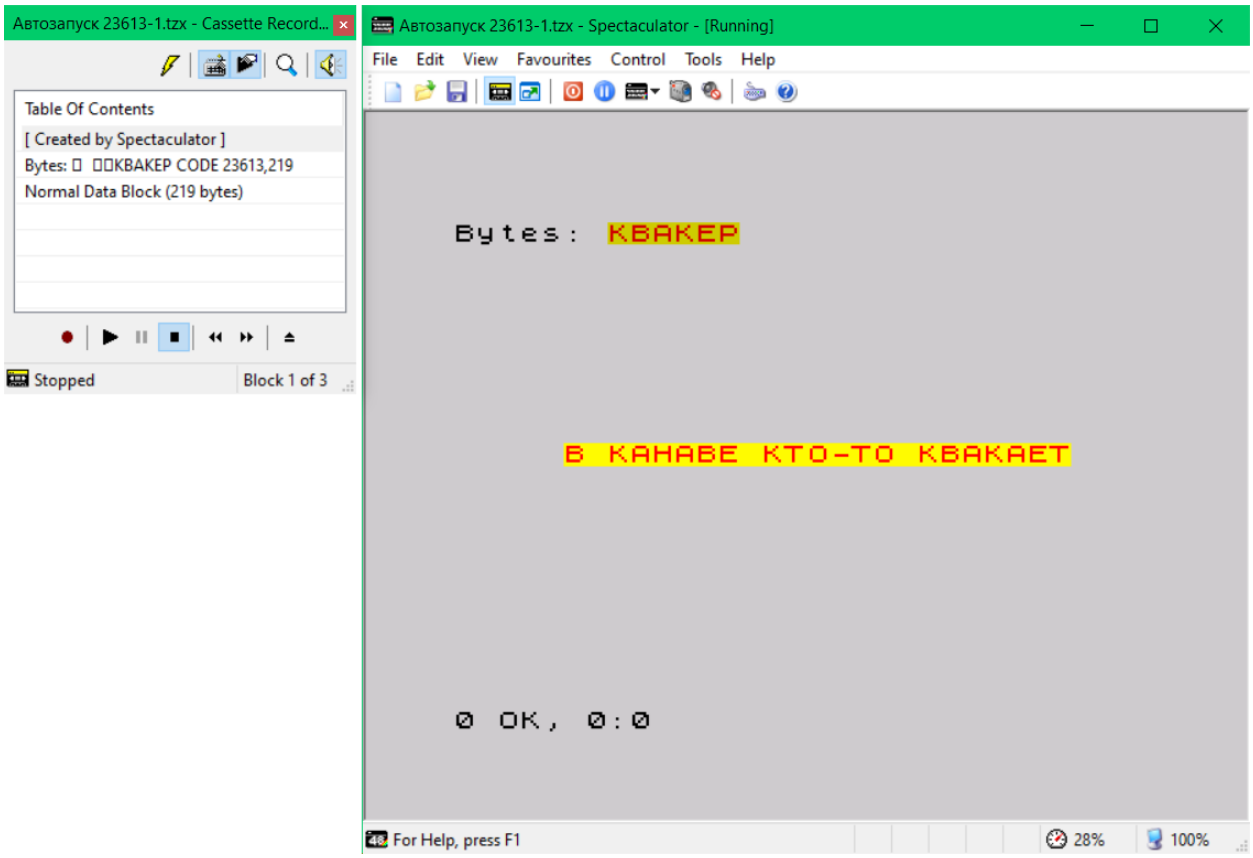


Рис. 421. Вывод текста после загрузки автостартующего блока «Bytes:».

А кто обычно квакает в канавах? Ответ на этот вопрос знает любой программист-системщик:

предоставит новая машина. А возможности немалые. Если Вы программист-системщик, то Вам не надо объяснять, что такое электронный квазидиск на 80 Кбайт; если Вы пишете программы на Basic'е, то, вероятно, представляете

Рис. 422.

Конечно же электронные квази диски! Да не простые, а на целых 80 килобайт! Восемьдесят, сука, килобайт! Обычно они так протяжно и по-взрослому квакают: «Ква-а-а-а-а-а-а, ква-а-а-а-а-а-а-а». Кто ходил по переулку Кваренги в Петербурге, тот просечёт тему и поймёт, о чём я. Вот квази-головастики, объёмом несколько байт, умеют только пищать. Но их можно услышать на газонах Смольного парка только в период размножения квази-дисков и только в переулке Кваренги.

В алгоритм я специально ввёл команду «*Reset*». Если вы набирали программу на голый компьютер, то ошибку для активации даже не придётся выдумывать. Она уже закралась и не одна. Поскольку первого запуска BASIC не производилось, указатель CH_ADD еще не выставлялся и равен нулю. А в нулевом адресе стоит команда «DI» с кодом «243». Если для NXTLIN она вполне сойдёт, то для CH_ADD на момент проверки требуются коды «13» или «58».

Точно также не получили защитные заглушки «пиздецовые» или «ППЦ» ячейки выполняемых строк и операторов 23618-23623 (NEWPPC, NSPPC, PPC, SUBPPC). После сброса они все занулены.

Запускается программа следующим образом: выполняя знакомый каскадный выход, Стрелочка выходит на точку STMT-R-1 по адресу 7037. Увидев в NSPPC (23620) отсутствие 7-го бита, происходит дополнительная проверка переменной NEWPPC (23618) на тот же 7-й бит. А там конечно по нулям, Стрелочка думает, что нужно выполнить строку 0 с оператора 0 и проваливается в LINE-NEW (7070) на выполнение

программы. Увидев сплошную пустоту и безнадёгу по вычисленному адресу, Стрелочка разочаровывается в указанной строке, проходит в REPORT-0 по адресу7088 и ловит «RST 8» с кодом сообщения «0 OK.». Попав по адресу 8, Стрелочка пытается его вывести, но на экране появляется совершенно другой текст, и только следом за ним печатается планируемое «0 OK.».

А сейчас будет более интересный вариант с окончанием в точке 23613/14. Тут рычагов давления через переменные нет, поэтому придётся вспоминать прошлую главу. Автостарт с окончанием ERR_SP предлагаю сделать на базе блока с эффектом сбитой контрольной суммы или недогруза. Для создания такого блока нужно, чтобы количество данных, заявленное в заголовке, было меньше, чем сам блок данных. Таким образом, после отмеренного количества, считывание прекратится. Вместо заключительного невидимого байта контрольной суммы, для сверки возьмется последний считанный. Возникнет рассинхронизация, после чего произойдёт попытка выдачи сообщения «R Tape loading error». На этом месте произойдёт перехват и выполнится записанная программа. Естественно, программа будет полностью дописана, просто для ошибки придётся накинуть один лишний байт, в который запишется контрольная сумма, но до считывания очередь не дойдёт.

Блоки такого типа можно создать двумя вариантами: записать традиционным способом, но после записи отредактировать последний байт файла внешним редактором «*Hex Editor*» или «*Tapir*».

Второй вариант наиболее интересен в рамках данной книги, потому что не требует вмешательство сторонних программ. По этому методу открывается файл для записи и сначала записывается заголовок с требуемыми параметрами. Следом после сброса к нему дописывается блок данных, после чего файл укомплектуется и его можно закрывать.

Внимательно наберите и выполните следующий алгоритм:

```
New File [Автозапуск 23613-2.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 18000
18000 ← 3
18001 ← 32 17 5 67 79 67 72 79 66 79
18011 ← 126 23489

AF ← 0
DE ← 17
IX ← 18000
SP ← 17996
PC ← 1218
Trace

Debugger
Go To 23489
23489 ← 0 23493
23493 ← 253 54 0 255 49 84 255 237 115 61 92 241
23505 ← 253 203 2 134 17 225 91 1 31 0 205 60 32
23518 ← 195 3 19 22 11 5 19 1 17 5 16 2
23530 ←A B СОСНОВО МНЕ ХЕРОВО
23613 ← 23491

AF ← 65280
DE ← 127
IX ← 23489
SP ← 17996
```

```
PC ← 1218
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

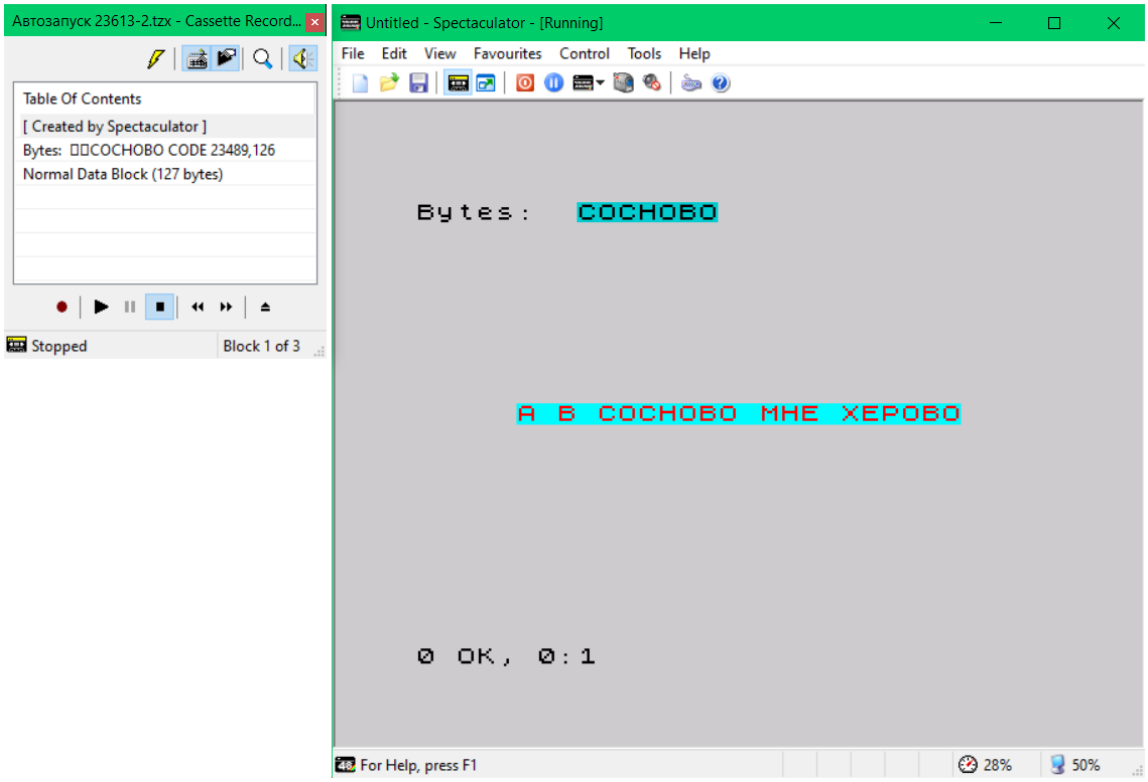


Рис. 423. Вывод текста после загрузки автостартующего блока «Bytes:».

Ну и версия автозапуска с недогрузкой по внезапному обрыву потока данных. Это получается, когда в заголовке заявлено 127 байт, а в блоке их всего 126:

```
New File [Автозапуск 23613-3.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 18000
18000 ← 3
18001 ← 19 1 16 7 17 0 75 79 75 67
18011 ← 127 23489

AF ← 0
DE ← 17
IX ← 18000
SP ← 17996
PC ← 1218
Trace

Debugger
Go To 23489
23489 ← 0 23493
23493 ← 253 54 0 255 49 84 255 237 115 61 92 241
23505 ← 253 203 2 134 17 225 91 1 31 0 205 60 32
23518 ← 195 3 19 22 11 5 19 1 17 0 16 7
23530 ← nempa mopkaem om kokca
```

```
23613 ← 23491

AF ← 65280
IX ← 23489
DE ← 126
SP ← 17996
PC ← 1218
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

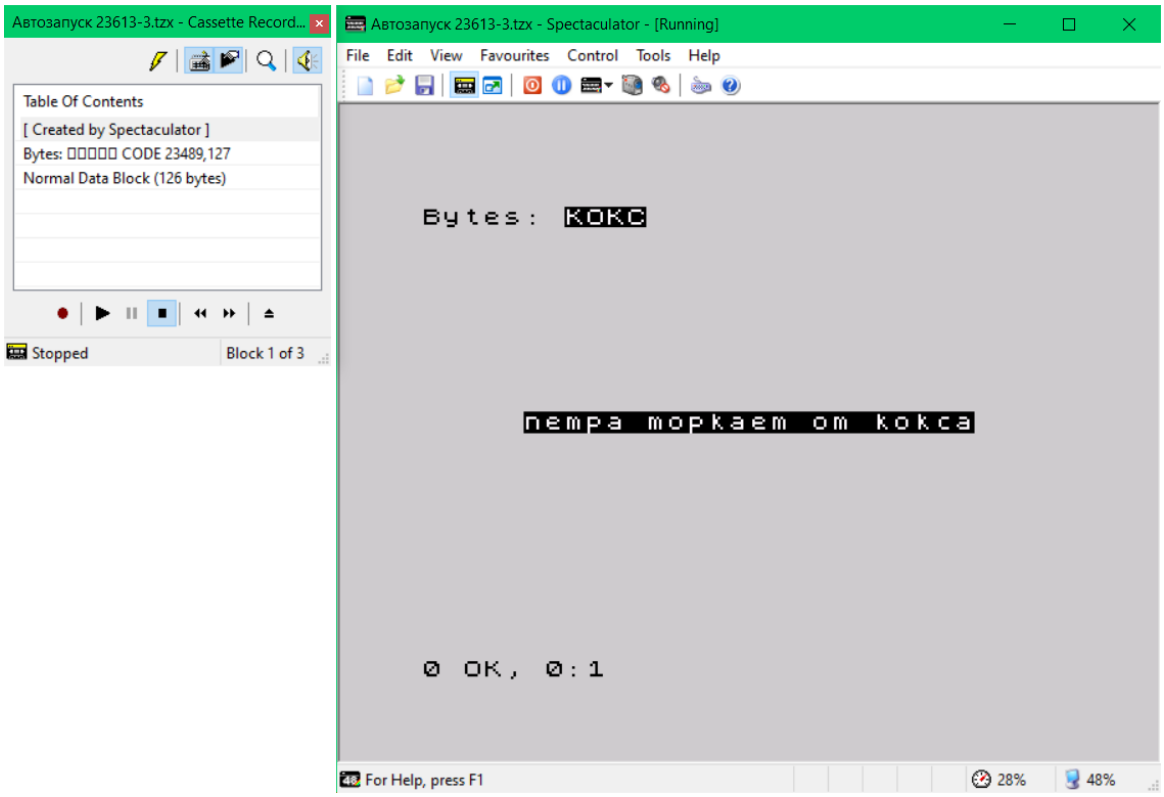


Рис. 424. Вывод текста после загрузки автостартующего блока «Bytes:».

В небольшой посёлок городского типа пришла весна. На солнышке уже совсем припекает и можно ходить в легкой куртке без свитера. В эти тёплые майские дни земля успела чуть подсохнуть после растаявшего снега. На обочине дороги, из покрывала сухой прошлогодней травы, торчат первые зелёные росточки. На берёзах выросли сережки, а из лопнувших почек пробились первые клейкие листочки. За панельным трёхэтажным домом 121-й гатчинской серии раздаются детские голоса и скрип старых железных качелей.

Неподалёку виднеется поселковая котельная. Из чёрной металлической трубы валит густой дым. После полудня откроется баня. В воздухе пахнет свежей листвой с лёгким ароматом угольного дыма. По дороге, в сторону полей проехал синий трактор «Беларус», поднимая с дороги клубы первой весенней пыли.

Возле белой кирпичной стены котельной лежат кучи серого кокса. Над входом в здание висит старенький светильник НСП02 со ржавыми застёжками. Возле двери стоит мужик неопределённого возраста в фуфайке. Пётр Иваныч работает оператором котельной пару десятков лет. Наслаждаясь весенним солнышком и запахом сгоревшего угля, он стоит рядом со своим стареньким «Жигулём» и смотрит на кучу шлака. Жмурясь от весеннего солнышка, на вершине кучи сидит чёрный кот с большими зелёными глазами.

– Кокс, иди обедать! – позвал кота Пётр Иваныч, продолжая отдыхать в свой законный обеденный перерыв…

Котики всегда против наркотиков, а любовь и добро должно победить. А в реальности – как карта ляжет. Вот такие дела.

Глава 54
Грязный и чистый метод записи

Краткое содержание: E_LINE, буферный автозапуск, след программ

Подводя итоги автозагрузки, и цвета, которого должен быть кокс, предлагаю сотый раз вернуться к рубрике «Возвращаясь к напечатанному». Добравшись до этой страницы книги, вы догадываетесь, что команда SAVE оставляет после себя следы. Где-то переключилась переменная, а в области буфера редактора и вовсе остаются незатёртые ошмётки работы предыдущих команд. Записывая длинный блок данных через системные переменные и SP-башенку, вы сохраняете весь этот мусор себе в создаваемую программу. В настоящее время такой тип записи данных получил название «грязный».

В 2013 году, я об этом не задумывался, поэтому все примеры, и даже пару адаптированных игр, записал именно таким способом. Но прогресс не стоит на месте. В процессе создания этой книги, технологии шагнули далеко вперёд. Все современные примеры были искусственно синтезированы из отладчика прямым обращением в «телепортационную щель» с внешним алгоритмом, поэтому в записях не оставалось никаких следов вспомогательной работы. Этот тип записи программ получил название «чистый».

Предлагаю вернуться к тому самому примеру со «Смещением от команды RUN» и искусственно синтезировать запись «полугрязным методом». Программа будет запускаться командой RUN из буфера редактора, но без намёка на остатки команды SAVE. Причём строка RUN [ENTER] будет искусственно синтезирована, а при загрузке она сама активируется и запустит BASIC программу:

```
New File [Автозапуск 23627.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 23627
23627 ← 23807
23641 ← 23808 23808 23808
23649 ← 23812 23812 23812
23755 ← 0 0 48 0 245 172 167 42 167 44
23765 ← 195 167 59 34 16 1
23771 ← РАКЕТА НЕСЕТ АСТРОНАВТА В КОСМОС
23803 ← 16 8 34 13 128 58 247 13 128

17996 ← 23627 0
18000 ← 3
18001 ← 32 65 67 84 80 79 72 65 66 84
18011 ← 185 23627

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""CODE ENTER
Cassette Recorder [Play]
```

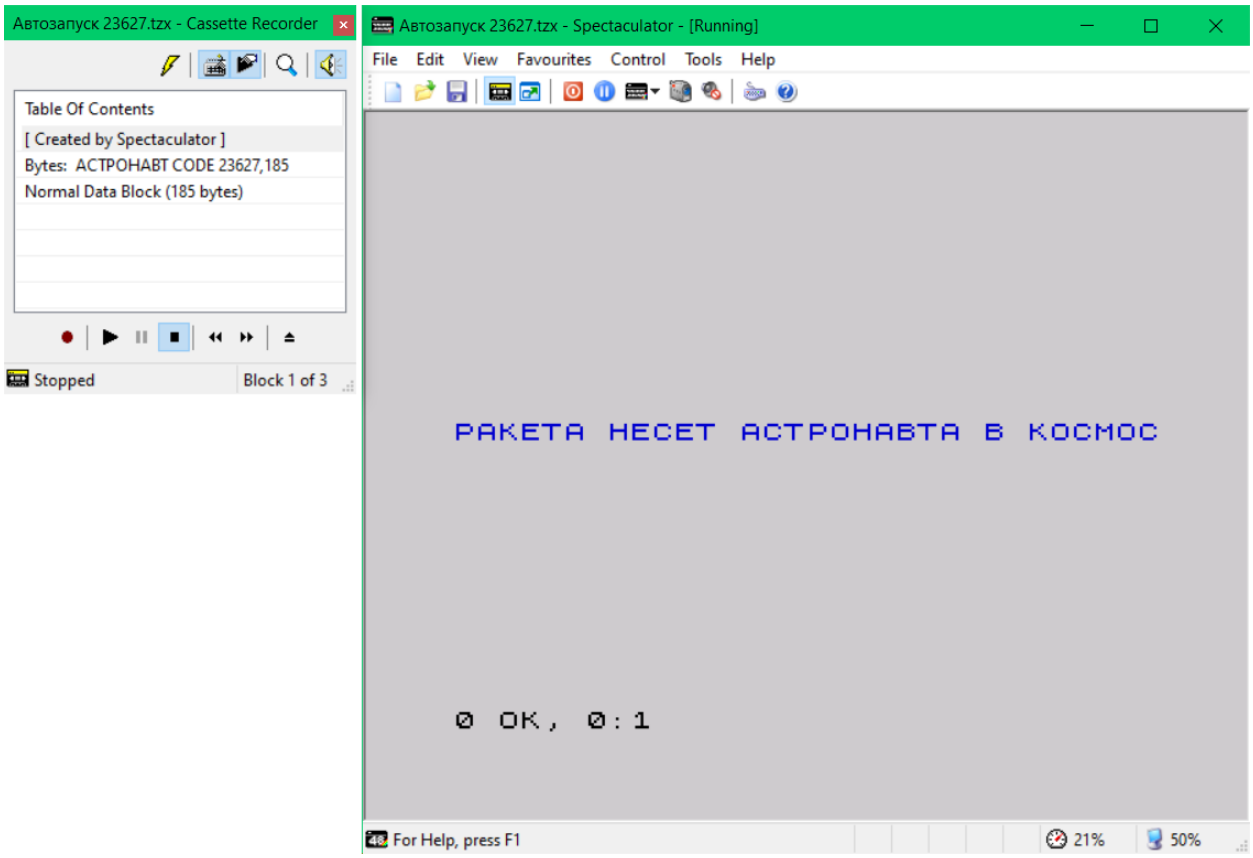


Рис. 425.

Врубаем рок - летит пилот, летит пилот...

Глава 55
Знакомство с «Program:»

Краткое содержание: блок Program:

Свершилось! Затянувшееся на 125 страниц общее и поверхностное знакомство с записью и загрузкой «`Bytes:`» наконец-то закончилось! Можно переходить к блокам из набора «`Program:`» и посмотреть на них свежим взглядом. Общие моменты, изученные ранее при загрузке «`Bytes:`», рассматривать не имеет смысла. Буду акцентировать внимание на новых моментах.

После ввода `LOAD ""` с адреса 1541 происходит стандартное формирование теоретического заголовка в области [WORKSP]+0. В `SA-TYPE-0 (1850)` начинается подготовка параметров и заполнение теоретического заголовка:

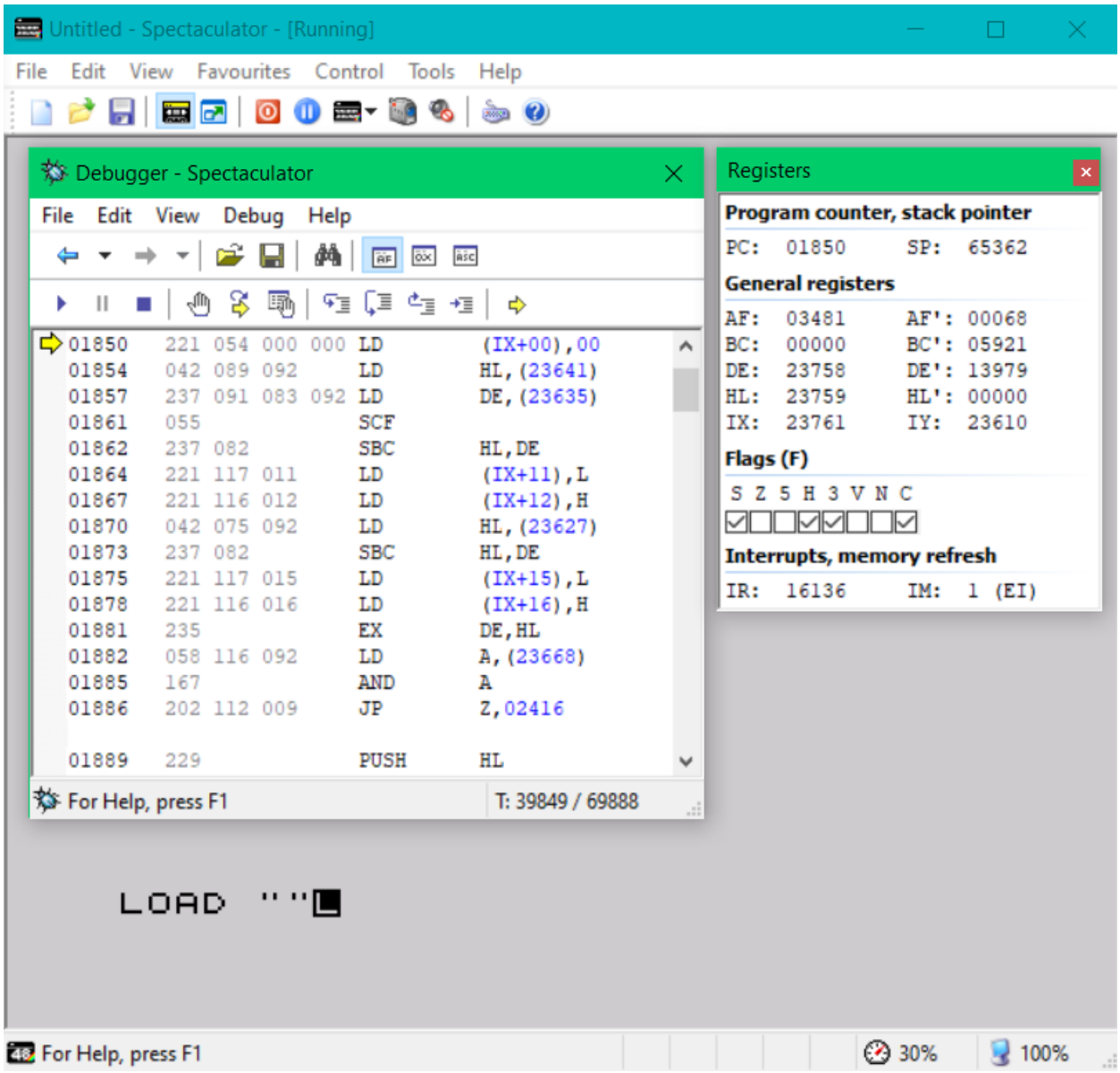


Рис. 426. Заполнение теоретического заголовка.

Первым делом устанавливается «00» – код заголовка «Program:». Из E_LINE (23641) берётся адрес буфера строки, он же конец области переменных, а из PROG (23635) место, куда загружать программу. Вычитанием указателей [E_LINE] - [PROG] узнаётся, общая длина программы с дружественной областью VARS. Результат записывается в 12-ю и 13-ю ячейки шаблона теоретического заголовка. Поскольку LOAD "" не имеет параметров, то на данном этапе в параметр длины записывается пустота.

Точно также, вычитанием VARS из PROG узнаётся длина программы без переменных и записывается в ячейки 16 и 17 теоретического заголовка.

С адреса1889 начинается общая подготовка к считыванию заголовка всех видов команд и готовится место для реально считанного заголовка. После считывания и вывода его на экран, в точке 1975, пути разных типов загрузок снова начинают расходиться:

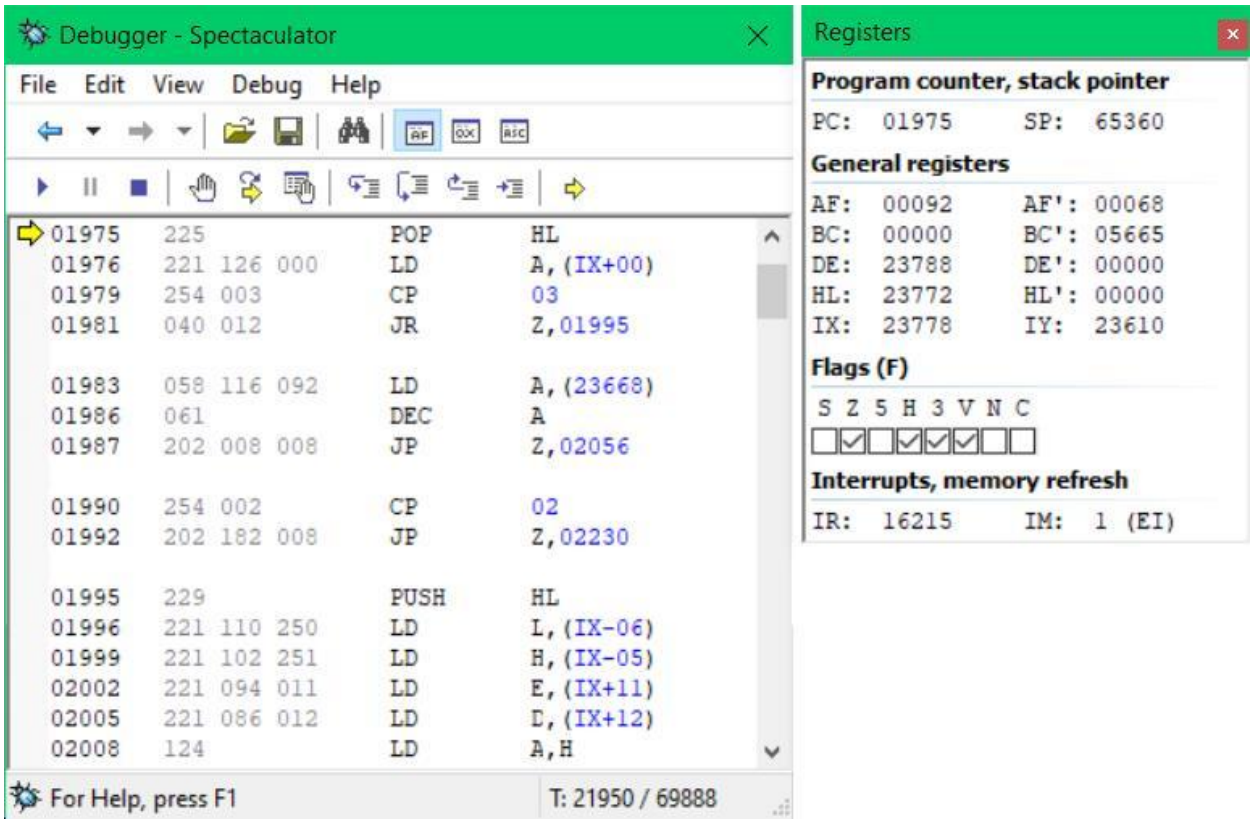


Рис. 427. Точка расхождения путей разных типов записи.

Первыми уходят блоки «`Bytes:`» (`1981 JR Z,1995`). Следом, по остаткам от команд из таблицы синтаксиса (`T-ADDR, 23668`), вычисляется команда `LOAD`. Ее дорожка отсоединяется в (`1987 JP Z,2056`) и следом отваливается команда `MERGE` (`1992 JP Z,2230`)

В 2056 начинается программа обслуживания `LOAD ""` под названием `LD-CONTRL:`

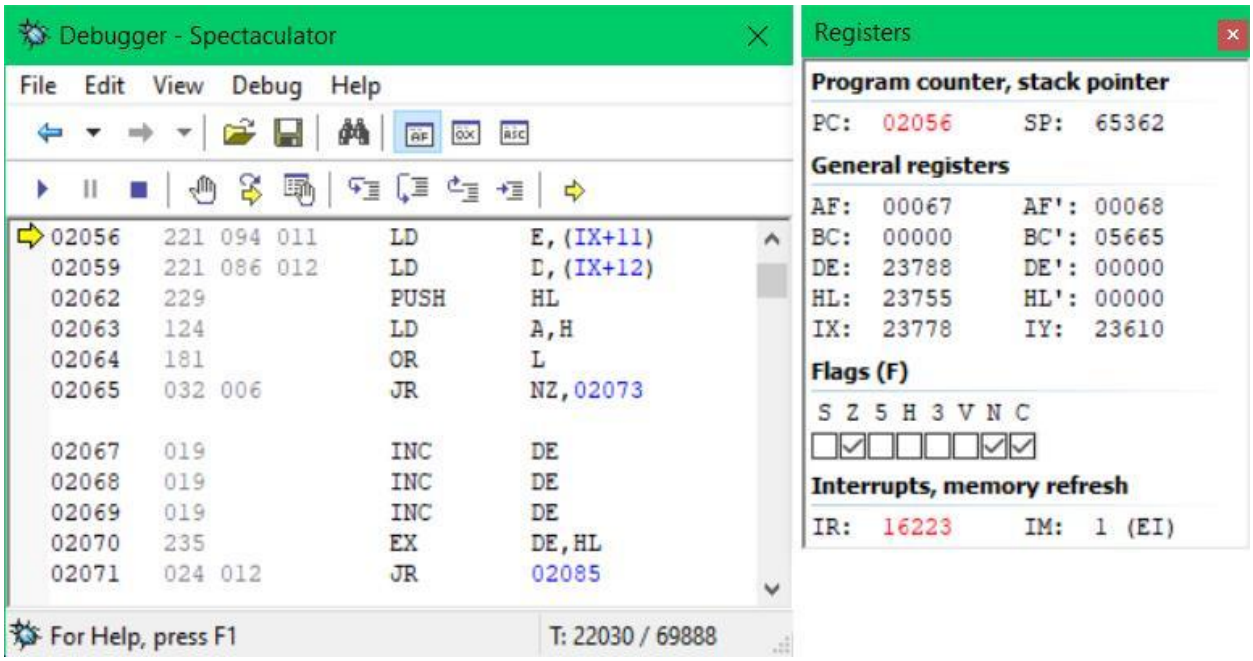


Рис. 428. Начало подпрограммы LD-CONTRL.

Следом в программе `LD-CONT-2 (2085)` начинается первая проверка места для будущего загружаемого блока данных. Накидывается 5 байт в запас и суммируется с

длиной будущего блока данных. Полученный результат копируется в «BC» и с 2091 Стрелочка идёт на трёхступенчатую программу проверки памяти `TEST-ROOM (7941)`:

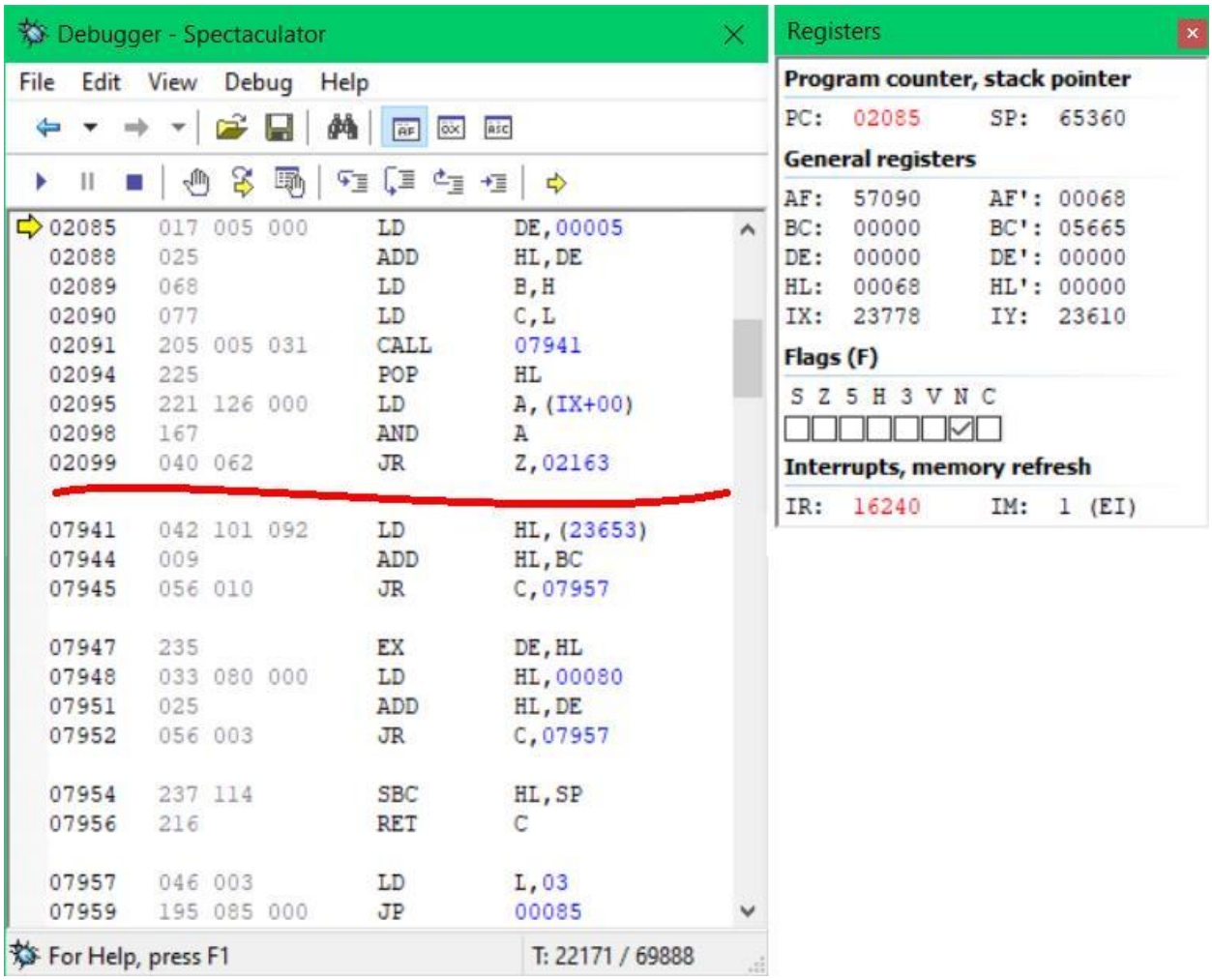


Рис. 429. Подпрограмма TEST-ROOM.

В самой подпрограмме продолжается подготовка к проверке. На первой ступени берётся значение `STKEND (23653)`, последнего из группы указателей забейсиковых областей, складывается с длиной будущей программы и накинутым 5-ти байтным хвостиком. Если при сложении сумма уже перелилась через предел 65535 (около 41737 байт), то устанавливается C☑ галочка «C». Прервав подготовку к записи, Стрелочка идёт выводить сообщение об ошибке «`4 Out of memory`» (`7945 JR,C 7957`).

Ну а если это не предел, то проверка продолжается и начинается второй акт. На этот раз берётся значение грубой прикидки, которое получилось в первом круге проверки и добавляется еще 80 байт для предполагаемой максимально возможной длины SP-столбика.

Если при сложении сумма перевалила через предел 65535 (около 41657 байт), то устанавливается C☑ галочка «C», и к записи можно не приступать. Как и в первом круге, в 7952 по команде `JR,C 7957` происходит аварийный выход для подготовки сообщения об ошибке.

На третьем шаге, полученное значение предела вычитается из ТЕКУЩЕГО значения SP (`7954 SBC HL,SP`). Если в процессе вычитания перелив через 65535, значит, снова устанавливается C☑ галочка «C». Но есть нюанс: на этот раз установленная галочка считается, что проверка успешно пройдена, потому что в данном случае из планируемого

меньшего значения вычиталось большее. По команде `RET C` (7956) происходит выход из программы проверки.

Ну а если прикинутая с запасом длина зацепила верхушку SP-башни, то запись прекращается. Стрелочка проваливается в адрес 7957, где напрямую задаётся код ошибки №3 (`LD L,03`). Далее происходит прямой переход в самый центр программы подготовки к выводу ошибки (`7959 JP 85`).

Обогнув кусок для подготовки загрузки массивов, с 2163 продолжается формирование параметров для считывания блока данных в область `PROG`. Снова берётся длина будущей программы из считанного заголовка и начинается подготовка к раздвиганию областей для ожидаемого блока данных:

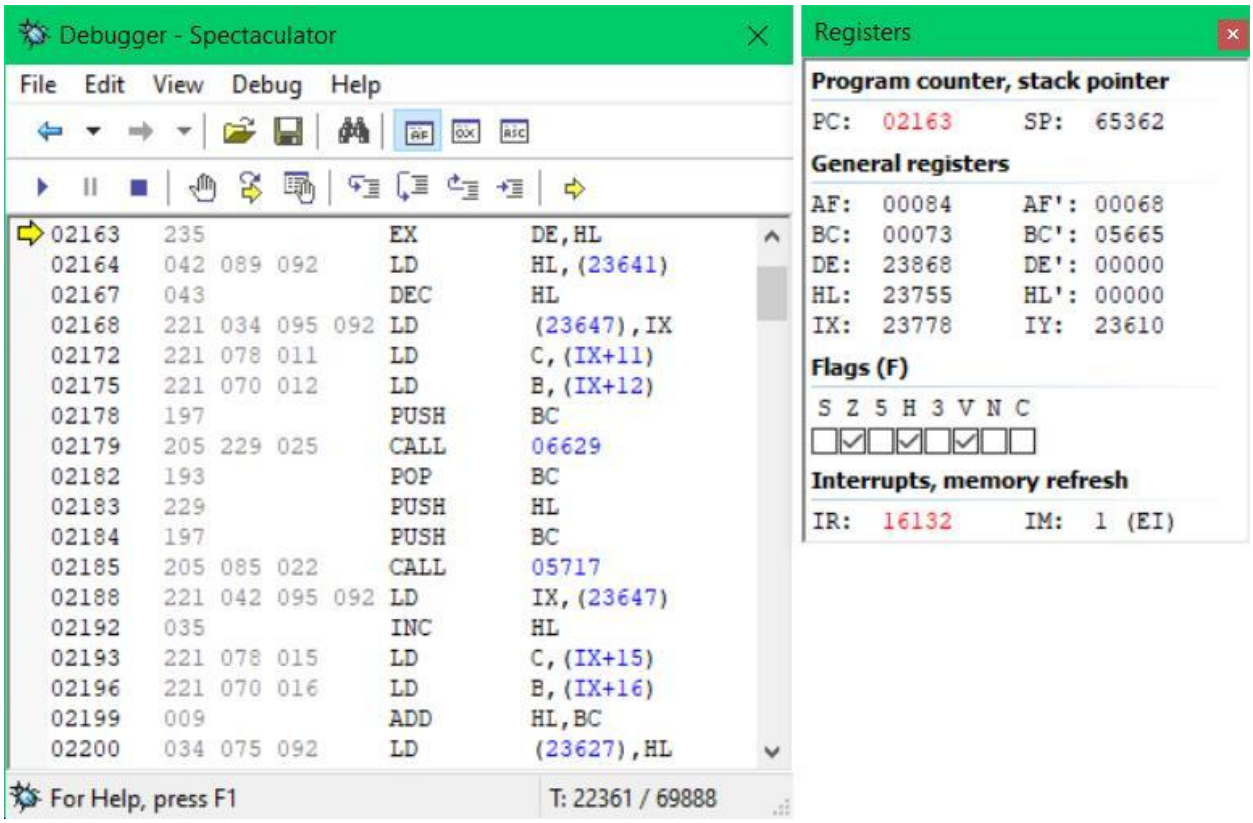


Рис. 430.

В 2185 начинается раздвигание областей подпрограммой `MAKE-ROOM` (`CALL 5717`). Внутри этой программы снова вызывается `TEST-ROOM` (`CALL 7941`). Производится повторная трёхступенчатая проверка, как описывалась выше. Столбик SP подрос из-за вложений, поэтому вычисленные значения последней ступени будут на пару байт меньше.

Итак, при идеальных обстоятельствах на чистом компьютере максимально допустимая длина, которая пройдёт все шесть ступеней защиты, будет 41476. Если SP опустить на самое дно в 65535, то блок может стать еще на 183 байта длиннее.

Последние приготовления начинаются с 2203 проверкой номера строки автостарта:

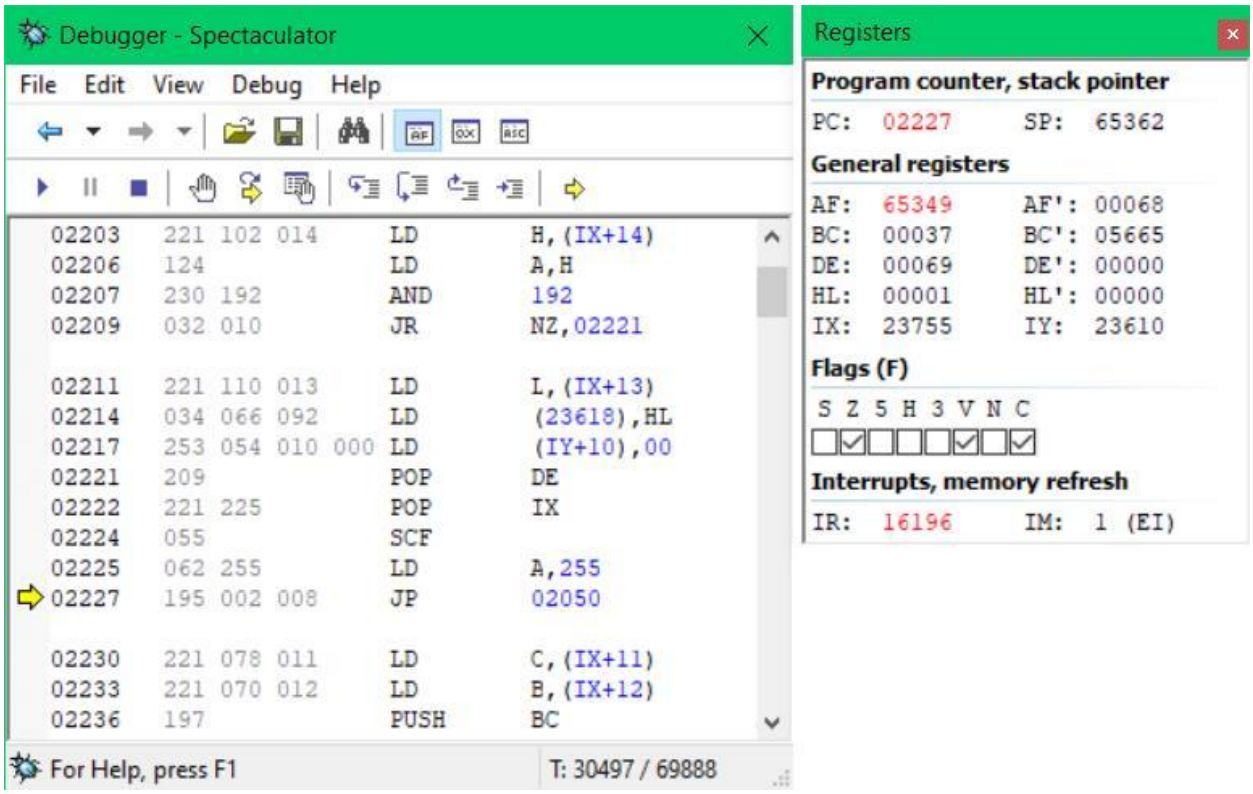


Рис. 431.

Если в ячейках автостарта оказалось число свыше 16383 (срезанные биты полностью не обнулили число), значит, программа не имеет автостарта и можно пропустить этот момент. В ином случае проверяется второй байт строки, и получившееся число переписывается в переменную NEWPPC (23618), организовав перехват, который произойдёт после записи блока данных.

Узнаёте? Это тот самый автостарт «Простой». В комплекте блоков «Program:» для автозапуска используется такой же приём подмены значений, изученный ранее с набором «Bytes:». Это узаконенный вариант перехвата. Именно поэтому, для последовательности и логичности, я решил начать изучение с блоков «Bytes:». Они являются основой всех остальных разновидностей. Напоследок для активации автостарта (во время каскадного выхода в ячейке 7037) номер оператора в NSPPC (23620) просто зануляется.

С 2211 вспоминается длина блока, стартовый адрес, а это обычно 23755, выставляется короткий сигнал «Пи-и-и», галочка «С» для записи считанного, после чего транзитом через 2050, Стрелочка отправляется на запись знакомого блока данных.

Вывод можно сделать один. Комплекс блоков «Program:», это разновидность «Bytes:» со стартовым адресом по указателю PROG и сильно ущемлённая в правах. Мало того, максимальный размер загружаемого блока будет напрямую зависеть от местонахождения столбика SP.

Глава 56
Автостартующий комплекс «Program:»

Краткое содержание: блок Program:, структура автостарта

Поскольку 99% блоков «Program:» обычно делают автостартующими, предлагаю начать именно с него:

```
New File [SAVE Program LINE.tzx]
Cassette Recorder [Record]
Debugger
```

```
Dec
Go To 23755
23755 ← 0 1 33 0 245 97 36 58 245 173 167 59
23767 ←""...CMOTPETb HA Cauraka
23790 ← 34 13 65 29 0 22 10 3
23798 ←Kocmuk egem B KA3AXCTAH
23821 ← 46 46 46 128

17996 ← 23755 0
18000 ← 0
18001 ← 19 1 17 5 67 97 117 114 97 107
18011 ← 69 0 1 0 37 0

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD "" ENTER
Cassette Recorder [Play]
```

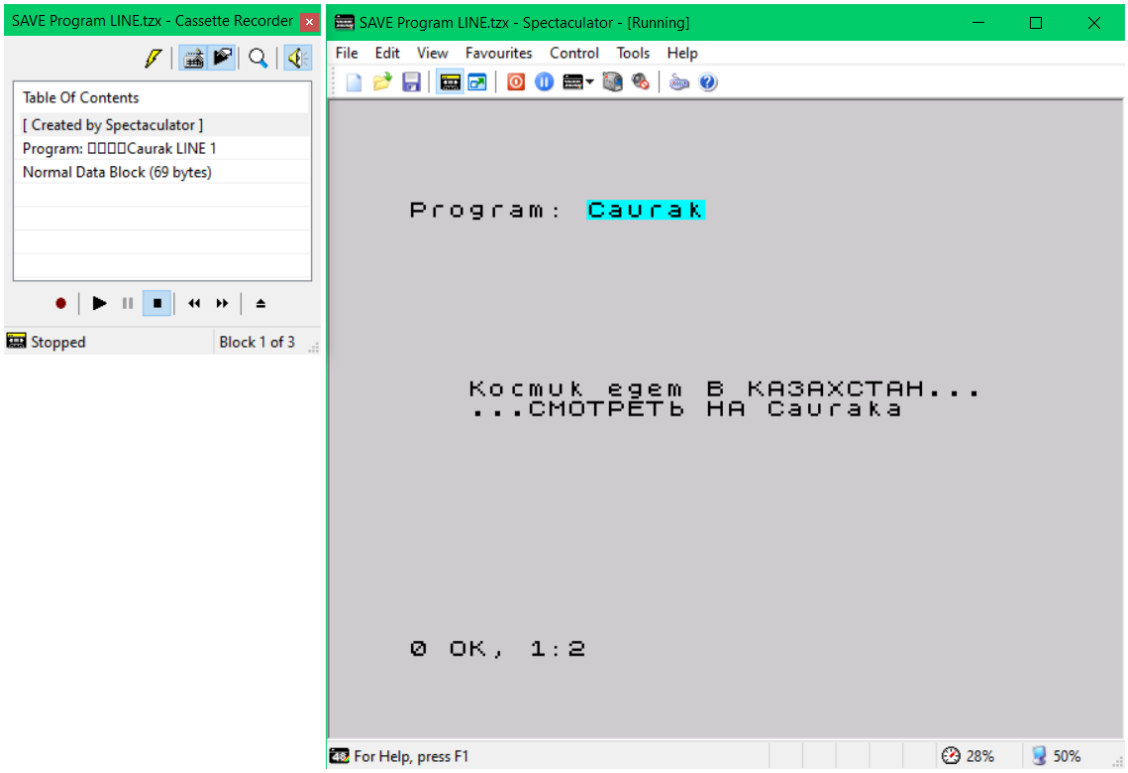


Рис. 432.

Вот пожалуйста! Полноценый автозапускной блок «`Program:`», записанный абсолютно той же методикой, что и «`Bytes:`». Различий вообще немного. Самое первое условие для создания «`Program:`» это код блока «0» в первом байте заголовка. Второе это размерность. У этого блока параметр адреса размещения отсутствует. Место для записи выбирается на ходу, по текущему состоянию указателя `PROG (23635)`. Несмотря на это параметров у него целых три штуки, Средний это указатель строки автостарта, а остальные два задают длину. В текущей программе этими значениями будут «69 0 1 0 37 0». Предлагаю ещё раз рассмотреть, но на конкретном примере. Расшифровываются они следующим образом:

69 0 – суммарная длина данных и переменных из дружественных областей [PROG]+[VARS] без маркера «128», который отделяет их от буфера вводимой строки E_LINE.

1 0 – номер строки автостарта. Именно номер, а не адрес. Для того, чтобы блок «Program:» был без автозапуска, достаточно поставить значение строки свыше 16383. При записи традиционным способом, команда SAVE забивает туда заглушку «128».

После считывания заголовка, этот параметр заносится в ячейки 23618/19 и после загрузки блока данных происходит автостарт «Простой», который подробно разбирался несколькими главами ранее. Он точно также основан на подмене значений системных переменных в процессе загрузки блока.

37 0 – чистая длина BASIC программы без области переменных, включая последний ENTER. Иными словами массив области [PROG] без [VARS].

Конечно, если программа не содержит блока с переменными, первый и последний параметр должен быть одинаковый. Например, если из этой программы удалить переменные, то параметры для записи станут: «37 0 1 0 37 0».

Наверное, нет смысла отдельно рассматривать блок без переменных или автостарта. Изменив пару значений, их можно создать из текущего примера.

Глава 57
Комплекс «Program:» с картинкой внутри

Краткое содержание: картинка в блоке Program:, возвращаясь к напечатанному

А теперь очередной раз возвращаюсь к напечатанному. В книге 2013 года «*ZX-Spectrum-48K в эпоху windows и эмуляторов*» я приводил пример блока «Program:» с картинкой внутри. Сколько лишних манипуляций и глупостей я там наворотил. Кучу преобразований и вклеек с помощью «*Hex Edit*» и «*Tapir*». Нахуа? И даже не смог нормально стереть вспомогательную строку «RANDOMIZE USR 1», а подменил её «REM Pustota!», специально подобрав слово нужной длины. Этот «индусский код» я даже критиковать не буду.

На самом деле всё гораздо проще. И приёмы для создания такой штуки уже изучали. Из нового будет только вставка картинки не в область экрана, а в заданный адрес.

Из предыдущих глав «*Графика*» и «*Запись «Bytes:» с сохранением нижних строк*», в папке с созданными программами, должны были оставаться картинки в формате .scr. Возьмите одну из них, сделайте копию и переименуйте расширение в .bin:

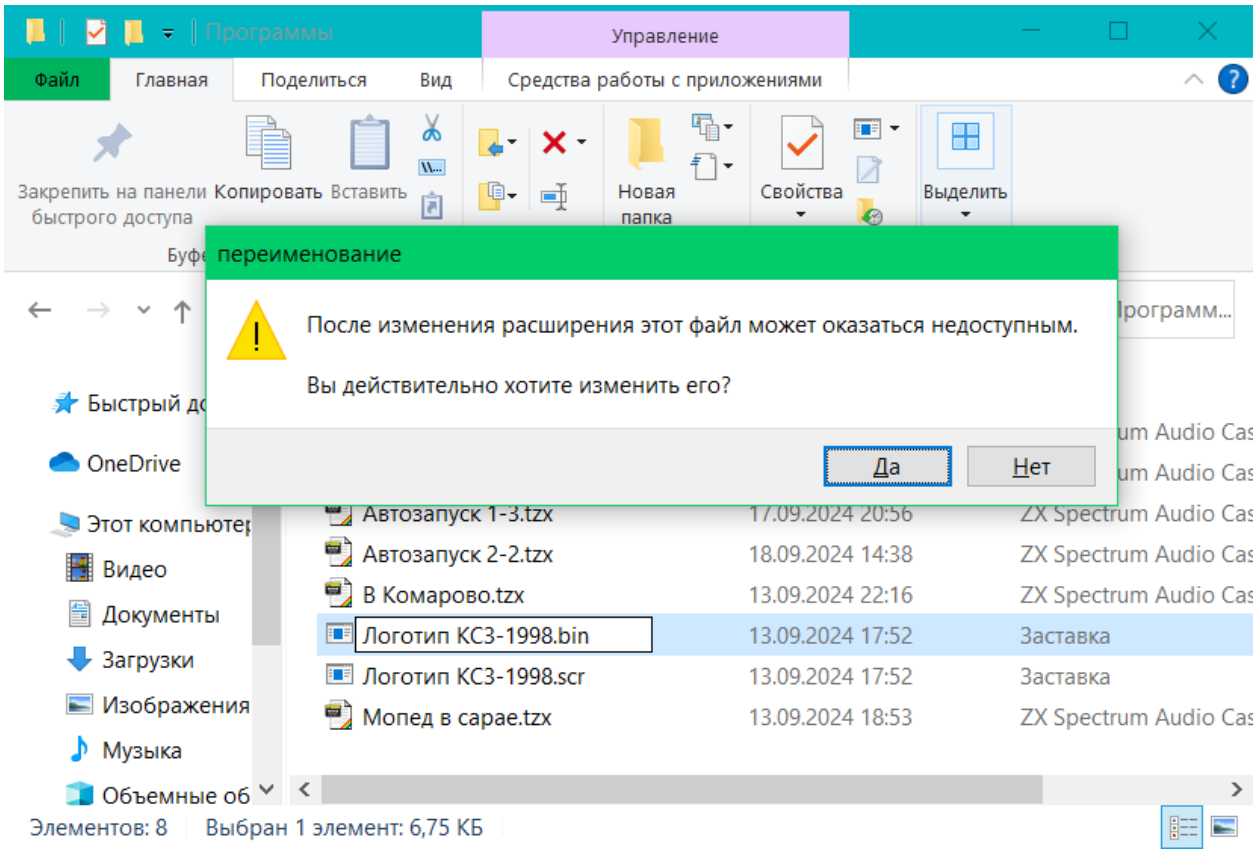


Рис. 433.

После переименования файла подготовка и закончена. На самом деле *.scr* это такой же блок сырых данных, как и *.bin*.

Откройте *Spectaculator* и перетащите на экран только что созданный .bin файл картинки. В моём случае это «*Логотип КС3-1998.bin*». Моментально откроется окошко «*Import Machine Code*». В полоску «*Start Address*» введите 23806:

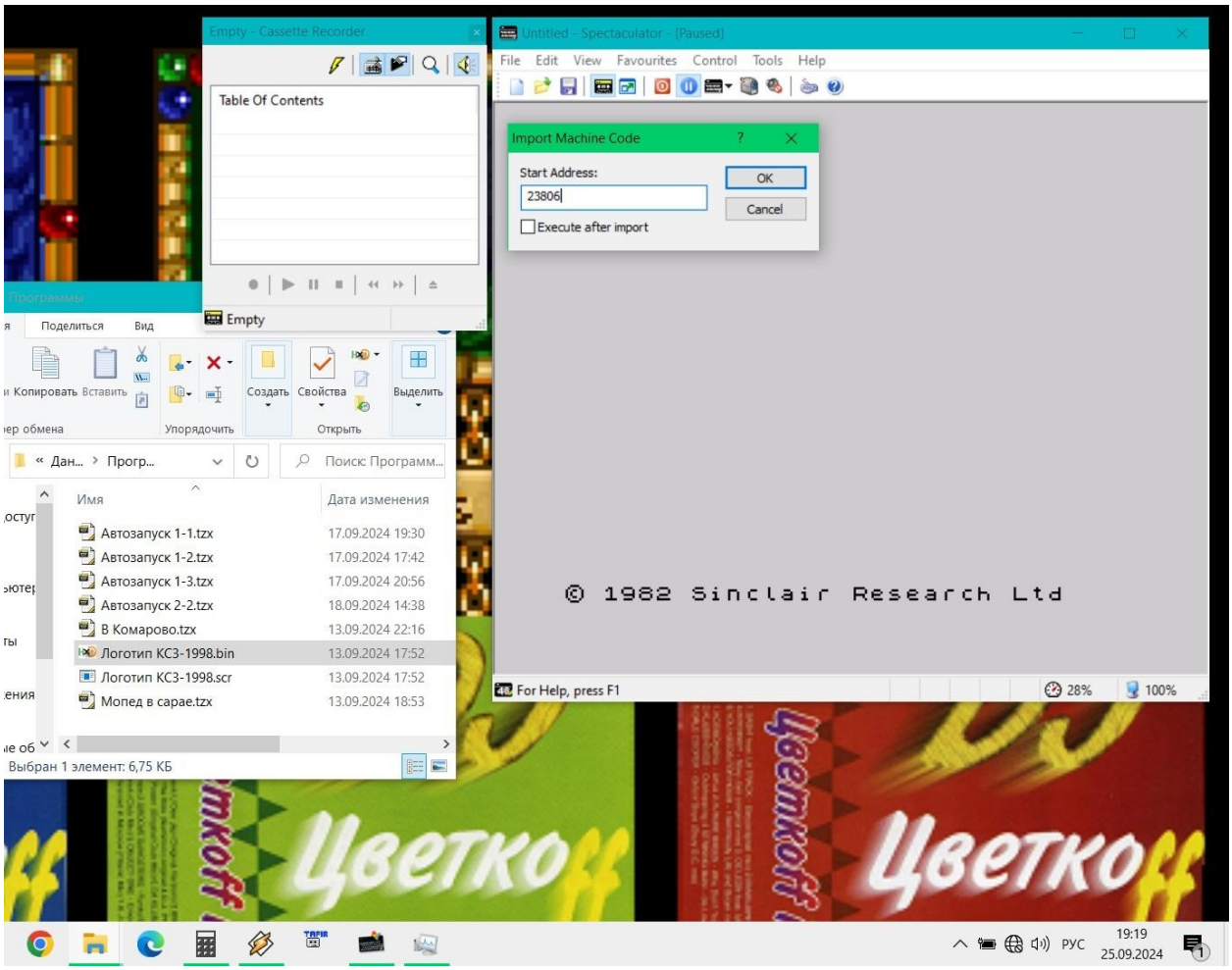


Рис. 434.

Окошко закроется, а с адреса 23806 вставится блок данных картинки. Вместо того, чтобы вводить данные вручную, в память оптом вставились 6912 байт графики. Удобно, правда? Данную манипуляцию предлагаю обозначить как:

```
Open File [Логотип КС3-1998.bin]
Import Machine Code 23806
```

Я надеюсь, что так будет достаточно лаконично подразумеваться манипуляция: «*вставьте файл данных такой-то по адресу такому-то*».

Картинка уже вставлена, дальше по знакомой технологии выполните следующий алгоритм. Я его привожу вместе с командой вставки картинки, но поскольку уже это сделали, второй раз вставлять не нужно:

```
New File [Картинка в программе.tzx]
Open File [Логотип КС3-1998.bin]
Import Machine Code 23806
Cassette Recorder [Record]
Debugger
Dec
Go To 23755
23755 ← 0 13 128 0 249 192 48 14 0 0 216 92 0 ; RANDOMIZE USR 23768
23768 ← 1 0 27 17 0 64 33 254 92 237 176
23779 ← 42 83 92 54 128 34 75 92 35 34 89 92
23791 ← 253 203 2 238 49 84 255 237 115 61 92
23802 ← 241 195 169 18
```

```
17996 ← 23755 0
18000 ← 0
18001 ← 32 16 2 75 16 1 67 16 4 51
18011 ← 6963 1 6963

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD "" ENTER
Cassette Recorder [Play]
```

Во время загрузки блока звучат знакомые звуки картинки, но экран остаётся белым:

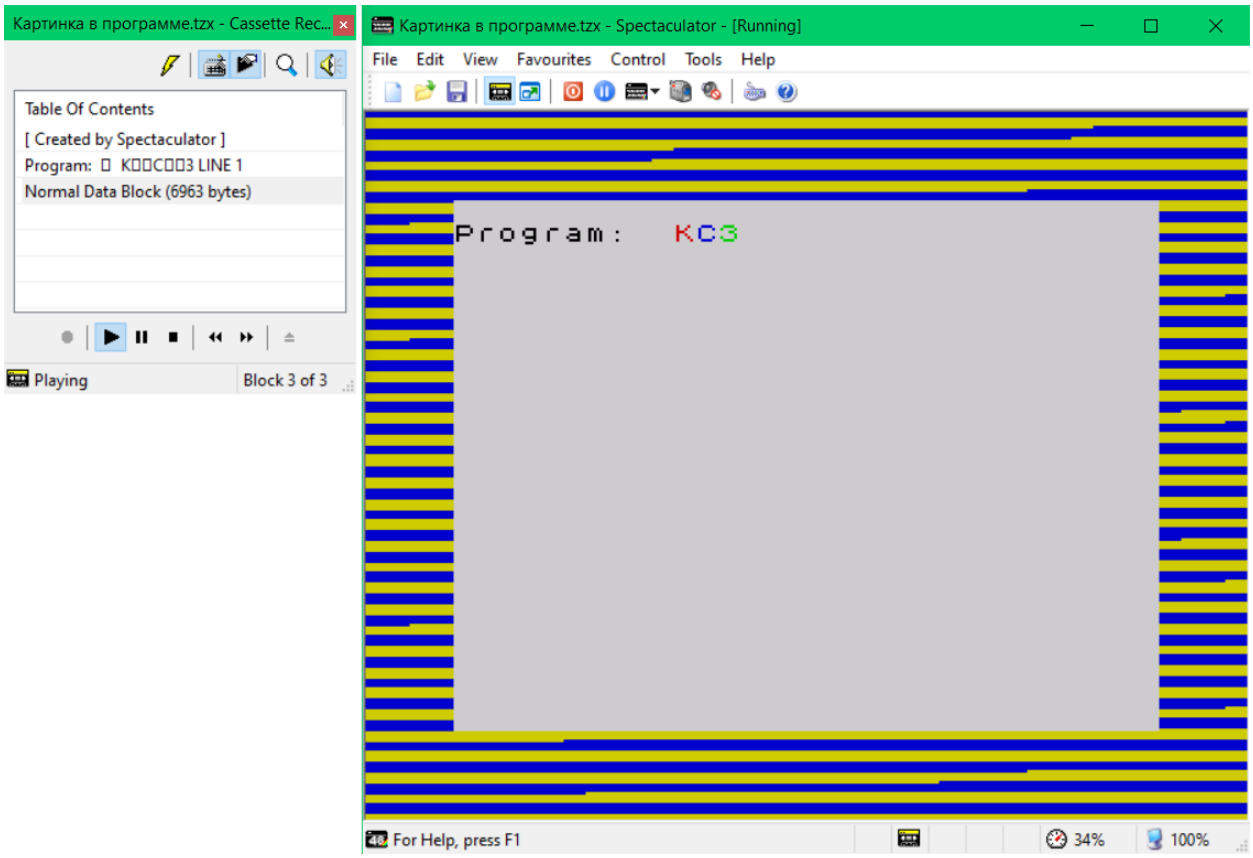


Рис. 435.

Настал момент, когда загрузка закончилась и на экран выскочила картинка. От установленного бита №5 `TV_FLAG (23612)`, компьютер застыл в ожидании нажатой клавиши:

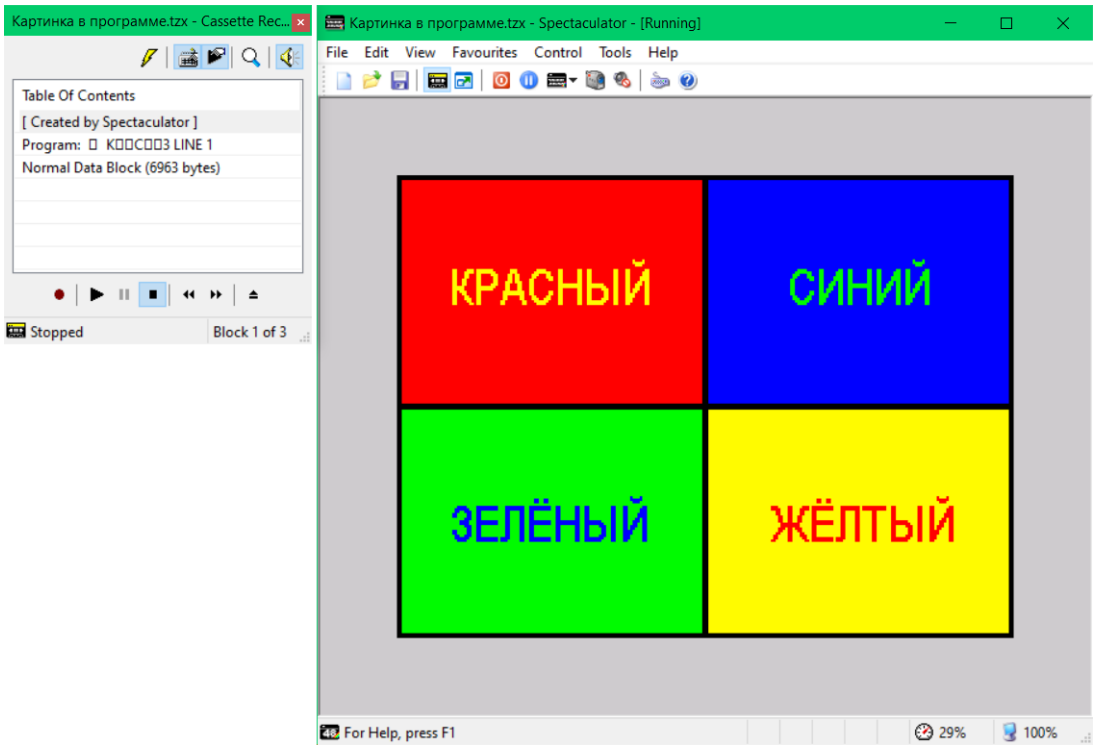


Рис. 436.

Нажмите клавишу ENTER, экран моментально очистится, а там:

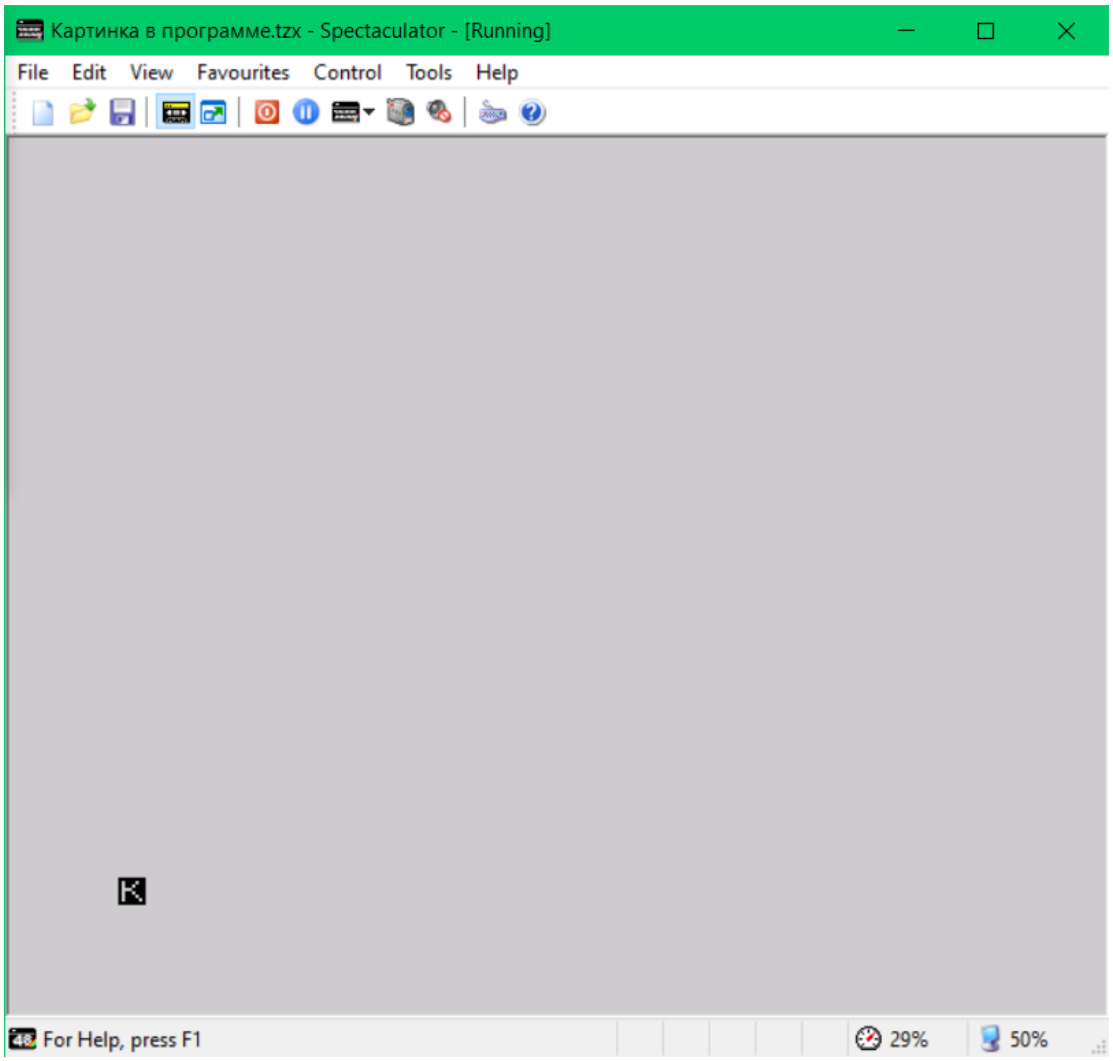


Рис. 437.

…пустота. Настоящая пустота, а не «`1 REM Pustota!`». Строки выполнились, картинка вывелась и область схлопнулась до минимума. А теперь сравните метод 2024-го и 2013 годов. Сколько же там ненужного гемора!

А теперь вариант этой программы с многократным выводом картинки:

```
New File [Картинка в программе-2.tzx]
Open File [Логотип КС3-1998.bin]
Import Machine Code 23790
Cassette Recorder [Record]
Debugger
Dec
Go To 23755
23755 ← 0 0 10 0 249 192 48 14 0 0 221 92 0 13; 0 RANDOMIZE USR 0
[23773]
23769 ← 64 0 18 27
23773 ← 1 0 27 17 0 64 33 238 92 237 176; программа переброски
картинки
23784 ← 1 0 0 195 61 31
23790                                    ; тут встанет картинка
30702 ← 13                               ; ENTER конца огромной строки

17996 ← 23755 0
18000 ← 0
18001 ← 32 16 2 75 16 1 67 16 4 51
18011 ← 6948 0 6948

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""ENTER
Cassette Recorder [Play]
```

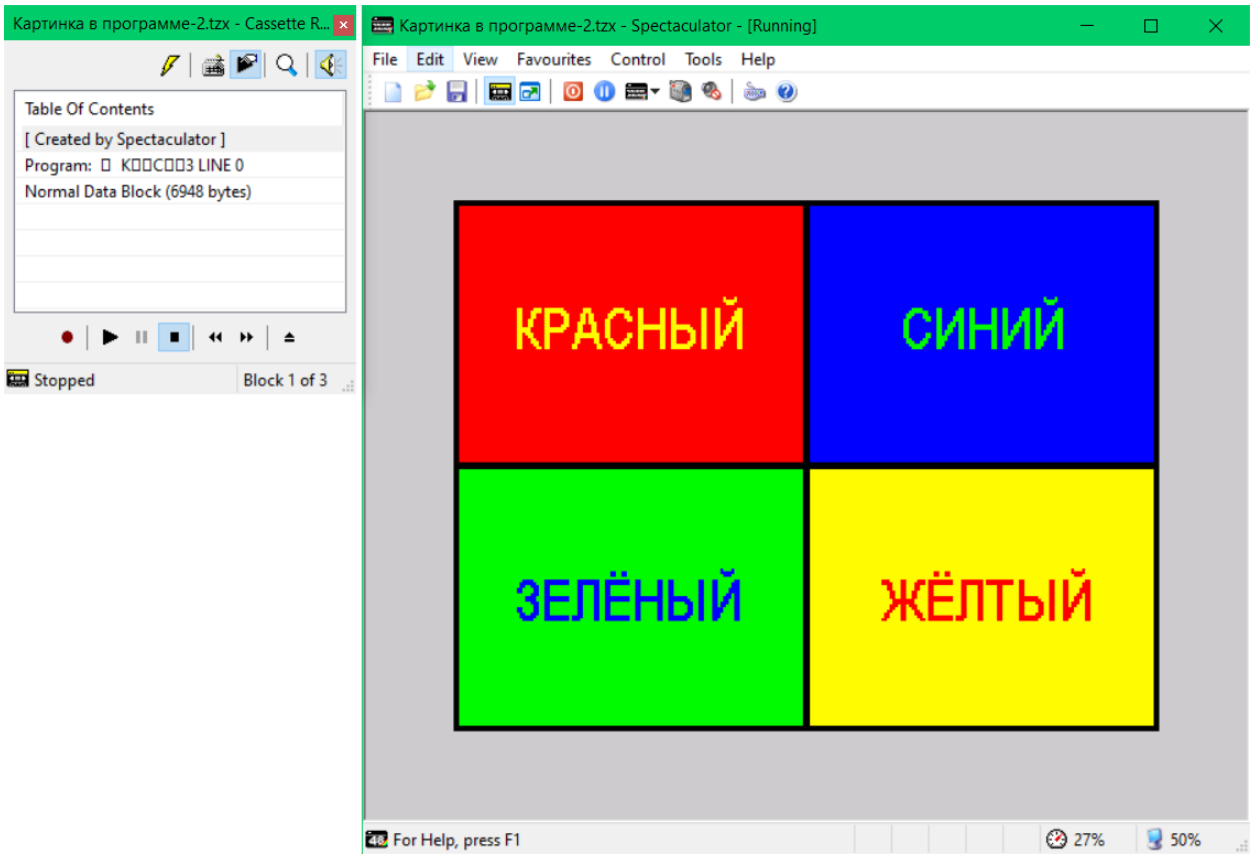


Рис. 438.

Нажмите ENTER пару раз. Экран очистится и появится строка 0:

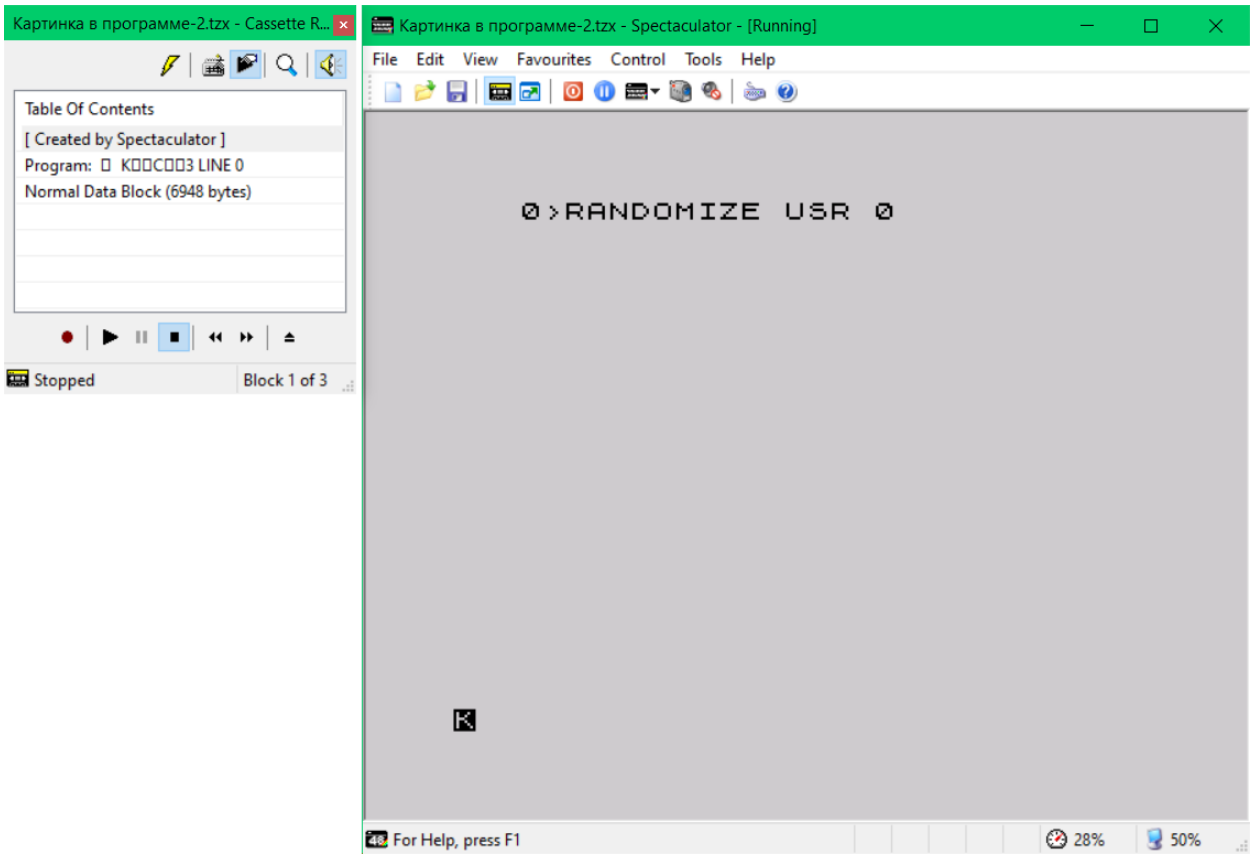


Рис. 439.

Введите RUN, и картинка снова появится на экране. Недостаток этой программы в том, что любая вставленная строка после 0, сместит адрес программы вывода картинки и программа больше не запустится без корректировки.

Глава 58
Самый длинный «Program:»

Краткое содержание: блок Program:, обход защиты

Напоследок предлагаю сделать блок «Program:» максимальной длины. В главе «*Знакомство с Program:*» рассматривался принцип работы комплекса. Также упоминалось, что максимальная длина блока данных этого типа будет 41476 байт. Напомню, что после вывода заголовка «Program:» кроме общей подготовки параметров, размер загружаемого блока данных, дважды проходит трёхступенчатую проверку. Если в какой-то из шести проверок значение превысит допустимое, то блок после заголовка не загрузится, а вместо него выскочит сообщение «4 Out of memory».

Введите следующую программу:

```
New File [Самый длинный Program.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 23755
23755 ← 0 1 38 0 245
23760 ←''''''''''"       КОСМОНАВТ В КАСКЕ
23795 ← 34 13 0 2 27 0 245
23802 ←""       МАХАЕТ АВТОМАТОМ
23826 ← 34 13 128 13 128

17996 ← 23755 0
18000 ← 0
18001 ← XXXLPA3MEP
18011 ← 41476 0 41476

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD "" ENTER
Cassette Recorder [Play]
```

Начинает загружаться программа. Преодолены пять степеней проверки на допустимый размер, и Стрелочка встала на последнюю шестую проверку:

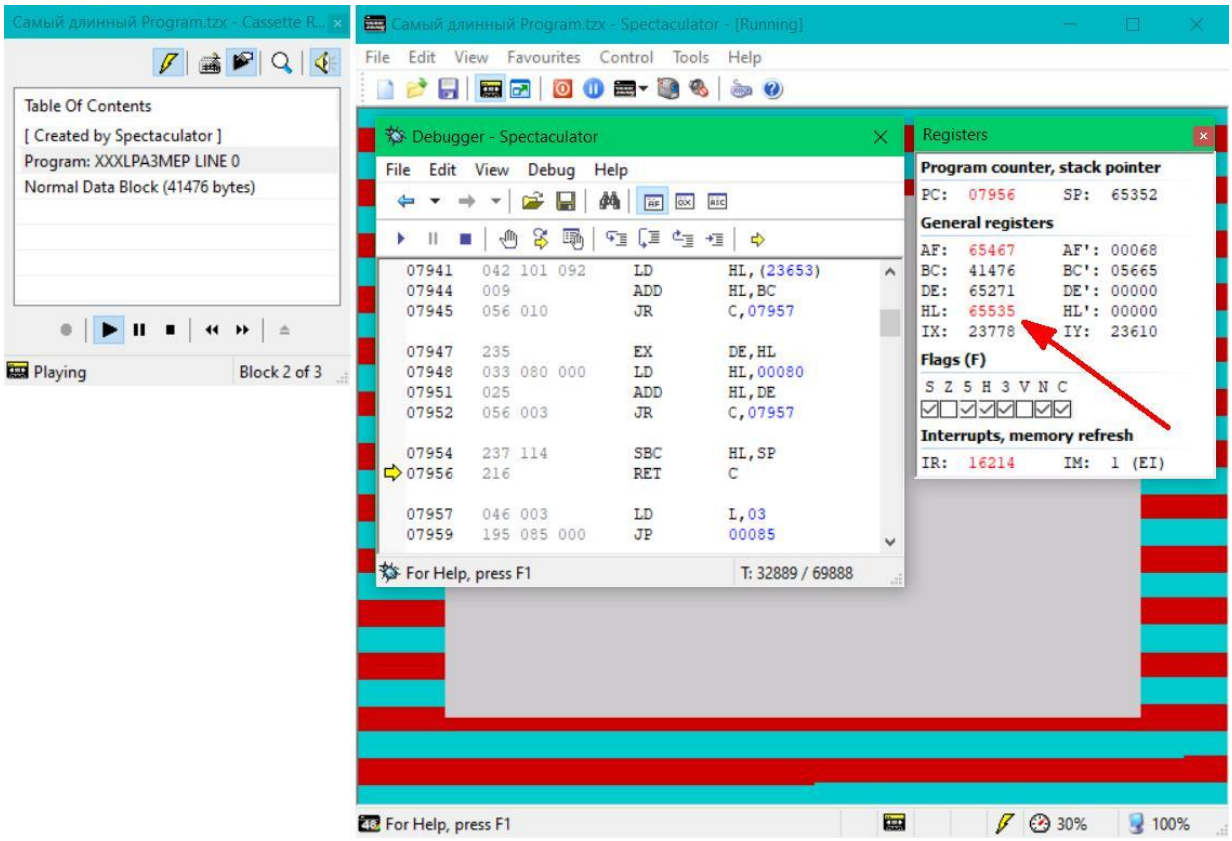


Рис. 440.

И это можно назвать впритирку, или «проскочить на тоненького». Посмотрите, какой точный расчет! Байтом больше и следом идущий блок данных уже не считается. После загрузки, на экран выдался тестовый текст, сигнализирующий об успешном запуске программы:

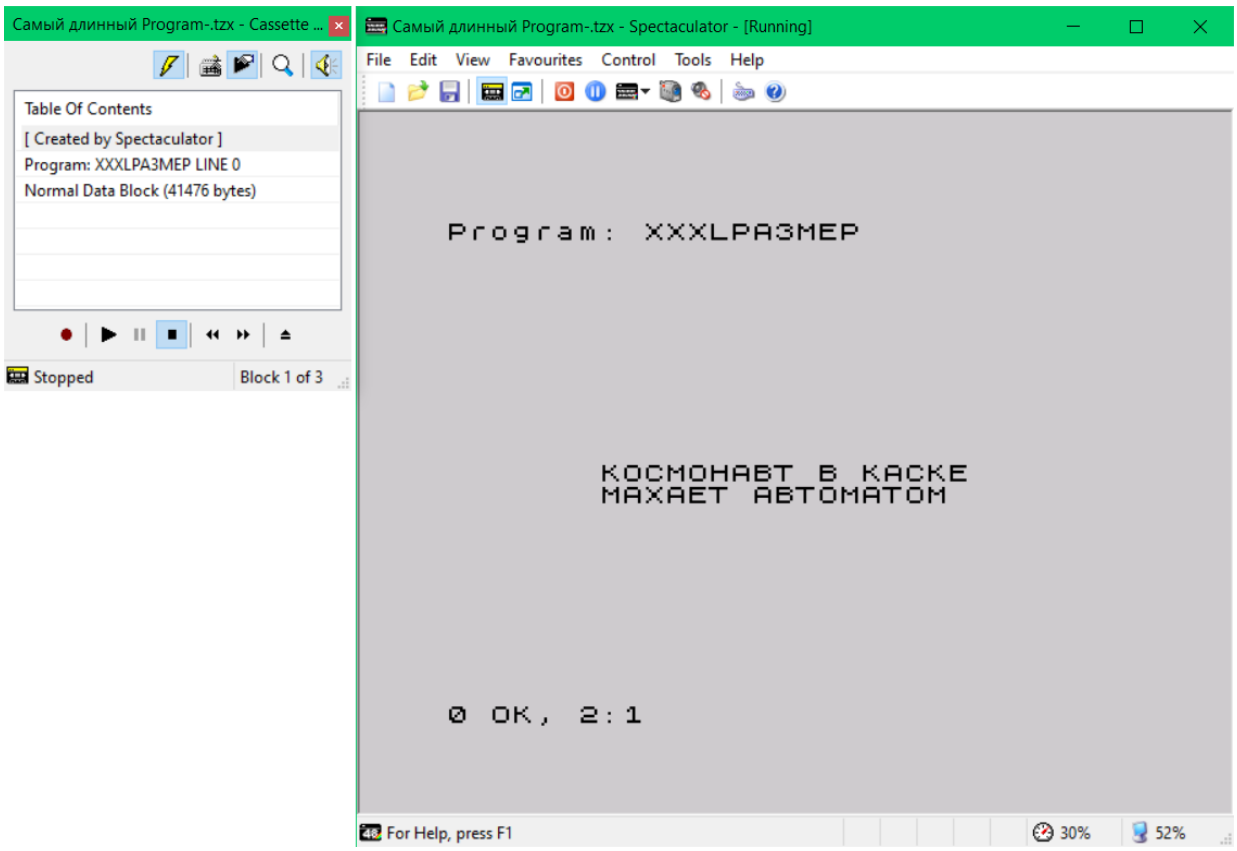


Рис. 441.

На далёкой планете, которая находится за несколько тысяч световых лет, начались беспорядки. Гуманоиды затерроризировали местных жителей, и те забили тревогу. На призыв о помощи, на планету отправили экспедицию космонавтов, которые были экипированы по последнему слову техники. На головах у них были чёрные каски, а в руках блестели новенькие автоматы АК-74, модифицированные версии которых стреляли сверхмощными синими лазерами…

Вспоров гиперпространство, космический корабль «УРАЛ-4320» понёсся к далёкой звезде спасать жителей от нашествия гуманоидов…

А вот теперь если вы попробуете ввести дополнительную строку программы, более 15-20 символов, то в ответ получите сообщение: «`G No room for line`».

Отредактировать текущую строку тоже не выйдет. Вместо редактирования выскочит сообщение «`C Nonsense in BASIC`». При дальнейших экспериментах, в буфере строки вместо символа послышится рычание. В любом случае, после загрузки этого блока, память закончилась и ничего хорошего не выйдет:

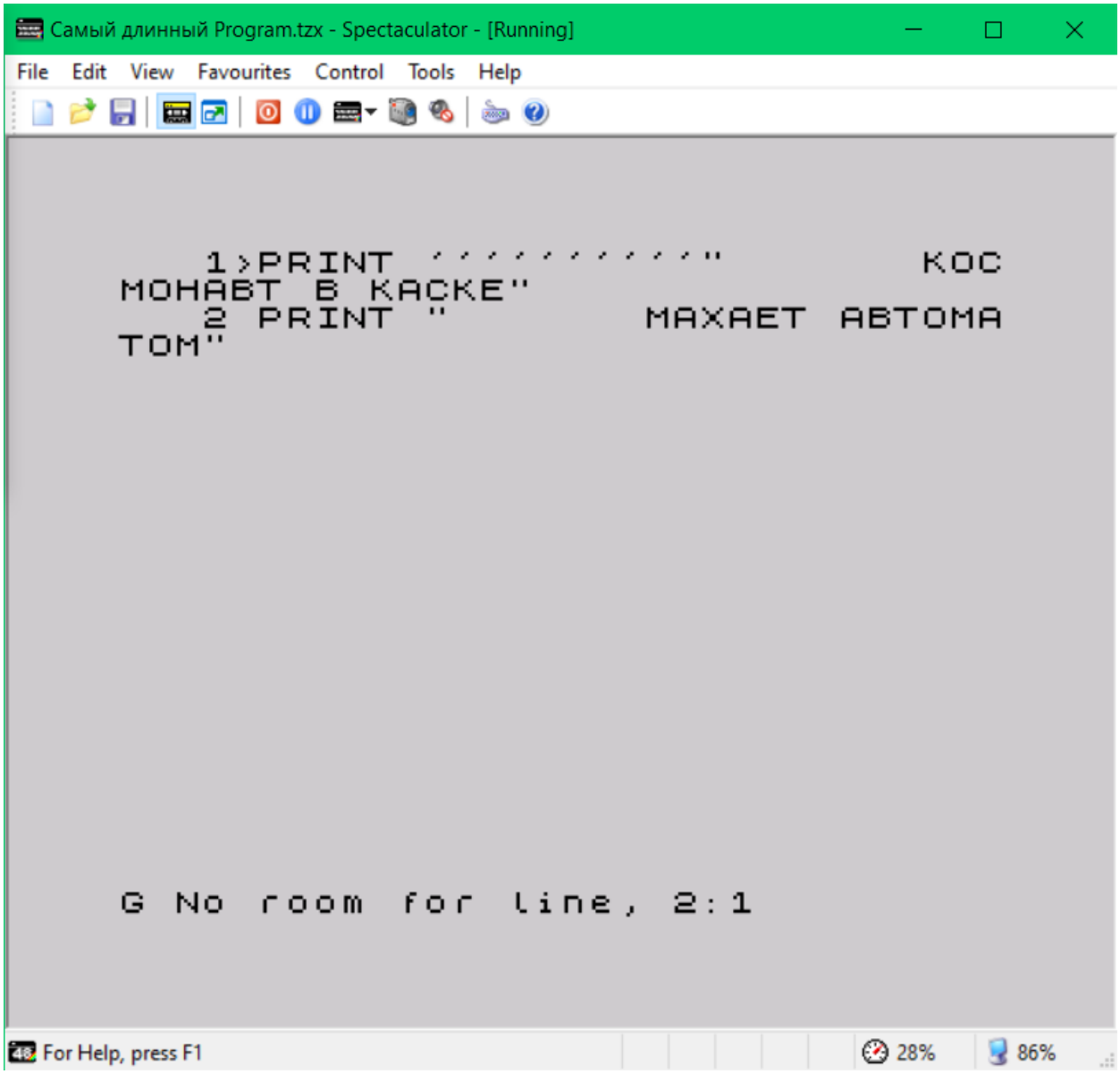


Рис. 442.

В этом случае остаётся только нажать сброс, если лень вручную смещать указатели. Ниже даю небольшой расчёт по критическим значениям для всех шести точек

проверки. Информация актуальна при загрузке на абсолютно чистый вариант 48К после сброса:

1-1 - 41737
1-2 - 41657
1-3 - 41479
2-1 - 41741
2-2 - 41661
2-3 - 41477

И напоследок предлагаю загрузить сверхбольшой набор «Program:», размером от 41477 до 65533 байта. Создать и загрузить его можно с помощью следующего алгоритма:

```
New File [Фантастический Program-65533.tzx]
Open File [Логотип КС3-1998.scr]
Cassette Recorder [Record]
Debugger
Dec
Go To 23613

23610 ← 255
23611 ← 128
23613 ← 65364
23618 ← 0 0 0
23627 ← 23785
23641 ← 23786 23786 23786
23649 ← 23788 23788 23788
23755 ← 0 0 26 0 245 173 167 42 167 59 221 188
23767 ← 167 59 34 117 32 66 117 109 97 101 109
23778 ← 32 114 97 112 98 34 13 128 13 128

40000 ← 118 59 59 209 253 203 2 134 33 21 0 25
40012 ← 235 1 28 0 205 60 32 195 118 27
40022 ← 22 11 5 19 1 16 2
40029 ← В ukарусах cmoum BOHb

65358 ← 1343 2053 40000 4867

23396 ← 23755 0
23400 ← 0
23401 ← XXXLPA3MEP
23411 ← 65533 0 65533

IX ← 23400
SP ← 23396
PC ← 2436
Trace
Cassette Recorder [Stop]

Add Breakpoints 2050, 2091, 2185
Trace
BASIC ← LOAD ""ENTER
Cassette Recorder [Play]
PC ← 2094
Trace
Step Into
PC ← 5722
```

```
Trace
Step Into
Step Into
AF ← AF-32
Remove Breakpoints 2050, 2091, 2185
Trace
```

Программа состоит из трёх частей: содержимое, запись блока и загрузка с обходом защиты. Когда вы выполните этот сложный алгоритм, начнётся загрузка блока «Program:», который пройдётся по всей памяти, перепишет SP. Перескочив через предел 65535, прогуляется по ПЗУ и нарисует картинку в реальном времени:

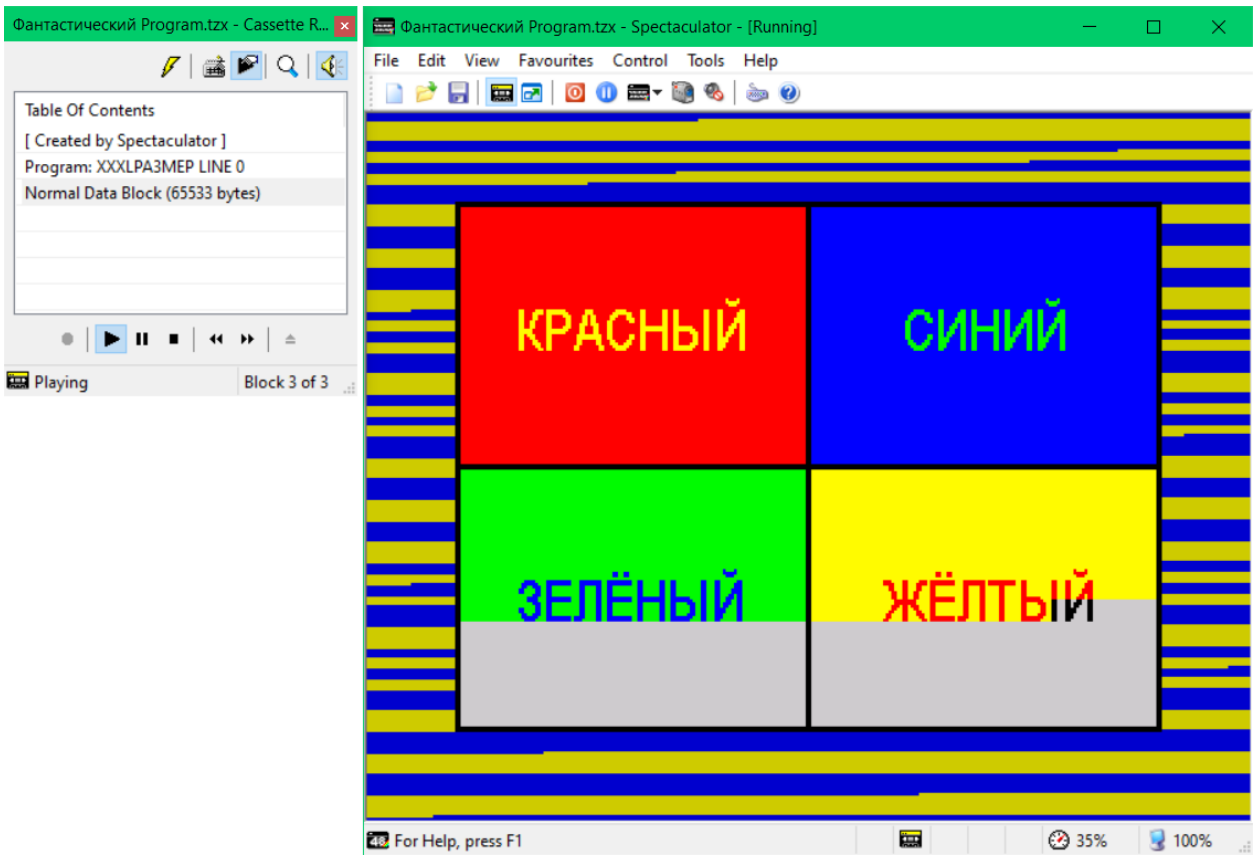


Рис. 443.

После загрузки, по созданному SP-столбику произойдёт переход на адрес 40000, где выведется первая часть фразы. Следом, после возврата в BASIC, по адресу 7030 запустится строка BASIC с продолжением фразы. А после всего этого в нижней строке появится сообщение «0 OK»:

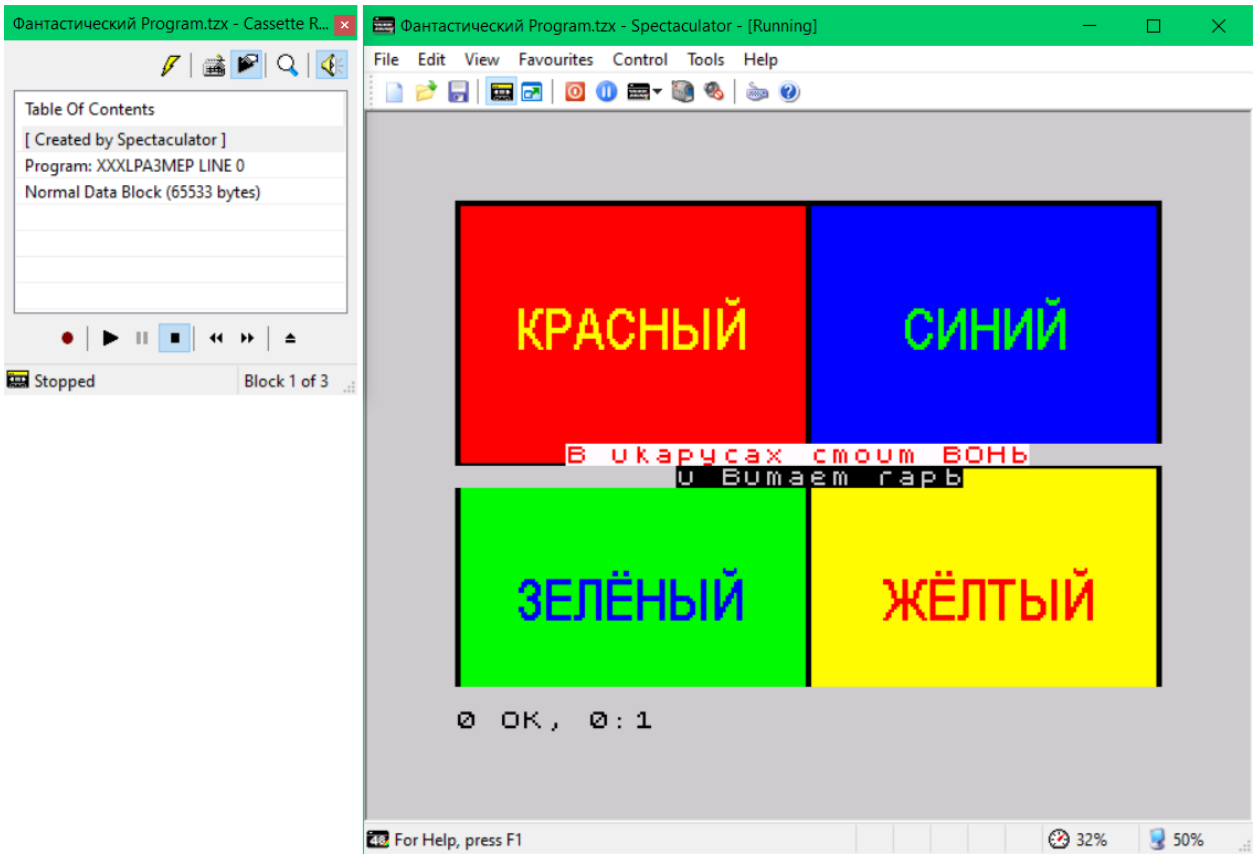


Рис. 444.

Мощный дизель крутит поршни
Вдаль везёт Икарус нас.
Кто изобрёл этот автобус –
Тот настоящий молодец!

В нём печки нет, но много вони,
Соляркин нюхай аромат.
Заносит сильно в поворотах,
Но есть коробка-автомат.

Пусть драная, скрипучая резина
Которая гармошкой названа.
Сыграет нам мелодию ноктюрна
На флейте своего...

Рис. 445. Гламурный Икарус (борт №7255 В 426 ВА 78RUS) на проспекте Славы. Фото 1 мая 2006 года.

А если без приколов, Икарусы, это легенда 1980-х и 90-х годов. Зимой не жарко, летом не холодно. Вонь в салоне, дождь льёт сквозь с драную гармошку. Несмотря на это, об автобусах остались самые тёплые воспоминания.

Нажмите ENTER, и на экране появится BASIC строка:

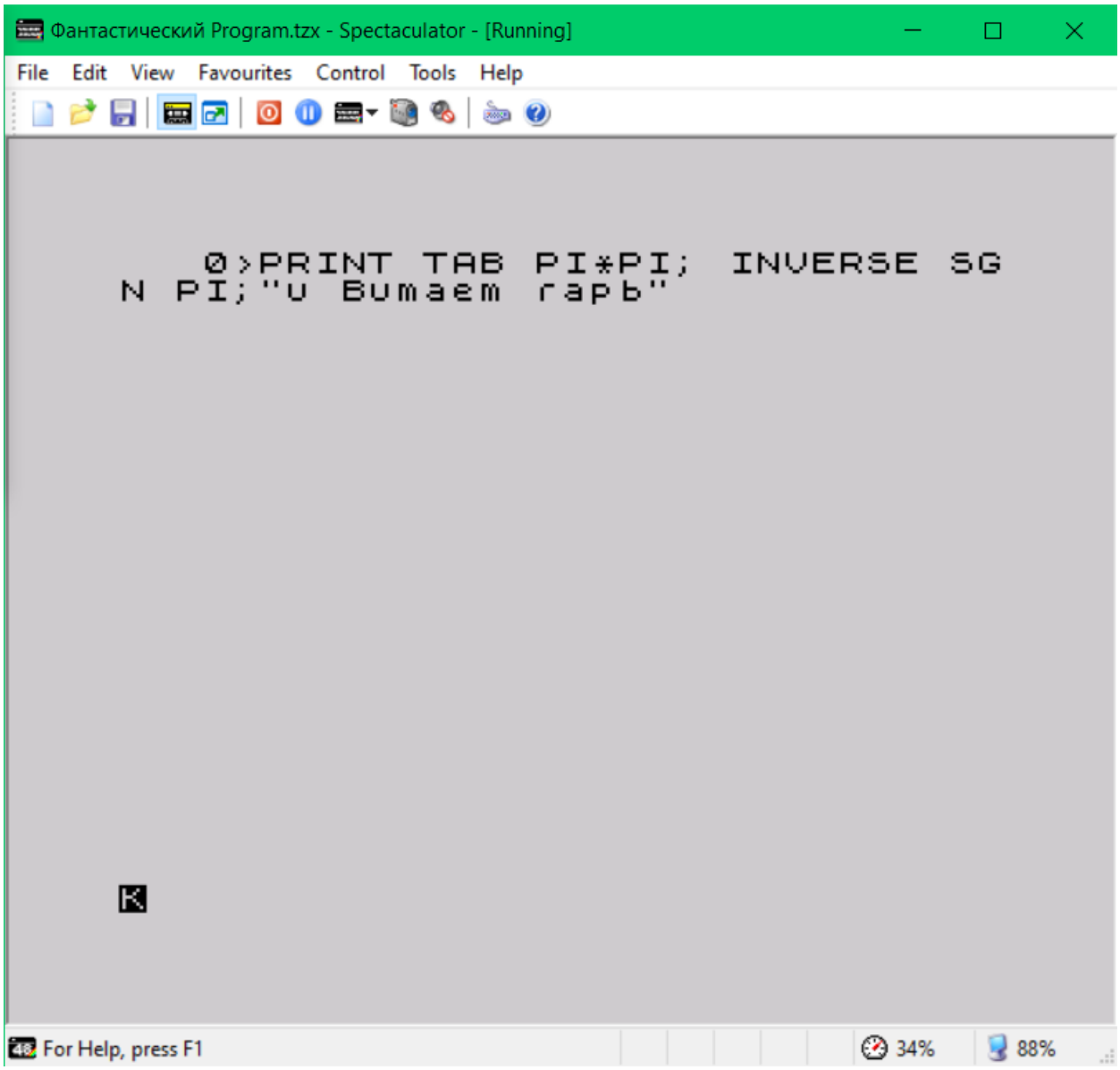


Рис. 446.

Таким образом, получилась достаточно сложная пародия на главу 7 «*Загрузка и запуск блоков с переходом через предел 65535*» из старого издания книги.

Глава 59
Тайна ошибки «Character Array:»

Краткое содержание: блок Character array:

«`Character Array:`» и «`Number Array:`» самые специфические и малоиспользуемые типы блоков. Если первый можно хоть как-то приспособить для хранения текстовой информации, то за каким фигом придуман последний вообще непонятно. И проблема даже не в применении. Отсутствие в самоучителях по BASIC нормальной информации, о том, как правильно подготовить такой блок к записи для корректного считывания, вгоняло в тоску. А зачем объяснять? Всё должно быть ясно на подсознательном уровне.

Именно поэтому после загрузки такого блока, кроме ошибки «`3 Subscript wrong`», до последнего момента считать ничего не удавалось. Но всё изменилось, когда я вплотную занялся этой книгой. Взглянув в отладчик, всё встало на свои места.

Основные ошибки и недоработки предлагаю рассмотреть на примере. Предположим, вы ввели следующую BASIC-строку натуральным способом:

```
FOR a=1 TO 10: LET a$="ХРЕН ТАМ": LET b=100500
```

После отработки, в область VARS запишутся следующие данные:

225 0 0 1 0 0 0 0 10 0 0 0 0 1 0 0 254 255 2 **65 8 0 88 80 69 72 32 84 65 77** 98 145 68 74 0 0

Синяя цепочка это заряженный FOR-цикл, красная – символьная переменная и зелёная однобуквенная числовая переменная. Допустим, вы хотите сохранить эти данные. Для записи символьной информации такого типа разработчиками прошивки предполагался блок «`Character Array:`». Из BASIC он должен записаться следующей командой:

```
SAVE "ХРЕН ТАМ" DATA a$ ()
```

После записи получился блок, длиной 8 байт. Произошла фильтрация лишних данных, и в блок записались только переменные определённого вида. Теперь нужно нажать Reset и попробовать считать:

```
LOAD "" DATA a$()
```

После загрузки на чистый компьютер, в области VARS появилось следующее:

193 8 0 88 80 69 72 32 84 65 77

Осталось сравнить с тем, что было в начале:

65 8 0 88 80 69 72 32 84 65 77

Символьная переменная 64+ трансформировалась в «DIM-массивную» 192+, однако больше ничего не изменилось. Но из прошлых глав вы знаете, что для массива данных необходима разметка, которой изначально не было. В итоге после загрузки родился такой неполноценный уродец. Логично, что при попытке ввести `PRINT a$` возникает ошибка. Варианта тут два: поменять число 193 обратно на 65 или добавить между значением длины и содержимым массива DIM-разметку. В любом случае, без вмешательства записанный массив работать не будет. На мой взгляд, это серьёзная недоработка и можно было отсечь возможность записи однобуквенной символьной переменной, чтобы не вводить людей в заблуждение. К счастью, если записывать полноценный массив с разметкой, то всё нормально запишется.

И снова вернусь к своей книге 2013 года. На этот раз к Главе 3 «*Создание символьного массива, начинённого машинной программой*». Приоткрою тайну. Тогда я хотел записать символьную переменную этим специальным блоком, а после загрузки запустить массив. Не понимая, как он работает, я записывал переменную формата «A$», и после загрузки получал эту самую ошибку.

Свалив всё на неправильную работу эмулятора, я плюнул и на ходу переиграл пример. Чтобы выйти из положения я запихал в мутированный символьный массив машинную программу, сделав вид, что так и задумывалось. При этом деликатно умолчал, что проверка `PRINT a$` даже после загрузки чистой переменной даст ошибку «`3 Subscript wrong`».

Сейчас другое время подход и знание ситуации. Предлагаю синтезировать работоспособный блок «`Character Array:`» с нормальной DIM-разметкой, который после загрузке по команде `PRINT a$` выведет строку с текстом безо всяких уловок с машинными кодами:

```
New File [Запись Character Array.tzx]
Cassette Recorder [Record]
```

```
Debugger
Dec
Go To 23300
23300 ← 1 40 0
23303 ← 22 11 0 16 3 19 1 17 6
23312 ←MAMA, PAMA ga KACCETA Radiorama

17996 ← 23300 0
18000 ← 2
18001 ←Radiorama+
18011 ← 43 0 0 193

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD "" DATA a$() ENTER
Cassette Recorder [Play]

BASIC ← PRINT a$ ENTER
```

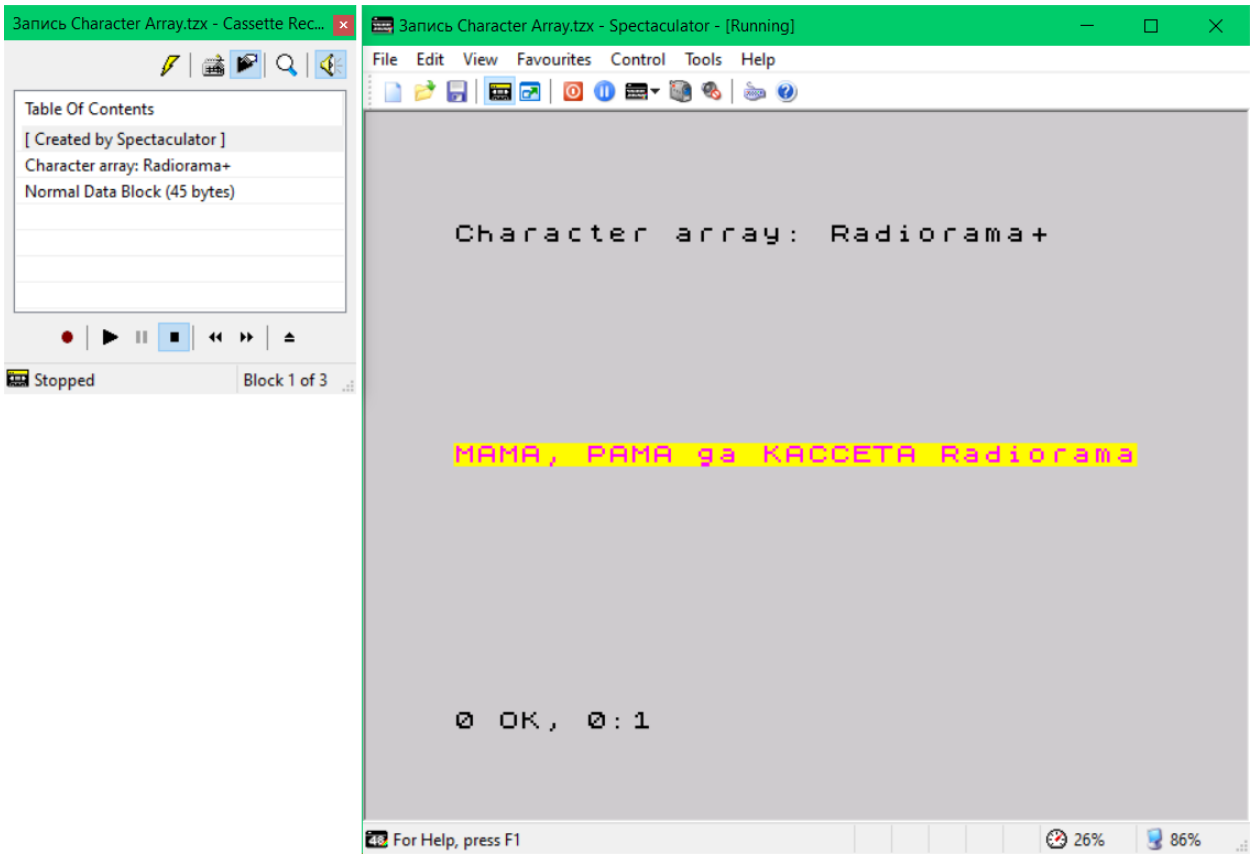


Рис. 447.

Как видите никакой «ошибки эмулятора». Блок загрузился, правильно считался и выдал текст по прямому вызову переменной. Неудобство состоит в сложной конструкции загрузки, по сравнению с `LOAD ""` или `LOAD ""CODE`. Все эти скобочки и обязательная установка буквы после `DATA` не вселяют оптимизма. Ну и команду `RUN`, без сдвига указателя `VARS (23627)` в конец области, набирать противопоказано. Всё будет моментально стёрто.

Глава 60
«Number Array:» преврати числа в программу

Краткое содержание: блок Number Array:

Блок числового массива «Number array:» самый редкий обитатель мира ZX-Spectrum. Как и ближний родственник, он имеет все те же недостатки: трудности загрузки и отсутствие толковой информации. В настоящее время это не помешает изучить его свойства и научится синтезировать искусственным путём. Именно в него я предлагаю записать программу, выводящую текст. Может возникнуть мысль, что текст логичнее размещать в текстовом массиве. Чистый текст, действительно лучше размещать в текстовом массиве, а программу в машинных кодах пихать туда бессмысленно.

Из прошлых глав вы знаете, устройство числового массива и то, что числа хранятся в рыхлом 5-ти байтном формате. Если в символьном массиве типы кодов будут иметь разные свойства, то в числовом массиве программа в машинном коде будет распознаваться как набор 5-ти байтных чисел от «0 0 0 0 0» до «255 255 255 255 255». Любая последовательность управляющих символов будет восприниматься, как часть крупного или дробного числа. Таким образом, данные из загруженного числового массива можно будет вывести как запланированным образом (PRINT a(1), PRINT a(2)…), так и обращением по команде RANDOMIZE USR 237xx. В обоих случаях произойдёт корректное выполнение программ, но с разными видимыми результатами.

Введите следующую программу:

```
New File [Запись Number Array.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 23297
23297 ← 1 11 0
23300 ← 118 59 59 209 253 203 2 134 33 18 0 25
23312 ← 235 1 36 0 195 60 32 22 11 1 19 1 16 2
23326 ←На прocнекме Hayku Ecmь Мерро

17996 ← 23297 0
18000 ← 1
18001 ← прocnHayku
18011 ← 58 0 0 129

IX ← 18000
SP ← 17996
PC ← 2436
Trace
Cassette Recorder [Stop]

BASIC ← LOAD ""DATA a() ENTER
Cassette Recorder [Play]
```

После запуска алгоритма блок «Number array:» записался и считался. Для двойной проверки наберите натуральным способом:

```
PRINT a(1): PRINT a(1): RANDOMIZE USR (PEEK 23627+(PEEK 23628*256)+6)
```

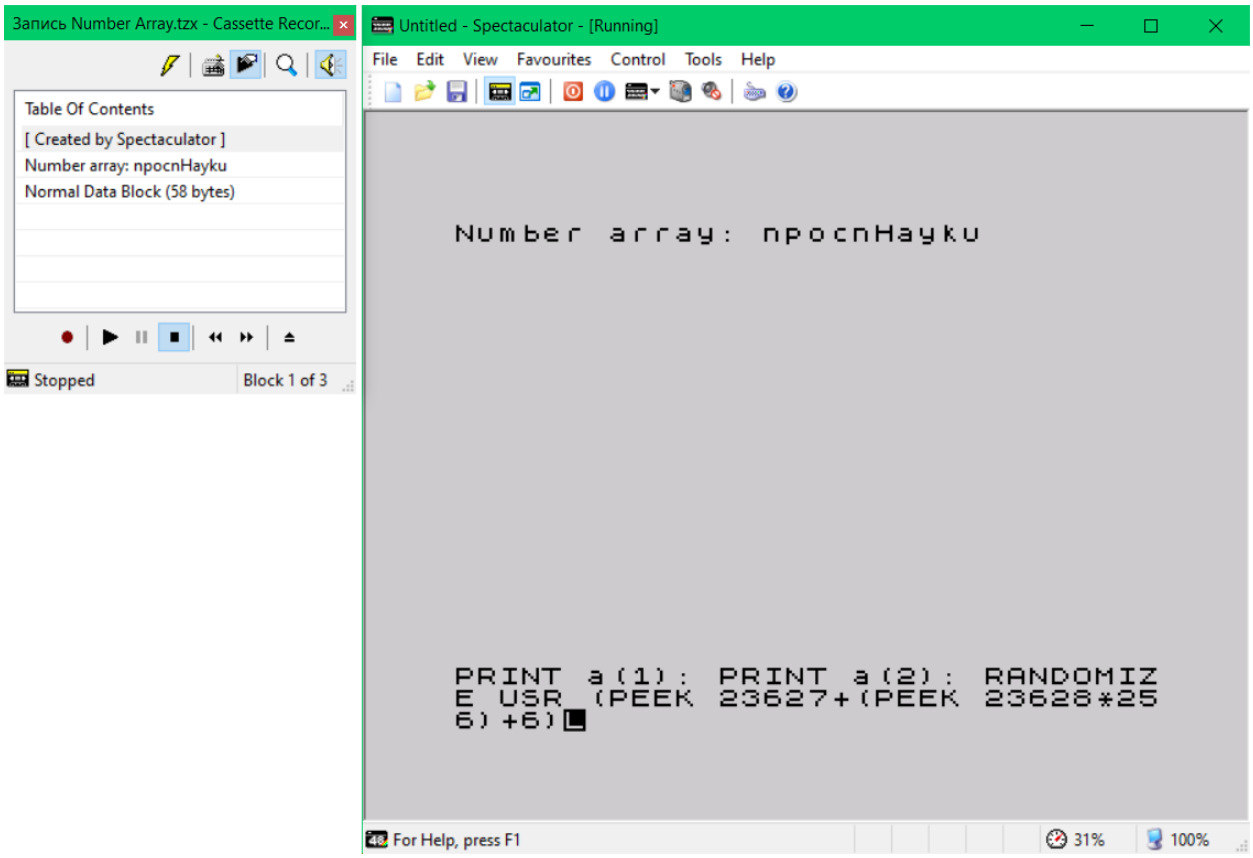


Рис. 448. Набор строки натуральным способом для проверки содержимого переменных.

Вводите строку:

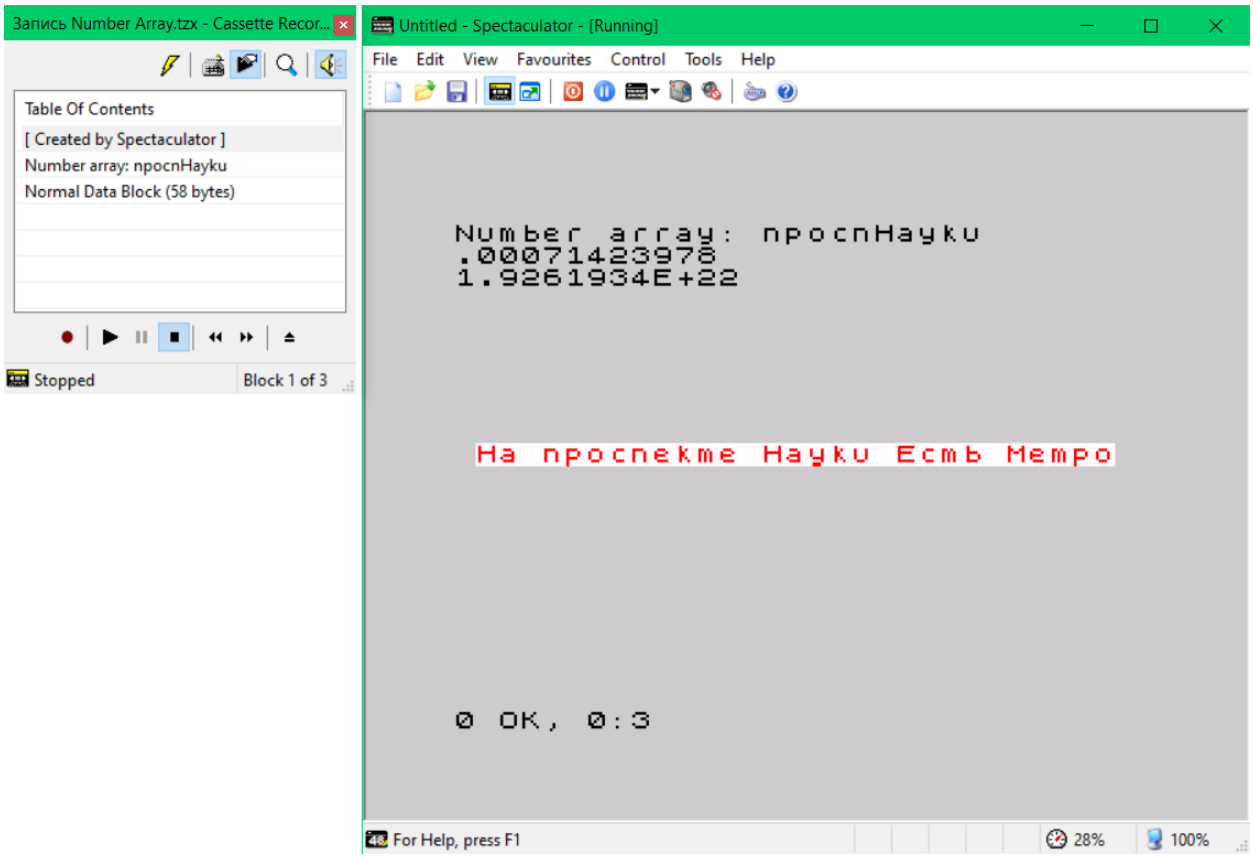


Рис. 449.

С утверждением поспорить трудно. Наземный вестибюль станции метро «Академическая» располагается на углу проспекта Науки и Гражданского проспекта.

После заголовка, по команде `PRINT`, выдались первые два числа из массива. Поскольку вместо чисел, там лежит фрагмент программы вывода на экран, то значения получились соответствующими. Следом из тех же самых ячеек запустилась программа вывода текста. В зависимости от введённых номерных BASIC строк, массив будет постоянно мотаться по памяти.

Команда `RANDOMIZE USR (PEEK 23627+(PEEK 23628*256)+6)` на ходу вычисляет текущее месторасположение области переменных, а следовательно, начало загруженного числового массива. На пустом компьютере числовой массив ожидаемо расположится с адреса 23755, а начала программы в машинном коде на 6 ячеек ниже, по адресу 23761:

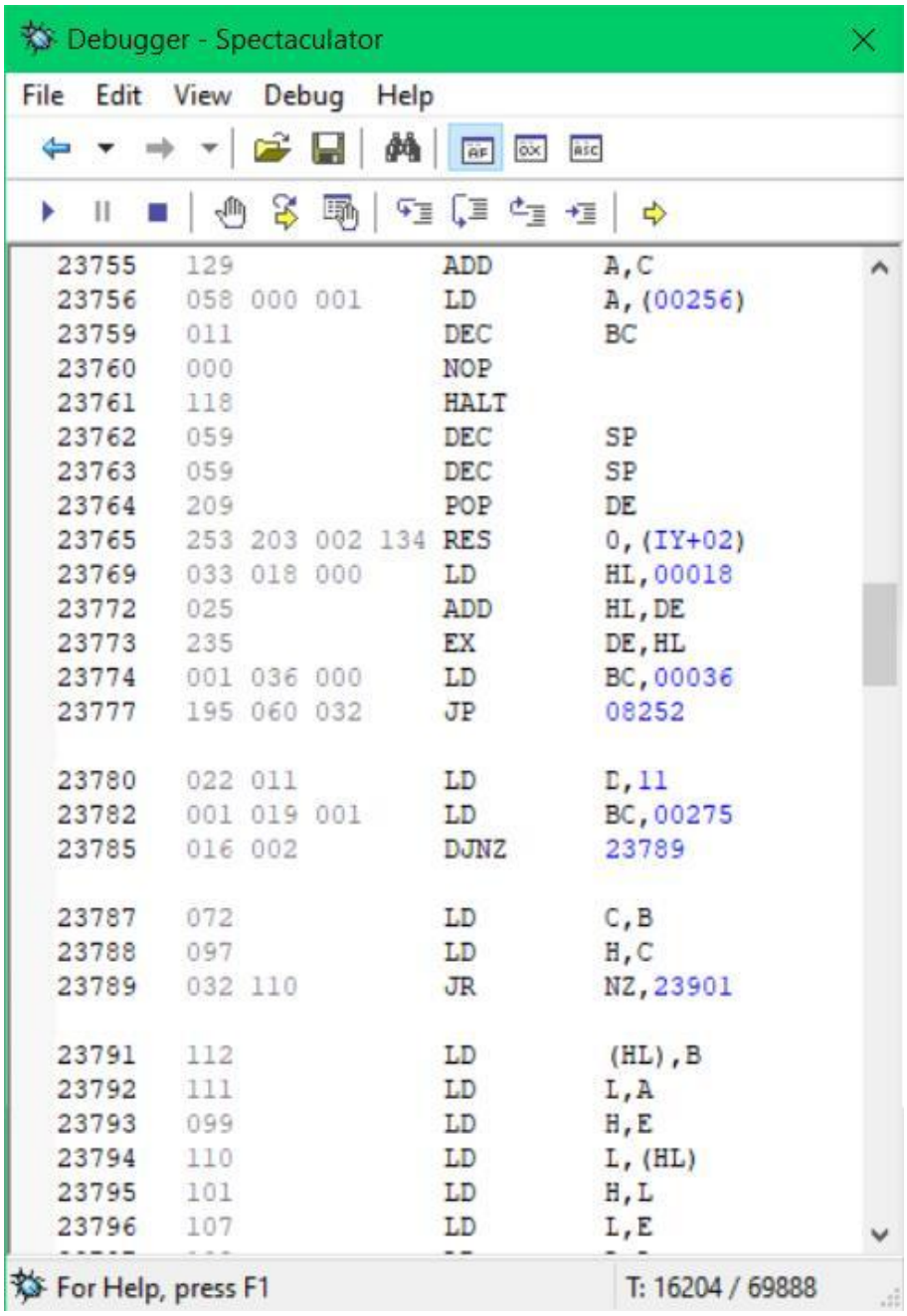


Рис. 450.

В текущем случае, команда `RANDOMIZE USR 23761` даст аналогичный результат.

Глава 61
ПерекосоMERGEвание программ

Краткое содержание: MERGE, NEXT-ONE, невидимые строки, предотвращение зависания

Вроде бы всё самое важное рассмотрел, и можно заканчивать обширный раздел «Внешняя память на магнитной ленте», но вспоминается странная фраза из книжки, которая вызывает неприятные ощущения недосказанности. И тянется она из прошлой части с главами о невидимых строках:

Из других любопытных трюков приведем еще два. Если в первой строке программы в ячейках, содержащих ее номер, поместить число 0, то такой оператор невозможно будет скопировать в нижнюю часть экрана в область редактора, а затем модифицировать. В другом конце программы возможна другая штучка. Допишем там строку 9999 REM и после установления ее адреса в обоих байтах, содержащих ее номер, разместим значение 255. Первый результат этого проявится при выводе программы на экран. Модифицированная нами строка не появится. Второй эффект, гораздо более ценный, проявится при попытке считывания такой программы с кассеты инструкцией MERGE. Spectrum "обижается" на пользователя и перестает реагировать на любые клавиши.

Рис. 451. Фрагмент старой книги «Тайники ZX-Spectrum».

Недосказанность была умышленной. Нет, я не собирался применять приёмы из прошлой книги и заминать непонятные моменты, просто не хотел забегать вперёд и решил дотянуть разбор вопроса до раздела записи и загрузки.

В связь невидимых строк с командой `MERGE` верили еще в стародавние времена, но каких-то весомых доказательств не выдвигалось. Предлагаю посмотреть на эту аномалию сейчас. Для этого можно набрать и ввести простую BASIC-программу с двумя строками, которая, как обычно, выводит на экран произвольный текст:

```
1 PRINT "    ВАРВАРА ЕСТ МАРС";: GO TO
16384
16384 PRINT " В МЕТРО"
```

Первая часть или строка программы будет видимая, а вторая невидимая. Поскольку блок «`Program:`» не умеет автоматически запускать строки от 16384 и больше, то видимая строка будет запускать невидимую:

```
New File [Видимое и невидимое.tzx]
Cassette Recorder [Record]
Debugger
Dec
Go To 23755

23755 ← 0 1 38 0 245 34 32 32 32 32
23765 ←ВАРВАРА ЕСТ МАРС
23781 ← 34 59 58 236 49 54 51 56 52 14 0 0 0 64
23795 ← 0 13 64 0 12 0 245 34 32 66 32 77 69 84
23809 ← 80 79 34 13

17996 ← 23755 0
18000 ← 0
18001 ← 32 19 1 66 65 80 66 65 80 65
18011 ← 58 0 1 0 58 0

IX ← 18000
SP ← 17996
```

```
PC ← 2436
Trace
Cassette Recorder [Stop]
```

А теперь внезапно! Натуральным способом наберите `MERGE ""`:

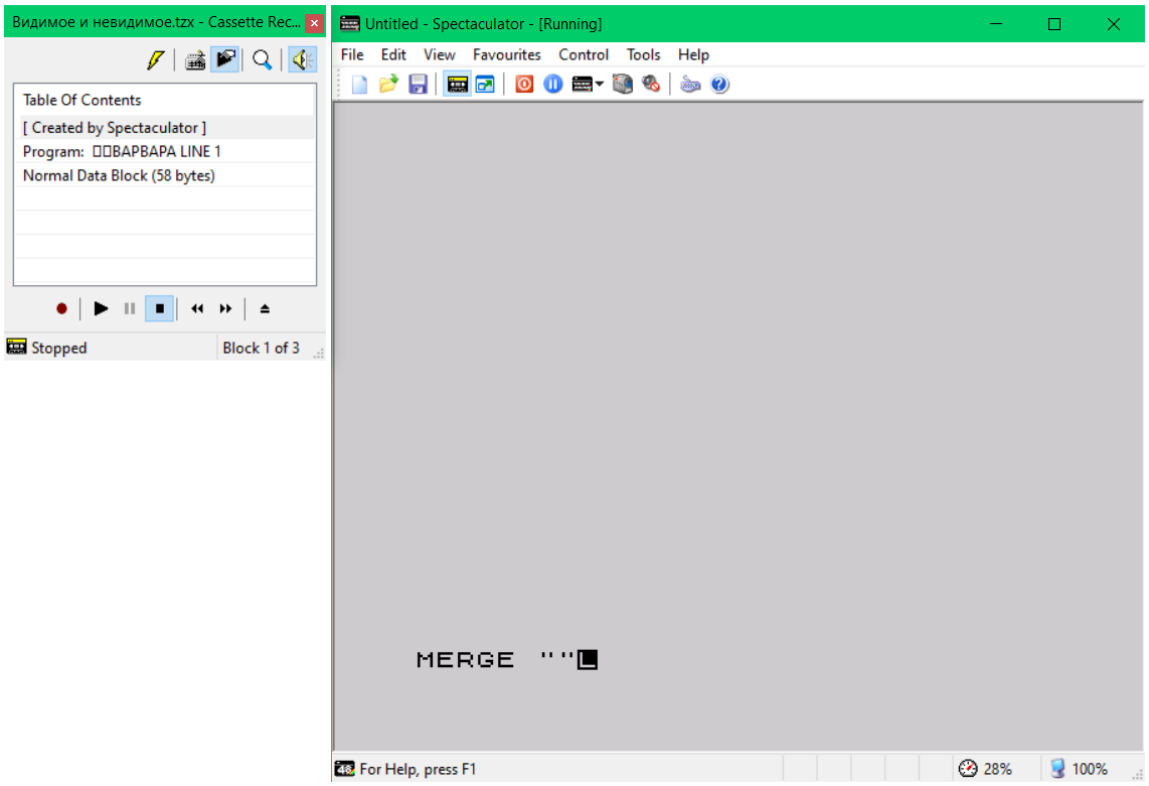


Рис. 452.

Введите и наблюдайте за происходящим. После загрузки на экране появляется живописная картина детства-юности, которую невозможно забыть:

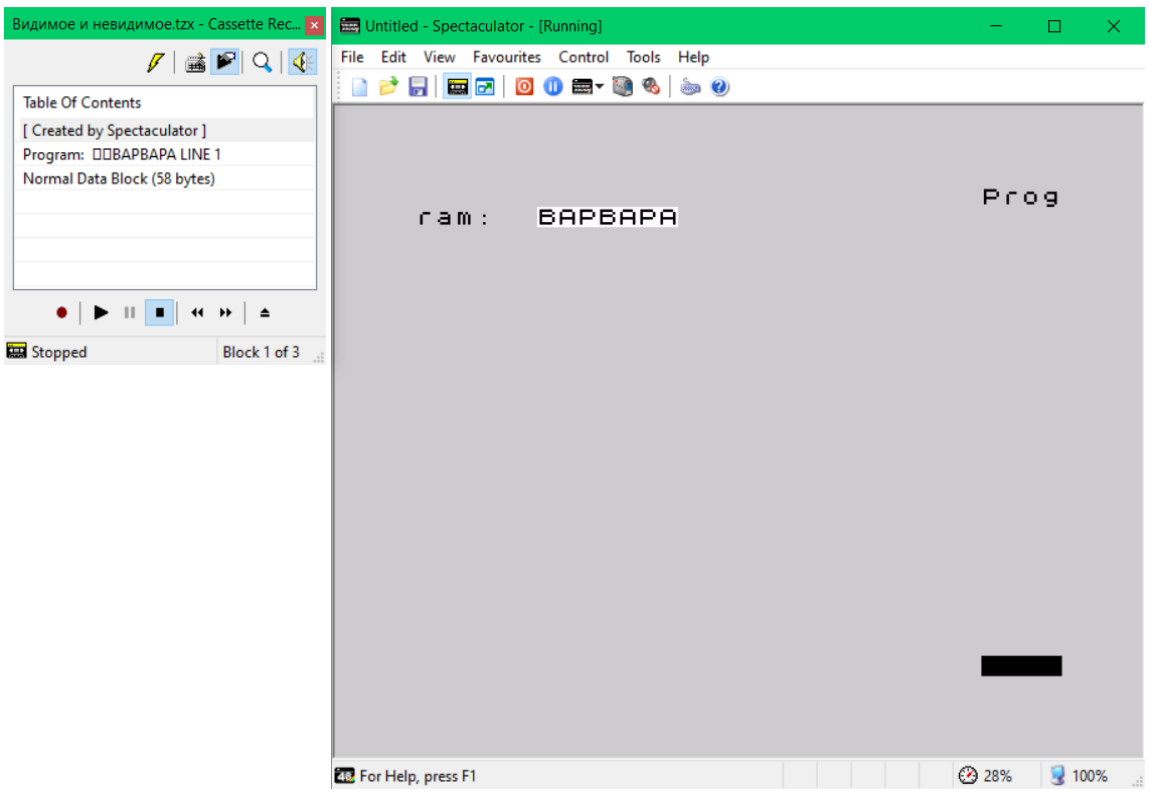


Рис. 453. Последствия загрузки командой MERGE строк с номерами 16384-65535.

Откройте и посмотрите:

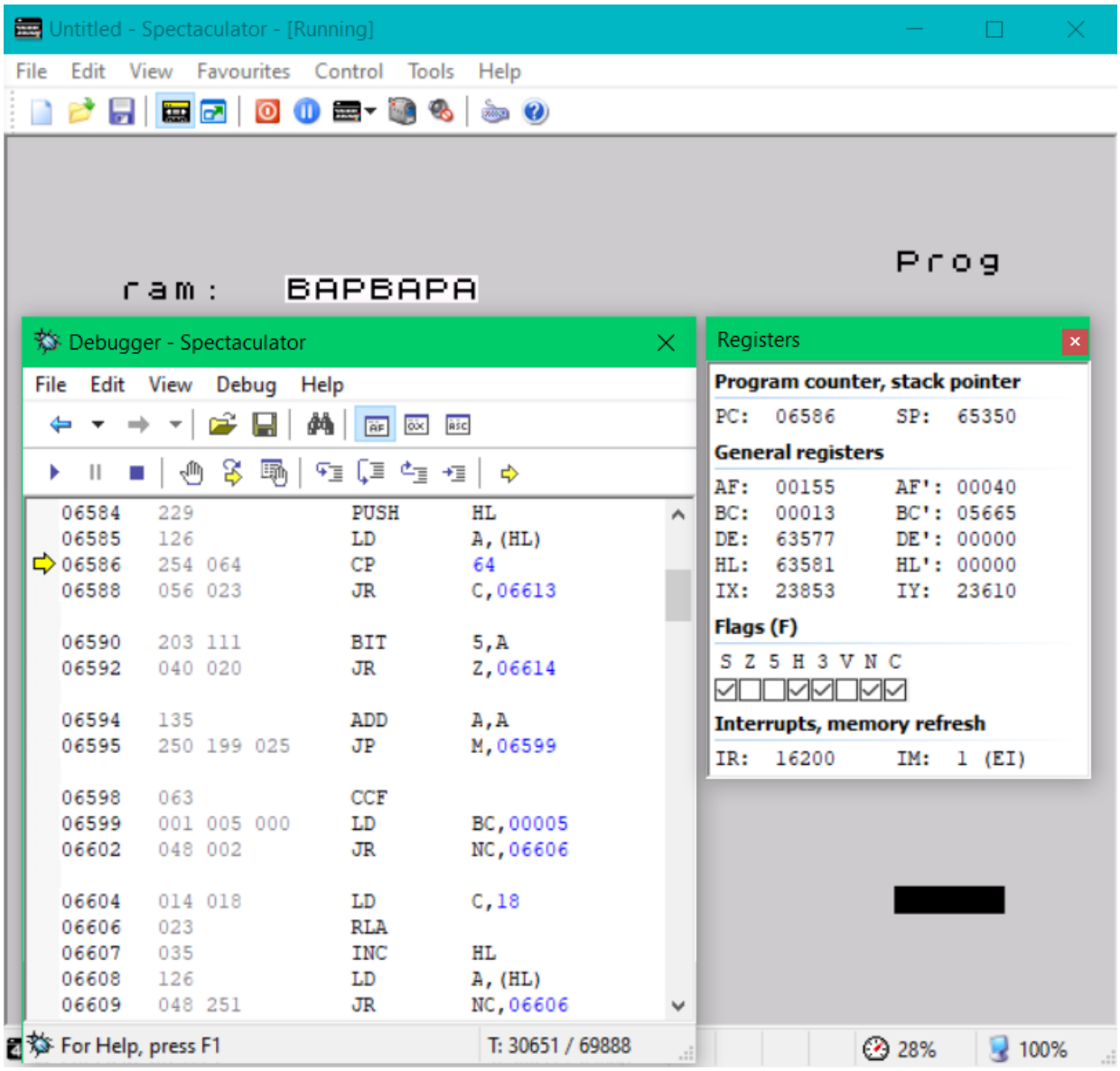


Рис. 454.

Ну конечно же, знакомая программа NEXT-ONE, к которой за помощью обращаются ещё как минимум 7 разных подпрограмм, в том числе и система программ MERGE!

Стабилизировавшийся цикл «MERGE зависания» достаточно простой:

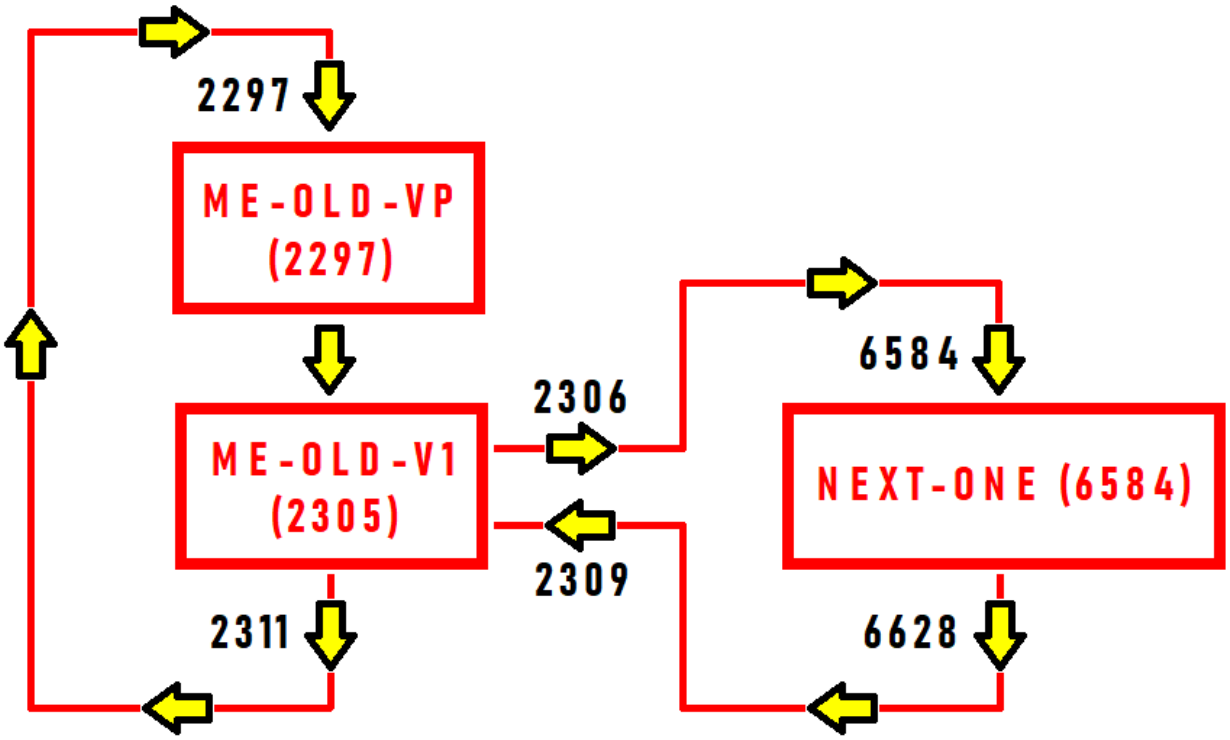


Рис. 455. Цикл зависания программы с невидимыми строками от команды MERGE.

Причина всё та же, что и в запуске строк с большими номерами. После загрузки программы командой `MERGE ""`, когда идёт первое обращение к подпрограмме, `NEXT ONE` некорректно обрабатывает длину самой первой невидимой строки 16384+, ну а дальше происходит глобальный сбой.

Как с этим бороться, детально рассматривалось в прошлых главах. При ошибке вычисления невидимой строки достаточно отловить Стрелочку на первом проходе по этой подпрограмме и выставить C ☑ галочку «C» на адресе 6588, принудительно заставив пройти по условию «`JR C,6613`».

В данной ситуации предпринимать что-либо, уже поздно, потому что в системных переменных произошли необратимые изменения. Предотвращать ситуацию нужно до момента попадания Стрелочки на эту подпрограмму, поэтому предлагаю очередной раз повторить загрузку, но теперь рассмотреть с позиции команды `MERGE`:

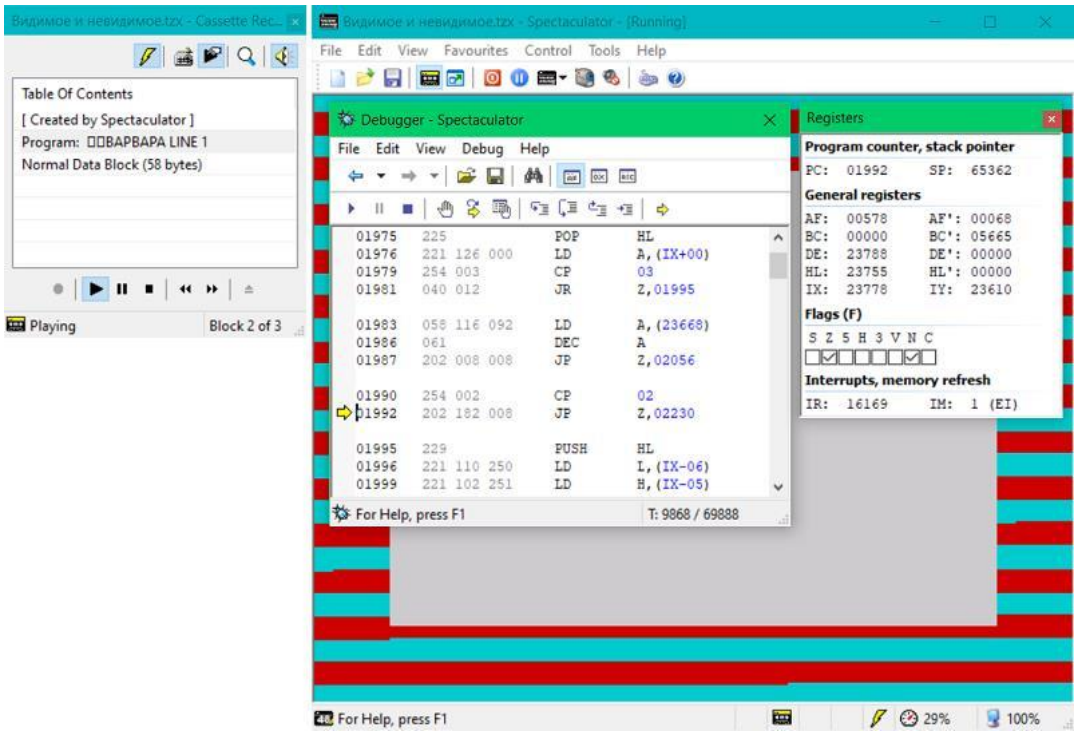


Рис. 456. Рассортировка по разным типам загрузки.

Путь загрузки по команде `MERGE ""`, отсоединяется с адреса 1992, сразу после `LOAD ""`:

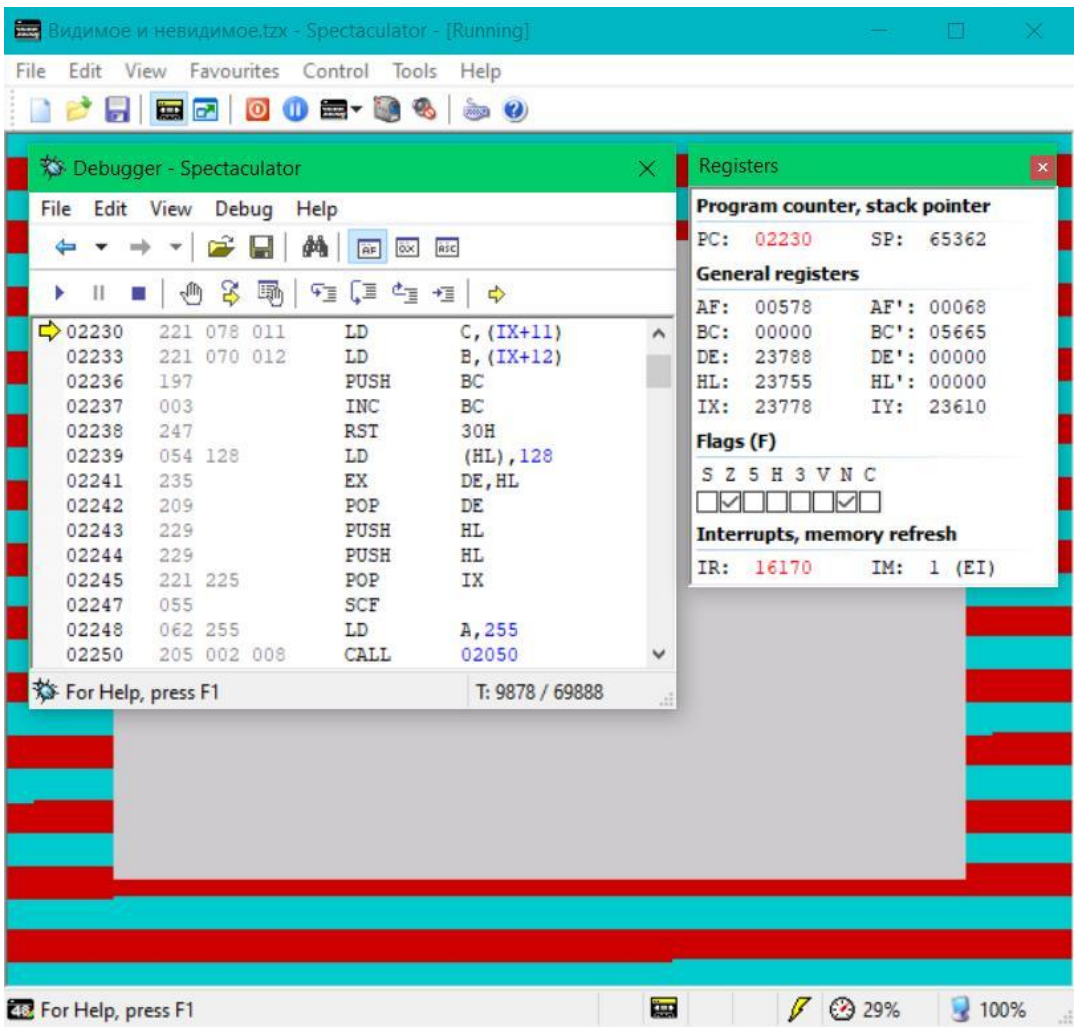


Рис. 457. Подпрограмма ME-CONTRL. Подготовка параметров для загрузки блока данных с BASIC.

После отделения сортов загрузки, Стрелочка попадает в программу ME-CONTRL по адресу 2230. Там берётся длина загружаемой программы, прибавляется единица для маркера и сразу за заголовком в рабочей области создаётся место для временного сброса данных. В конце предполагаемой программы, перед ENTER, выставляется маркер «128». Задав в «IX» стартовый адрес, а в «DE» длину, вызывается программа загрузки блока данных LD-BYTES (1366) транзитом через адрес 2050 (синяя точка автостарта).

После успешного считывания, Стрелочка выходит из подпрограммы загрузки. С адреса 2253 начинается подведение итогов:

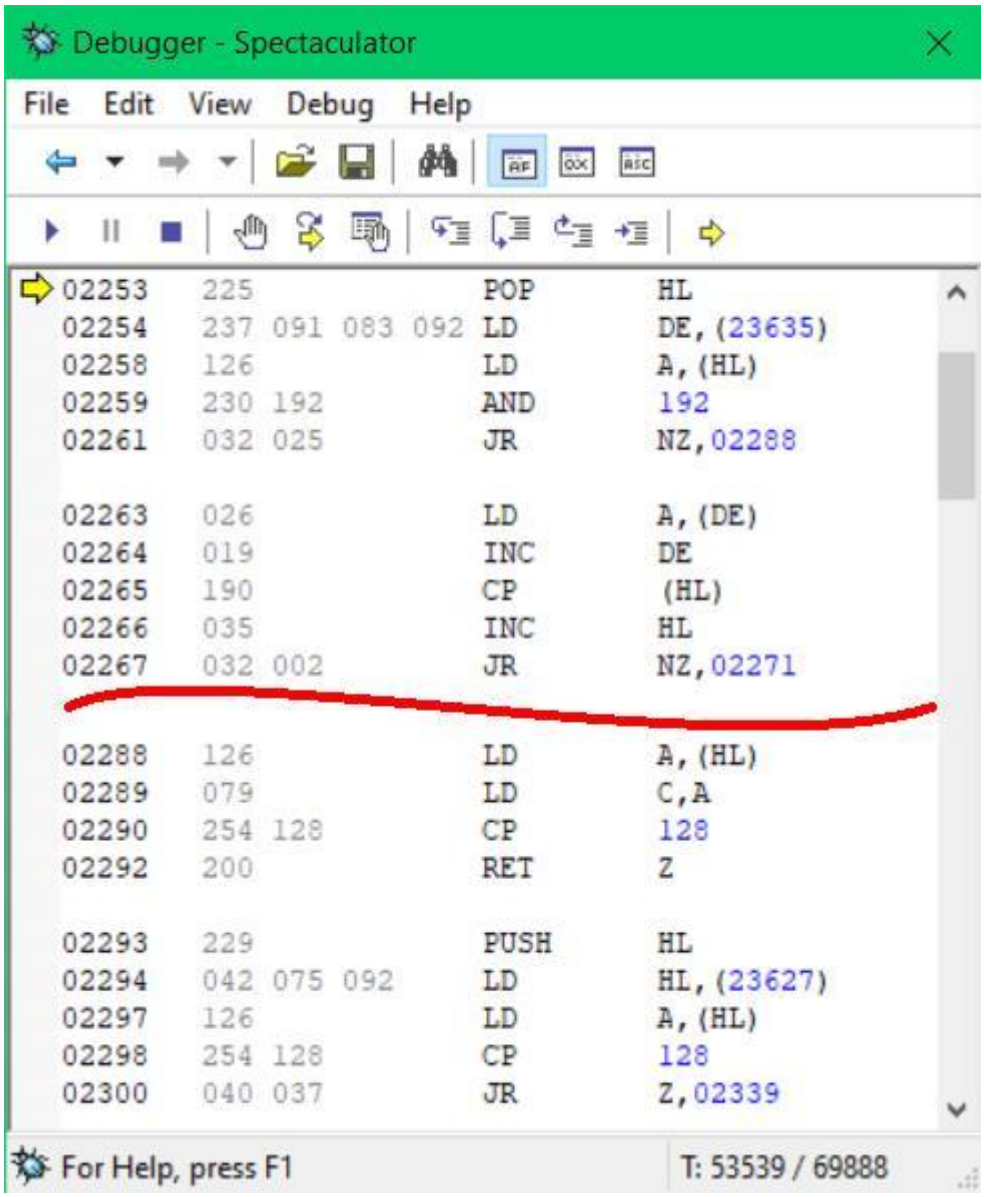


Рис. 458. Подготовка к присоединению новых данных после загрузки командой MERGE.

После загрузки начинается сравнение данных и построчный перенос из [WORKSP] в [PROG]. Восстанавливается адрес начала считанной программы и берётся первый символ, который по расчётам разработчиков должен быть номером строки или началом переменных. Если число меньше 64, значит это первый байт номера строки. В таком случае, номер строки по адресу [PROG], сверяется с текущим номером строки из [WORKSP]. Если номера совпали, то происходит подготовка к замещению строк, если считанная строка имеет уникальный номер, то она просто добавляется к имеющейся программе.

Настал момент, когда строка №1 успешно перенесена в область [PROG] и подошла очередь проанализировать невидимую строку №16384. Стрелочка возвращается на

исходную позицию и берёт для анализа старший байт строки. Увидев число 64 (64*256=16384), она думает, что это переменная и в полной уверенности переходит на адрес 2288 для дополнительной проверки.

В ME-VAR-LP (2288) производится прямая проверка на окончание области переменных в новом считанном куске. Если там маркер «128», значит, программа кончилась и можно производить выход на поверхность с выводом сообщения «0 OK». (*Таким образом, программа с невидимой строкой 32768 от команды MERGE не зависнет и выйдет в BASIC, но и не запишется прим. авт.*).

Поскольку число «64» не является маркером, то выполнение программы продолжается. Следом в ME-OLD-VP начинается проверка на маркер «128» в старом куске программы, но эти действия уже не имеют никакого смысла. При любом значении «A», рано или поздно, Стрелочка попадает на программу NEXT-ONE (6584) для вычисления длины текущей строки или переменной.

А вот тут уже начинается весело, потому что, рассчитав длину невидимой строки по шаблону переменной, начинается непоправимый сбой. Он вызывает цепную реакцию, в результате которой засоряется память и повреждаются системные переменные. Дальше спасать что-либо уже бесполезно и проще нажать сброс.

Что можно сделать для предотвращения этой ситуации?

Способ первый, установить ☑ галочку «Z» во время нахождения Стрелочки по адресу 2292. Таким образом, можно загрузить все видимые строки и спасти от зависания, но потерять данные переменных и невидимых строк.

Второй способ предпочтительнее. С помощью него командой MERGE "" можно загрузить данную программу целиком и при этом также избежать зависания.

Давайте дадим спокойно доесть Варваре шоколадку в вагоне метро и характерно облизать пальцы. Для этого выполните следующий алгоритм:

```
Open File [Видимое и невидимое.tzx]
Debugger
Dec
Add Breakpoint 2288
Trace
BASIC ← MERGE ""ENTER
Cassette Recorder [Play]
PC ← 2263
Add Breakpoint 6588
Trace
AF ← AF+1
Remove Breakpoints 2261, 6588
Trace
BASIC ← RUN ENTER
```

После загрузки, вместо ожидаемого зависания, на экране появилось «0 OK»:

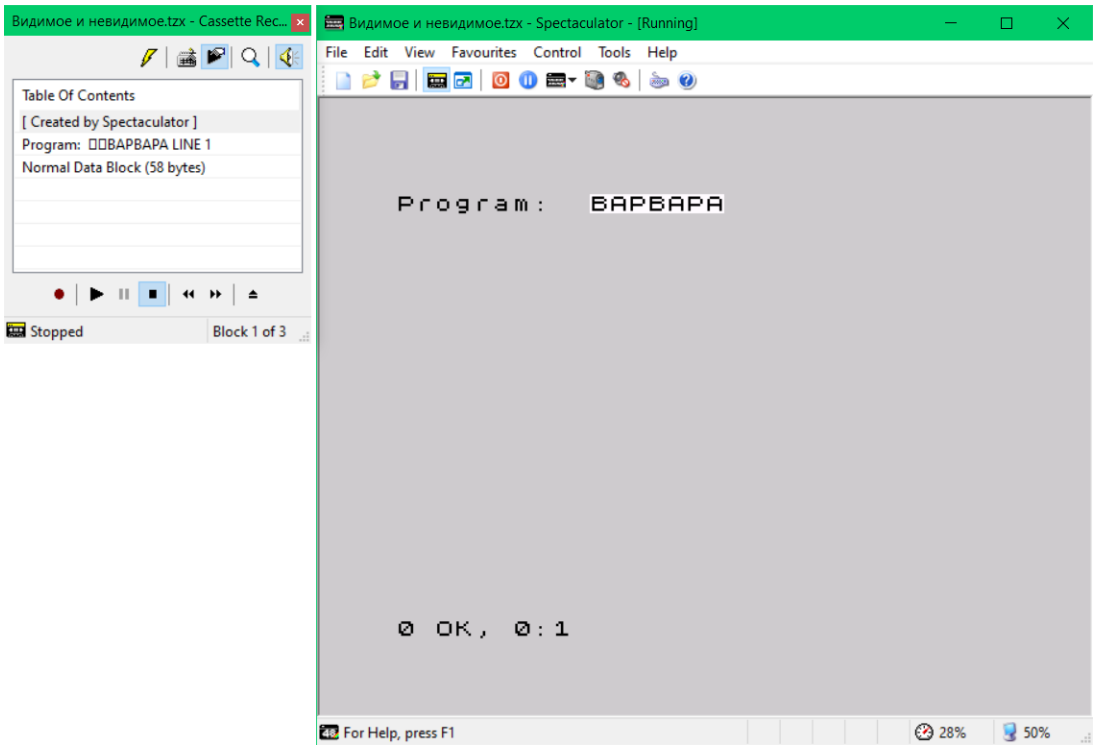


Рис. 459. Успешное считывание программы с невидимыми строками командой MERGE.

Запустите программу для проверки RUN'ом и на экран выведутся оба куска фразы:

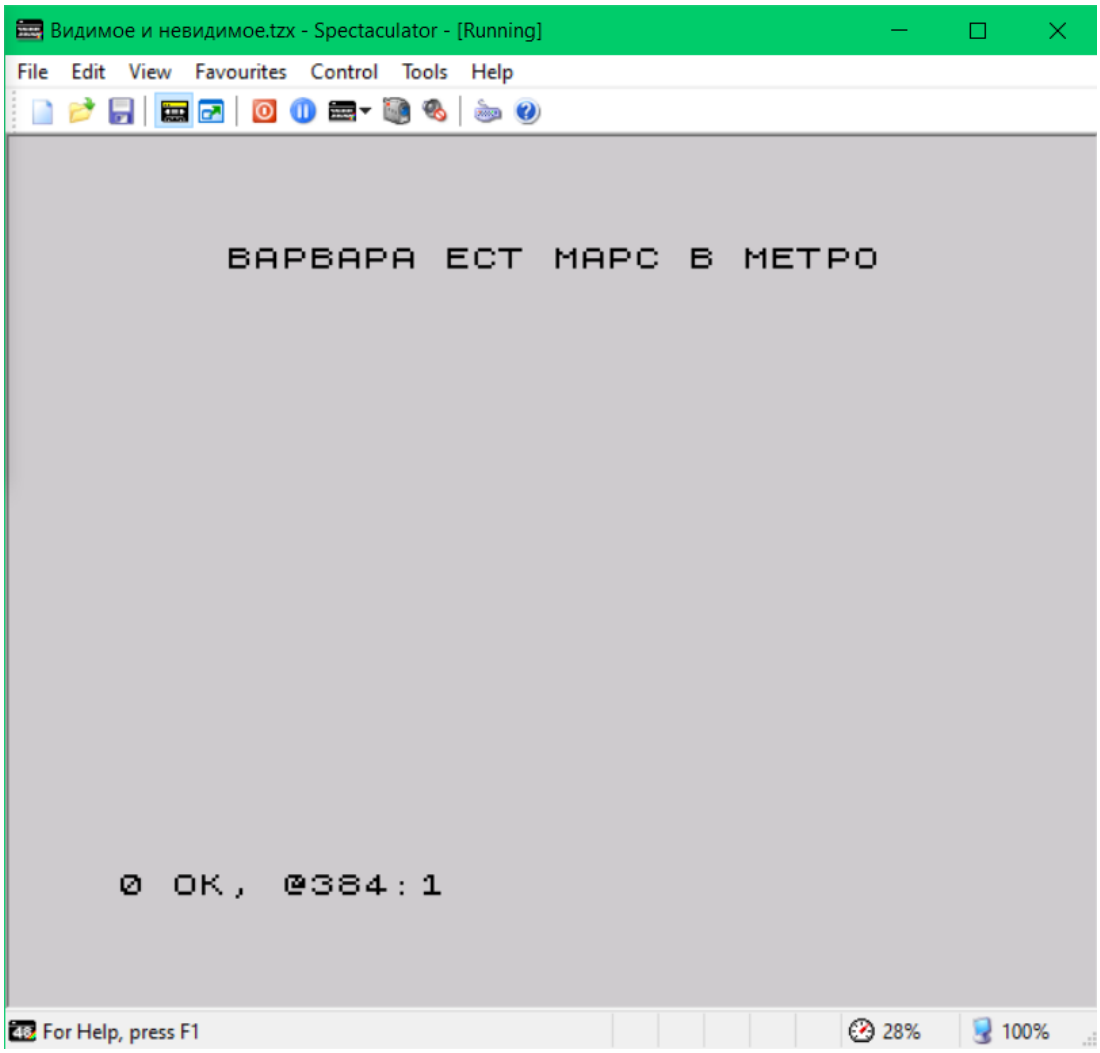


Рис. 460. Проверочный запуск считанной программы командой RUN.

Невидимая строка загрузилась.

В заключение главы и раздела приведу алгоритм имитации команды MERGE "" на языке «*God Mode*»:

```
Open File [*.tzx]
Debugger
Dec
Go To 23400
23400 ← 0 255 0 0 0 0 0 0 0 0 0 0 0 0 0 0
23610 ← 255
23613 ← 65364
23668 ← 3              ; полуфабрикат кода операции MERGE ""
65364 ← 4867

HL ← 0
IX ← 23400
SP ← 65364
PC ← 1889
Trace
Cassette Recorder [Play]
```

А следующий алгоритм еще более экзотический и эксклюзивный. Он загрузит блок данных методом MERGE без заголовка:

```
Open File [*.tzx]
Debugger
Dec
23610 ← 255
23613 ← 65364
65364 ← 4867

BC ← 47                ; длина загружаемого блока
SP ← 65364
PC ← 2236
Trace
Cassette Recorder [Play]
```

И вот на этом моменте я хочу закончить 2-ю синюю часть книги.

ЧАСТЬ III. ЗЕЛЁНАЯ ПОСТКУЛЬМИНАЦИОННАЯ
«И ЖИЛИ ОНИ ДОЛГО И СЧАСТЛИВО…»

«И жили они долго и счастливо…». Самое обидное, что так заканчивается большинство типичных сказок. И действительно, тихую и спокойную жизнь принято считать унылой и недостойной внимания, а говорить стоит только о движухе и страданиях. Но только не у меня.

К этому моменту, я узнал 1146% от запланированного. В ходе написания книги возникло столько внезапных открытий, что придётся создавать этот небольшой раздел с пародиями на концовку типичного самоучителя. Хотя, это даже будет логично. Книга с красным и синим разделом, без зелёного будет смотреться неполноценно, поэтому хоть что-то нужно чирканути для приличия. Как оно получится – жизнь покажет.

Глава 62
Программирование цветных звуков

Краткое содержание: цветные полоски на рамке

Эта глава должна была стоять перед «*Внешней памятью на магнитной ленте*», но по объективным причинам пришлось ее сместить в последний раздел.

Звуки так звуки. Для разминки предлагаю набрать алгоритм программы, выводящий аналог BASIC команды `BEEP 1,0`:

```
Debugger
Dec
Go To 23613
23613 ← 65364
65364 ← 4770

DE ← 261
HL ← 1642
SP ← 65364
PC ← 949
Trace
```

«У-у-у-у-у-у» – уныло ответил Spectaculator, сверкая серым экраном.

Юрец, а чего картинки никакой не будет? Ну а как я сделаю картинку звука! Я согласен, что это непорядок. В предыдущих главах иллюстрации вставлялись на каждый чих, а тут… А впрочем, есть идея!

Для начала предлагаю рассмотреть эту самую программу **BEEPER**, которая отвечает за одноимённую команду и не только.

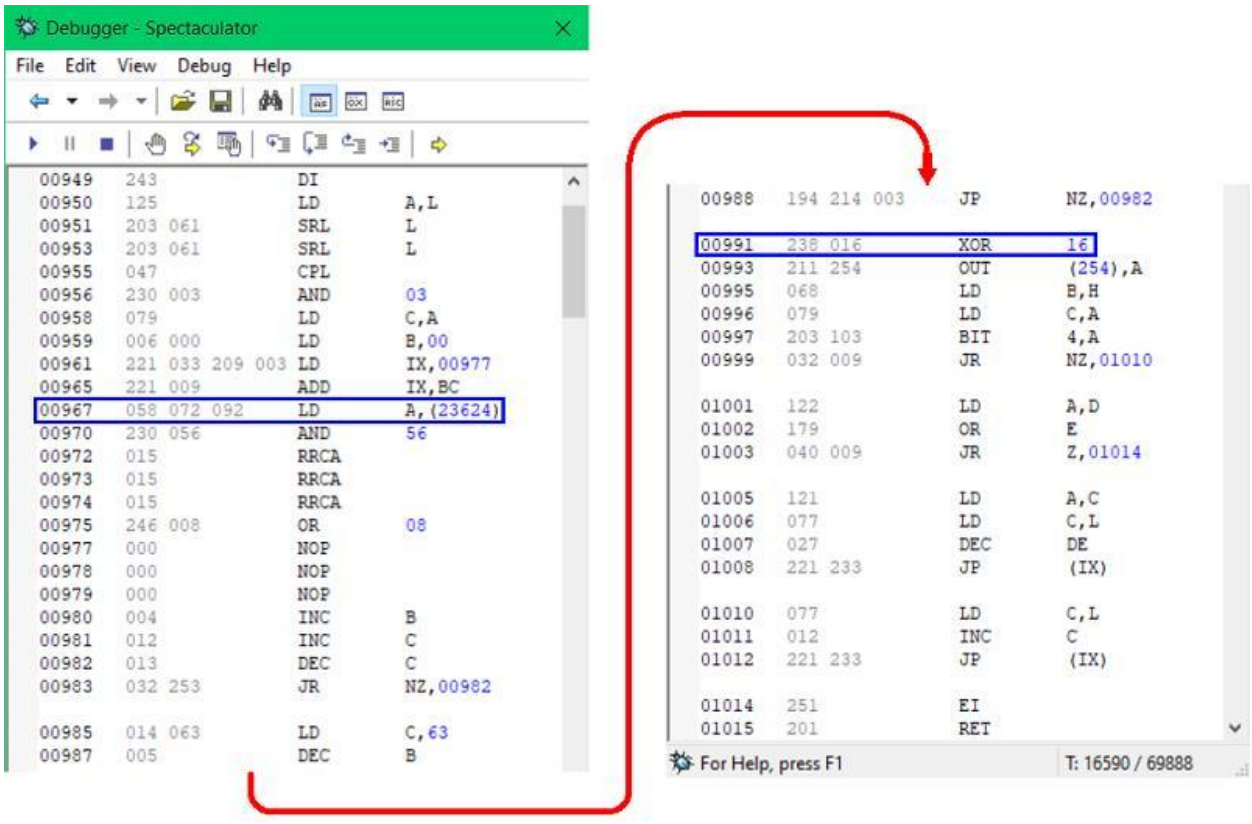


Рис. 461. Подпрограмма BEEPER.

Да-да! За щелчок от нажатия клавиши также отвечает она. Как видно, программа не сильно запутанная. Так уж устроен был реальный компьютер, что звук, цвет рамки и передача данных засандалили в один байт. Кому-то удалось впихнуть невпихуемое.

В данной программе, цвет рамки всеми силами блокируется, чтобы только предотвратить красивый визуальный эффект. Для того, чтобы вернуть цветомузыку, нужно по адресу 967 в «A» задать число от 8 до 15, а в 992 в `XOR` выставить от 16 до 23.

Можно, конечно поменять значения в ПЗУ, как это объяснялось в книге 2013 года, но есть более полезный вариант. Достаточно просто дизассемблировать эту программу, исправить переходы, модифицировать и запускать с любого адреса.

Итак, 10 минут работы, и появляется такая готовая цветомузыкальная программа на ассемблере *EmuZWin*, которую можно записать в любой адрес памяти. Например, в 40000:

```
; Вывод ноты BEEP 1,0 с цветными полосками на рамке

            ORG  40000

            LD HL, 1642       ; Длина звука
            LD DE, 261        ; Тональность
beeper:     DI
            LD   A, L
            SRL  L
            SRL  L
            CPL
            AND  3
            LD   C, A
            LD   B, 0
            LD   IX, BEIX3
            ADD  IX, BC
            LD   A, 4         ; Цвет первой полосы во время звучания (0...7)
BEIX3:      NOP
BEIX2:      NOP
BEIX1:      NOP
BEIX0:      INC  B
            INC  C
BEHLLP:     DEC  C
            JR   NZ, BEHLLP
            LD   C, 63
            DEC  B
            JP   NZ, BEHLLP
            XOR  18     ; Шаг чередования цвета для 2-й полоски (16...23)
            OUT  (254), A
            LD   B, H
            LD   C, A
            BIT  4, A
            JR   NZ, BEAGAIN
            LD   A, D
            OR   E
            JR   Z, BEEND
            LD   A, C
            LD   C, L
            DEC  DE
            JP   (IX)
BEAGAIN:    LD   C, L
            INC  C
            JP   (IX)
BEEND:      EI
            LD   A, (23624)         ; Восстановление исходного цвета рамки
            AND  56
            RRCA
            RRCA
            RRCA
            OR   8
            OUT  (254), A
```

```
RET                         ; Выход в BASIC
```

В рамках данной книги, уместен будет машинный код, поэтому вместо ассемблера введите нижеследующую программу в привычном God Mode:

```
Debugger
Dec
Go To 23613
23613 ← 65364

SP ← 65364
PC ← 40000

Go To 40000
40000 ← 33 106 6 17 5 1 243 125 203 61 203 61
40012 ← 47 230 3 079 6 0 221 33 90 156 221 9
40024 ← 62 4 0 0 0 4 12 13 32 253 14 63 5
40037 ← 194 95 156 238 18 211 254 68 79 203 103
40048 ← 32 9 122 179 40 9 121 77 27 221 233 77
40060 ← 12 221 233 251 58 72 92 230 56 15 15 15
40072 ← 246 8 211 254 195 162 18
Trace
```

Запустите программу и:

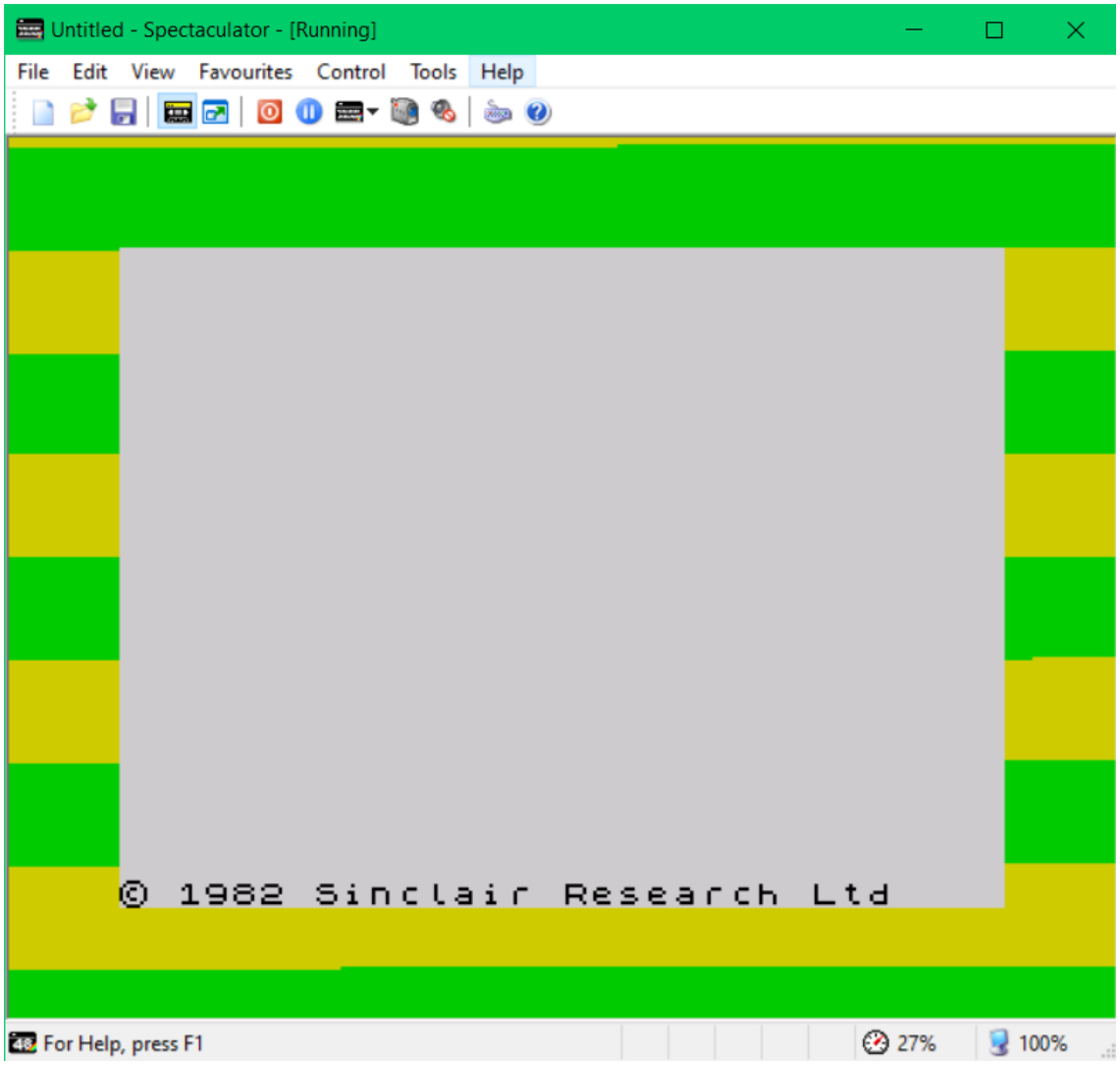


Рис. 462. Цветные полоски на рамке во время звучания аналога команды BEEP 1,0.

О какая красота! И теперь вы можете задавать цвета двум полоскам, независимо от текущего цвета рамки. Первый цвет разместится по адресу 40025. Туда можно вносить значения от 0 до 7. Второй цвет будет в комплексной переменной по адресу 40041, куда вносите значения от 16 до 23-х.

А где применяется цветомузыки на практике? Конечно же, в любимой загрузке с магнитофона, о которой не перестаю упоминать в этой книге и восхищаться с осени 1992 года! Это и есть та самая цветомузыка, когда во время приема данных по рамке пробегают сине-желтые полосочки.

И снова предлагаю вернуться к теме прошлого раздела, а именно к структуре программы, которая обслуживает BASIC команду `SAVE`.

Итак, на языке BASIC введена команда `SAVE "ТЕСТ"CODE 16384,6912`.

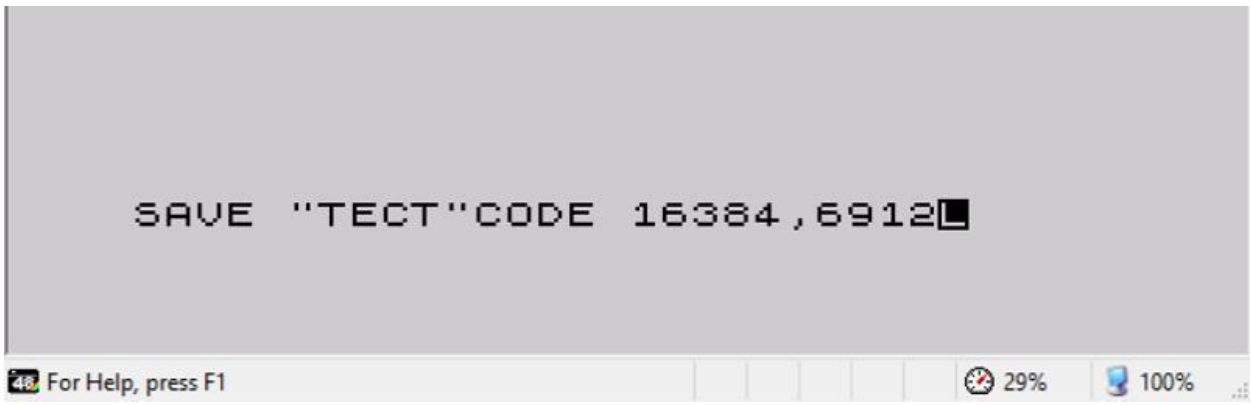


Рис. 463. Spectaculator. Нижний кусок экрана с командой.

Пропущу все то, что касается технической стороны записи и обсуждалось ранее, а сейчас остановлюсь на звуке.

Предположим, закончились последние приготовления и вот-вот начнётся запись блока данных. Предлагаю заглянуть в волшебный мир с адреса 1236:

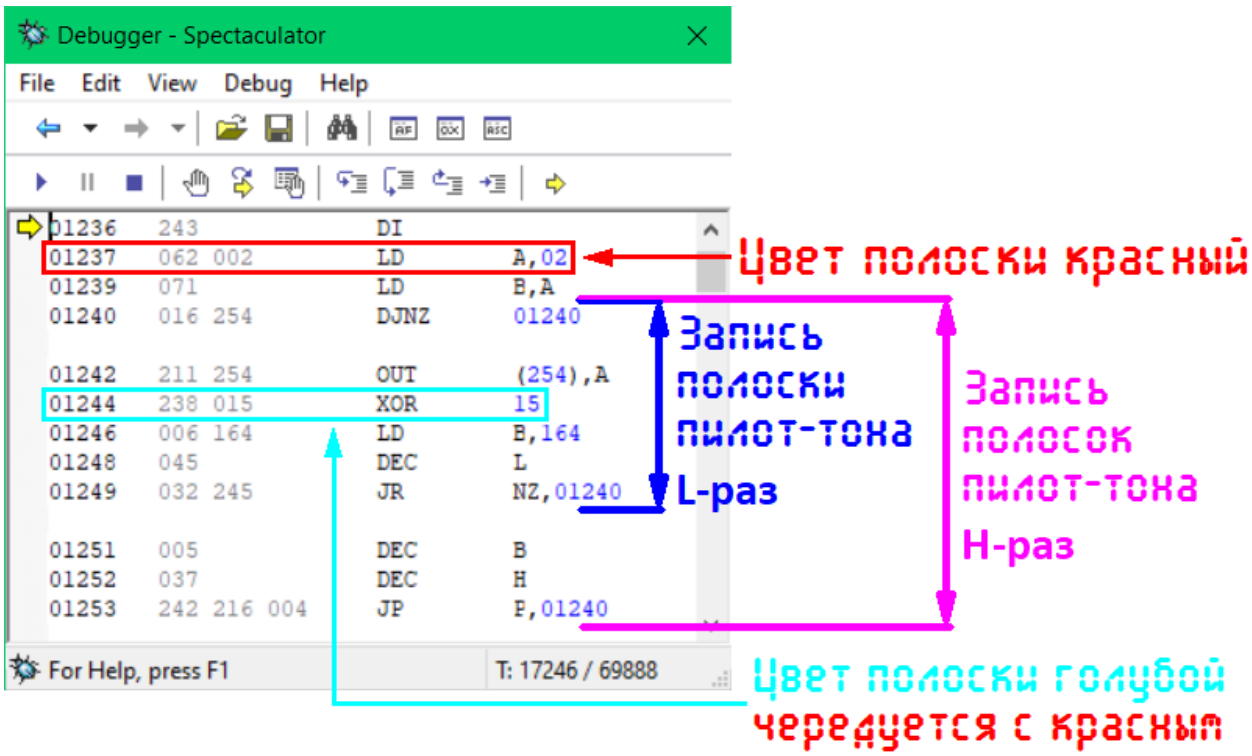


Рис. 464. Поблоковая генерация сигнала «Пи-и-и-и» (пилот-тон).

Это и есть программа генерации сигнала «Пи-и-и-и». Выставив цвет полоски 2 с битом выключенного сигнала (`1237 LD A,2`) формируется звук «Пи-и-и-и». По аналогии с нотой звука, командой `XOR 15` по адресу 244, значение «A» из 2-х циклически меняется на 13. Следом задаётся пауза в 164 единицы, которая записана в «B» (1246). В нагрузку к ней добавляется некоторое время на выполнение команд. По значению «L» цикл повторяется несколько раз. Таким образом, формируется короткий кусок сигнала «Пи-и-и-и» с постоянной частотой.

Дальше в «H» смотрится, сколько таких кусков нужно проиграть подряд (1252), чтобы получить нужную длину непрерывного гудения.

Следом идёт цементирующий одиночный импульс, который связывает «Пи-и-и-и» с идущими следом данными «Тшу-у-у-у»:

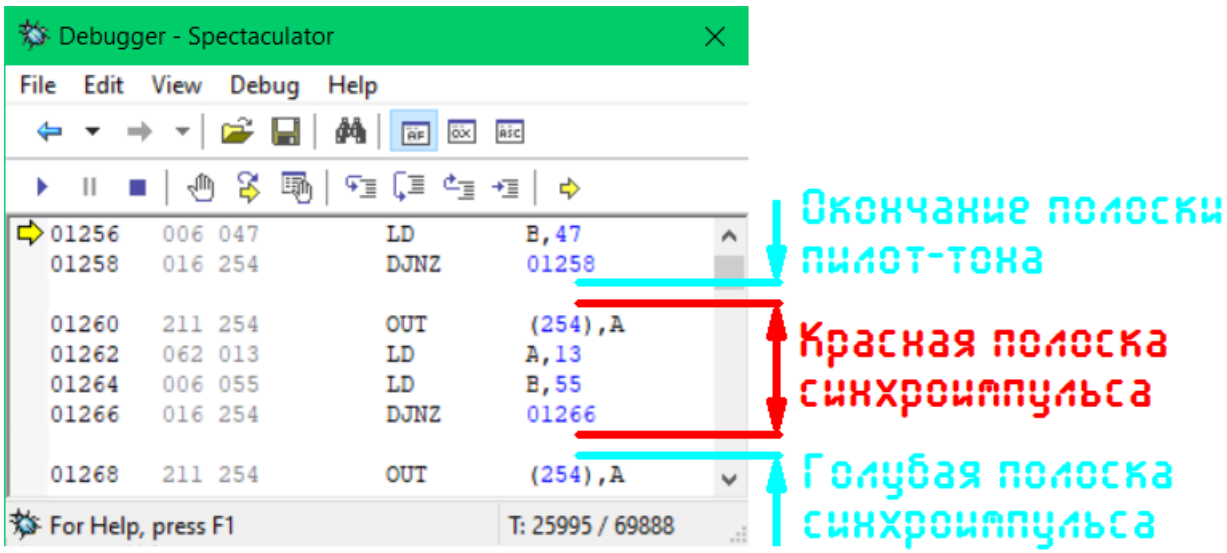


Рис. 465. Формирование цементирующего импульса.

Как видно, ради его генерации создана индивидуальная программка с двумя OUT'ами, которые и создают эту короткую звуковую волну. Она также формируется одиночным чередованием «2» и «13».

Начиная с адреса 1270, пока дописывается голубой связующий импульс, идет подготовка к проигрыванию данных.

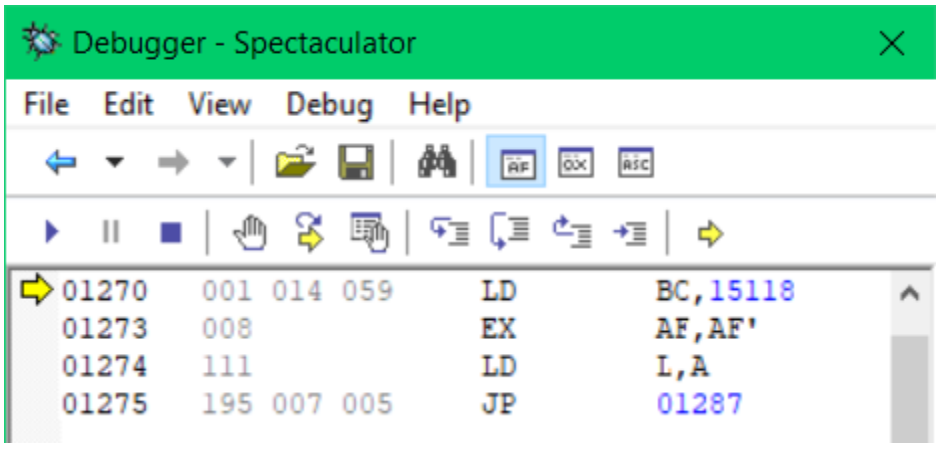


Рис. 466. Задание задержки и будущего цвета второй полоски для загрузки данных.

В «BC» заносится комплексное значение 15118, что следует раздельно читать как B=59 и C=14. Первое, это очередной фрагмент задержки для формирования нужной структуры сигнала. Второй – это желтый цвет будущей полоски + бит «Вкл» для формирования сигнала. Далее вспоминается первый скрытый байт идентификатора блока (`1273 EX AF,AF'`). Стартует отсчёт псевдоконтрольной суммы для будущих

записываемых байт. Следом после еще одной компенсационной задержки (`1300 DJNZ 1300`), с синей полоски, начинается запись байта идентификатора, положив начало проигрывания «Тшу-у-у-у».

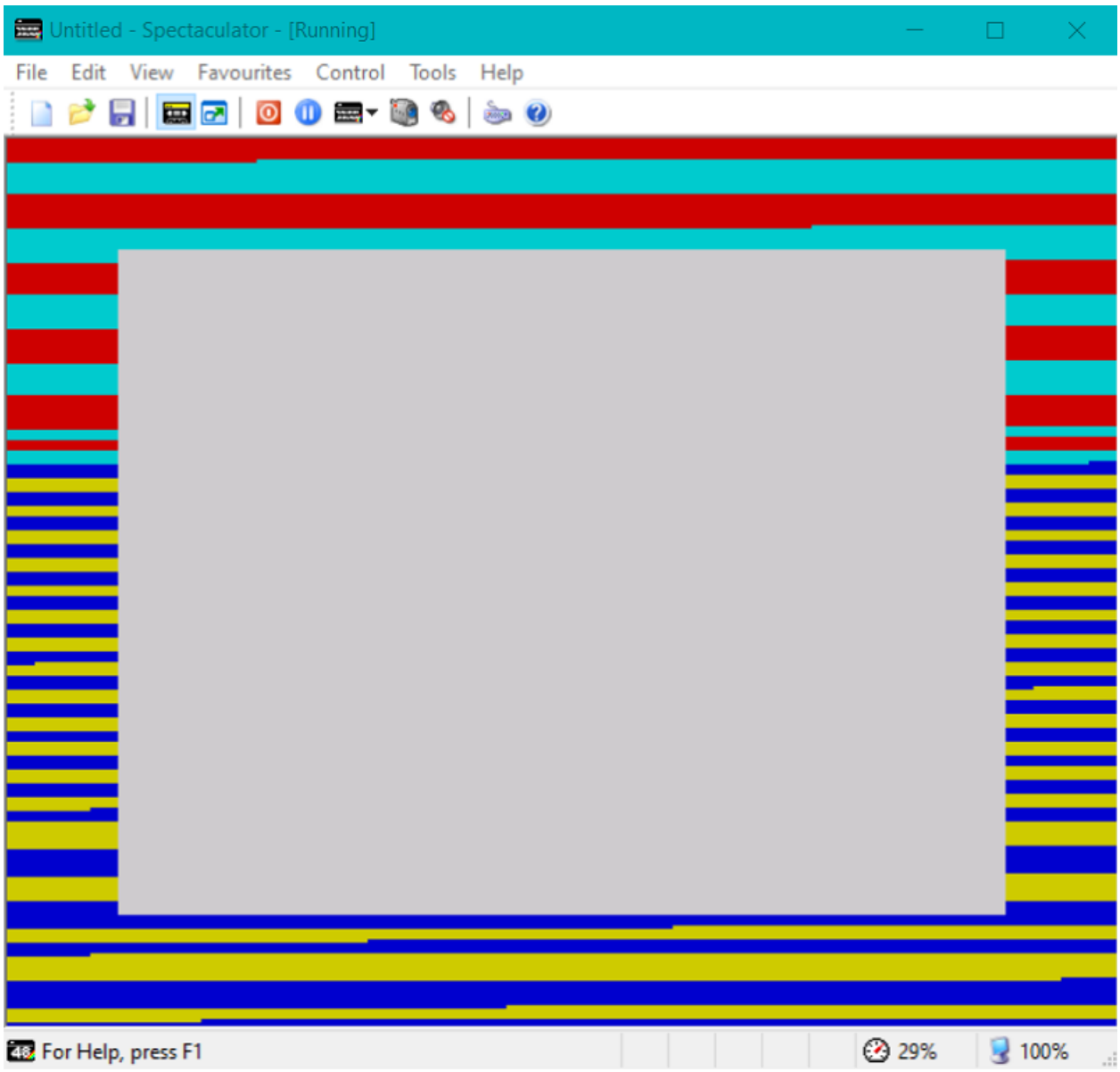


Рис. 467. Момент перехода сигнала «Пи-и-и-и» в «Тшу-у-у-у».

После записи служебного байта (0 или 255), Стрелочка попадает на адрес 1322, откуда начинается стабильный цикл проигрывания данных до самого конца. В рамках этой главы, он самый важный:

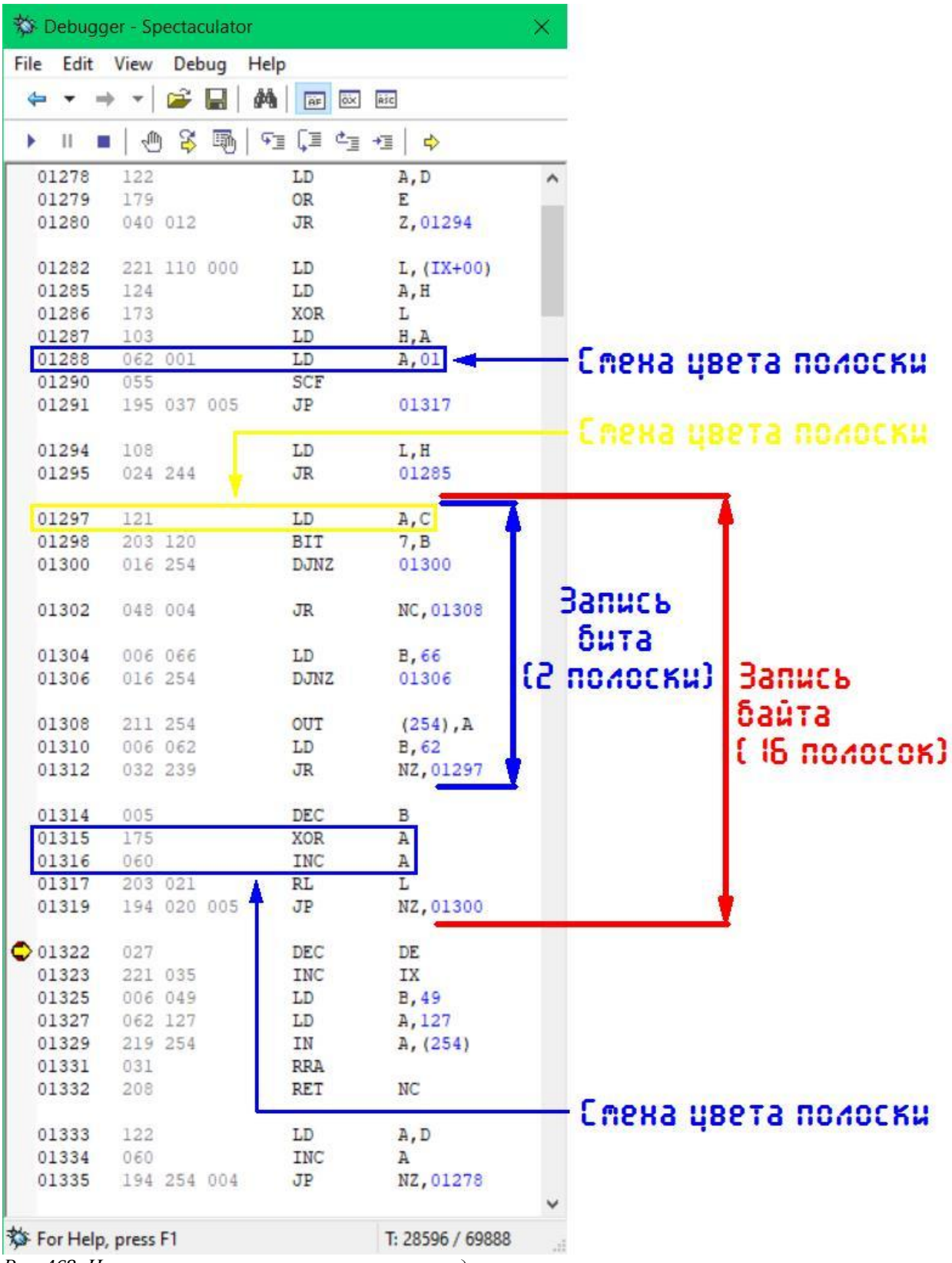


Рис. 468. Цикл проигрывания, записи и озвучивания данных из памяти.

Запись производится чередованием чисел 1 (бит «Выкл» + синий цвет рамки) и 14 (бит «Вкл» + желтый цвет рамки). Желтая полоска задаётся по адресу 1271, а синяя даже в двух местах. Первый раз ставится в 1288 (`LD A,1`). Второй раз исправляется по адресу 1315, командами `XOR A` и `INC A`.

Момент окончания записи идентификатора 255 и начала записи байта данных 179 выглядит вот так:

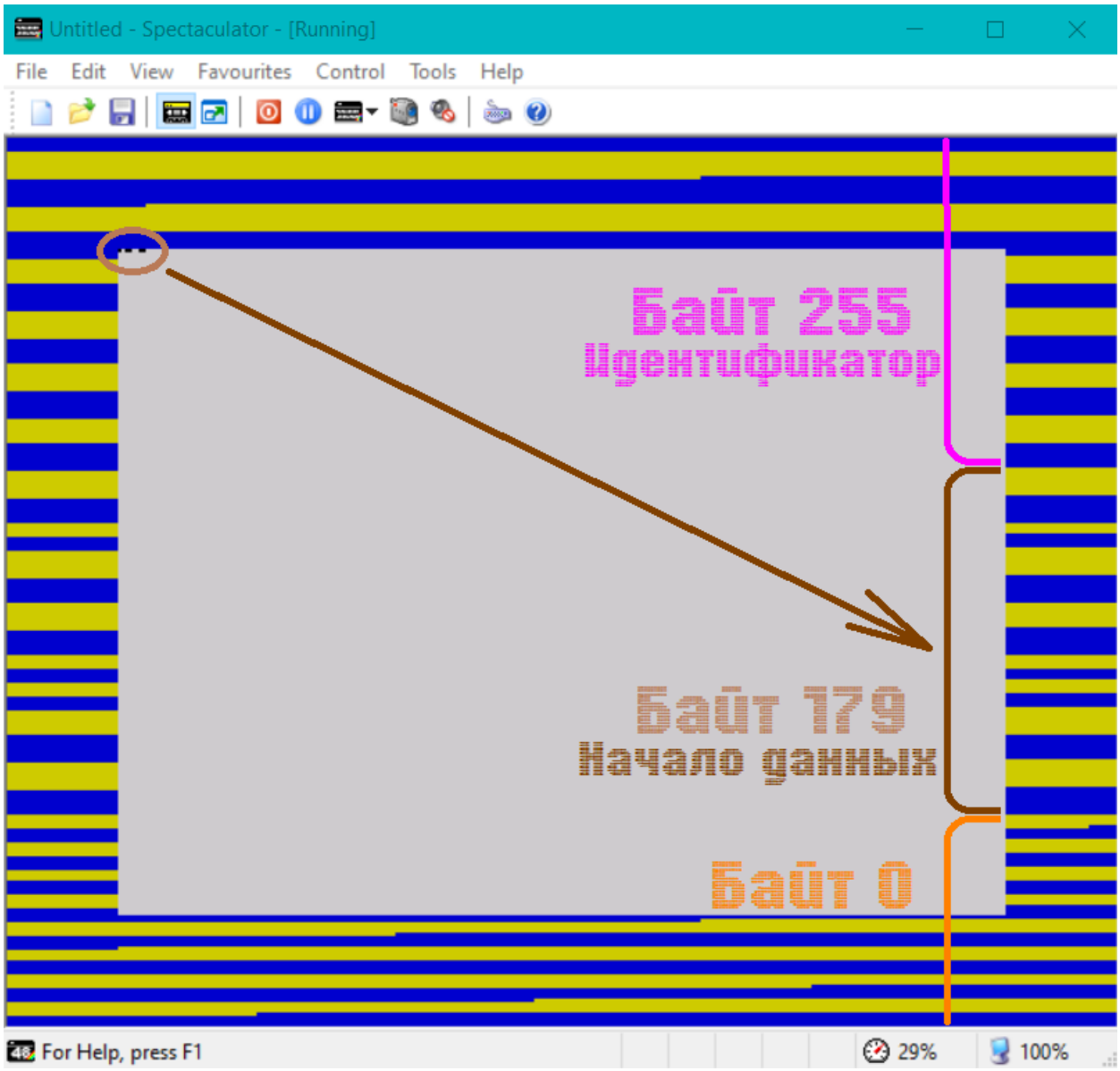


Рис. 469. Отображение битов данных полосочками во время загрузки.

Бит «0» отображается парой тонких полосок, бит 1 – толстыми. Одна полоска – это только половина бита. Таким образом, Стрелочка гуляет по адресам 1278-1337 пока не проиграются все данные. Следом запишется сформировавшийся байт контрольной суммы. Далее произойдёт выход из программы. Вот и всё.

А теперь представьте, что вы записали мелодию данных в виде обычного звукового файла *.wav* или *.mp3*, а потом открыли в обычном редакторе музыки:

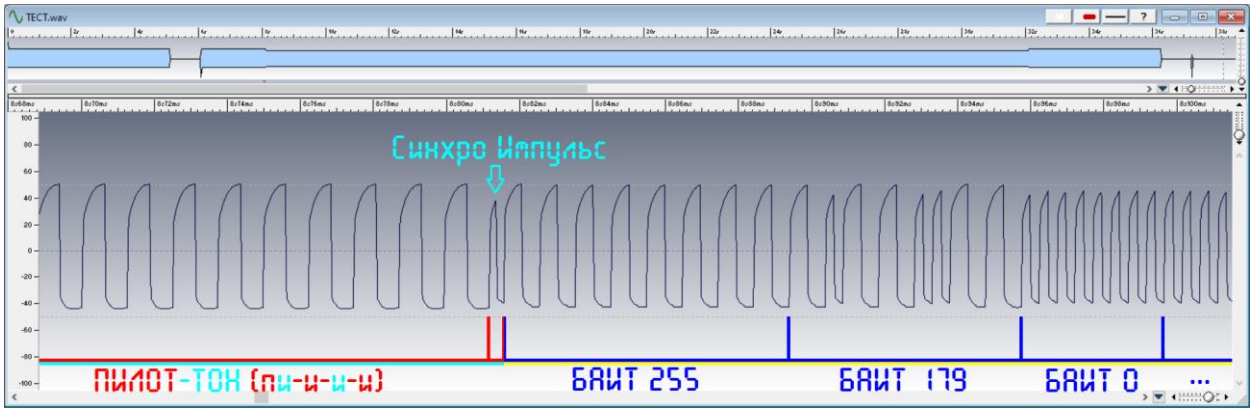


Рис. 470. Отображение загрузки в звуковом редакторе WaveLab. Искажение сигнала.

Вопреки предрассудкам, структура загрузки блока данных в виде звука выглядит еще проще. Одна палочка это и есть та самая полоска. Обратите внимание, какой искаженный сигнал с заострёнными пиками получился при конвертации. И это запись с компьютера. А только подумайте, какое качество звука было раньше на кассетах. Трудно представить, как программа считывания LOAD распознавала ту кашу с магнитной ленты. А ведь у каждого магнитофона еще и скорость плавала и головка по разному прилегала…

Как видно, короткую программу можно нарисовать вручную кусочками звуков требуемой частоты, причем с идеальными верхушками. В прошлом издании книги создание такого файла подробно описывалось. Так что тема полностью раскрыта.

А теперь предлагаю создать в памяти массивы чисел, имитирующие мелодию, а потом их проиграть методом загрузки. В книге 2013 года я поднимал эту тему, и проигрывал данные путем считывания файлов, которые перед этим записывал.

Изучив основы формирования сигнала, становится понятно, что для простой прослушки вовсе не обязательно так извращаться.

Вместе с мелодией нужно сделать плеер, который будет воспроизводить любой заданный массив из памяти, и проигрывать звуками, идентичными оригинальной загрузке. Можно придумать несколько вариантов плеера, в основе которого будет рассмотренные куски комплекса подпрограмм SAVE.

Первый вариант будет без пилот-тона и синхроимпульса, но с любым байтом идентификатора, который вы придумаете. В текущем случае, это не имеет значения. Пусть будет классический «255»:

Введите следующую программу:

```
Debugger
Dec
Go To 23613
23613 ← 65364

IX ← 29999
DE ← 1025
BC ← 15134
AF ← 65280
SP ← 65364
PC ← 29000

29000 ← 243 205 250 4 251 195 162 18
```

Не выходя из отладчика, добавьте мелодию:

```
Go To 30000
30000 ← 255 255 255 255 255 255 255 255
30128 ← 255 255 255 255 255 255 255 255
30256 ← 255 255 255 255 255 255 255 255
30320 ← 255 255 255 255 255 255 255 255
30384 ← 255 255 255 255 255 255 255 255
30512 ← 255 255 255 255 255 255 255 255
30576 ← 255 255 255 255 255 255 255 255
30640 ← 255 255 255 255 255 255 255 255
30704 ← 255 255 255 255 255 255 255 255
30832 ← 255 255 255 255 255 255 255 255
30896 ← 255 255 255 255 255 255 255 255
Trace
```

Смотрите и слушайте свое творение:

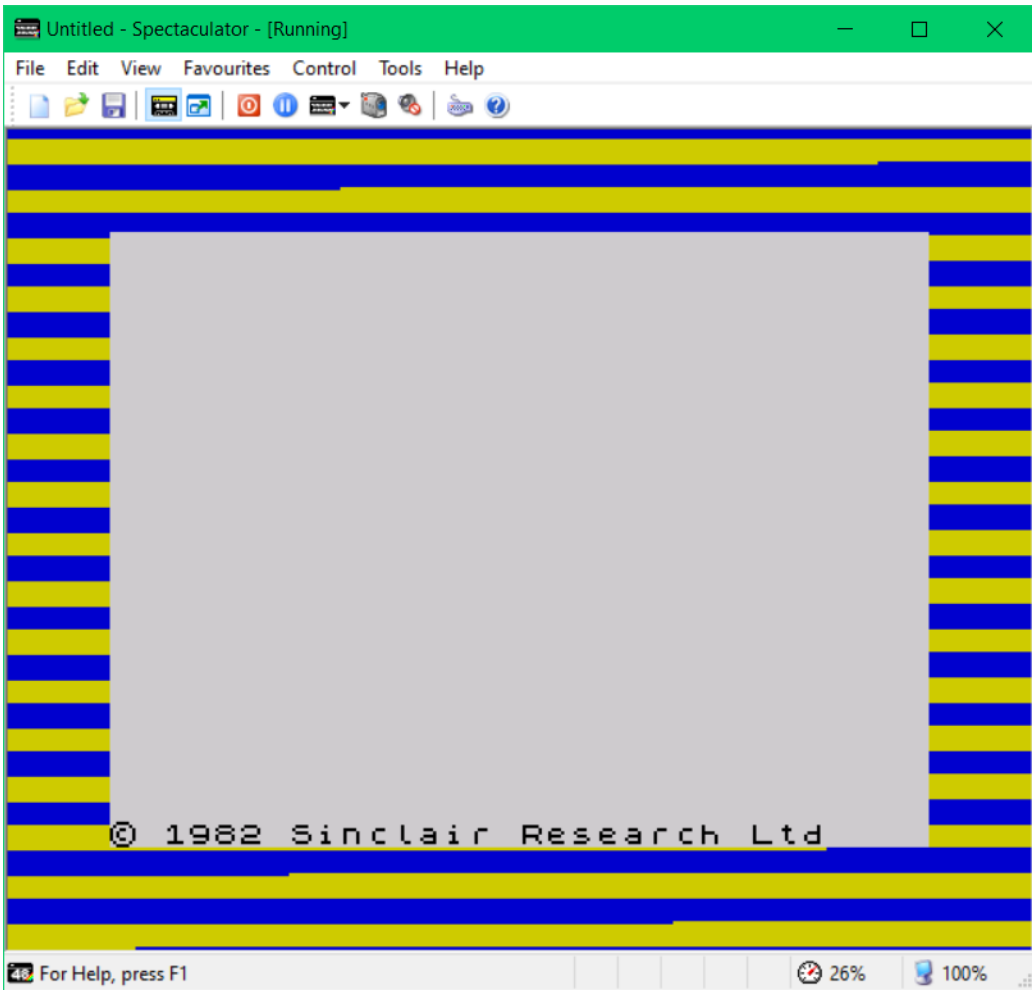


Рис. 471. Имитированная загрузка данных со звуком и выводом на экран.

По экрану побежали полосочки, и звуками загрузки проигралась короткая мелодия. Правда, круто? И вовсе не нужно для этого записывать холостой *.tzx* файл со всякой лабудой.

Для каких целей можно использовать такой плеер? Конечно для прослушки игр, которые вы хотите модифицировать. Разные типы данных имеют различное звучание. Прослушав игру можно предварительно понять, где и в каком количестве расположена графика.

Меняя значение «C» (26…31), можно задать цвет одной из полосочек.

Вариант этой программы на ассемблере *EmuZWin* можно использовать для многократного запуска.

```
      ORG 29000

DI                ; Запретить прерывания для предотвращения искажения мелодии
LD IX, 30000-1    ; Адрес откуда считать -1 для идентификатора
LD DE, 1024+1     ; Длина блока +1 байт для идентификатора
LD B, 59          ; Декоративная задержка для продолжения синхроимпульса
LD C, 30          ; Микс звука (+16), сигнала "Вкл" (+8) с цветом полоски (6)
LD A, 255         ; Байт идентификатора (0-заголовок 255-данные)
CALL 1274         ; Вызвать фрагмент подпрограммы последовательного
проигрывания битов
EI                ; Восстановить прерывания для работы клавиатуры
RET               ; Выход в BASIC
```

Проигрывание мелодии будет происходить по команде `RANDOMIZE USR 29000`.

Второй вариант плеера с проигрышем чистых данных без маркера, но с последним байтом контрольной суммы:

```
Debugger
Dec
Go To 23613
23613 ← 65364

IX ← 30000
DE ← 1024
BC ← 30
SP ← 65364
PC ← 29000

29000 ← 243 205 45 5 251 195 162 018

Go To 30000
30000 ← 255 255 255 255 255 255 255 255
30128 ← 255 255 255 255 255 255 255 255
30256 ← 255 255 255 255 255 255 255 255
30320 ← 255 255 255 255 255 255 255 255
30384 ← 255 255 255 255 255 255 255 255
30512 ← 255 255 255 255 255 255 255 255
30576 ← 255 255 255 255 255 255 255 255
30640 ← 255 255 255 255 255 255 255 255
30704 ← 255 255 255 255 255 255 255 255
30832 ← 255 255 255 255 255 255 255 255
30896 ← 255 255 255 255 255 255 255 255
```

Для теста я использовал мелодию из предыдущего примера.
Вариант для ассемблера EmuZWin:

```
ORG 29000

DI          ; Запретить прерывания для предотвращения искажения мелодии
LD IX, 30000      ; Адрес откуда считать данные
LD DE, 1024       ; Длина блока
LD C, 30    ; Микс звука (+16), сигнала "Вкл" (+8) с цветом полоски (6)
CALL 1325   ; Фрагмент подпрограммы последовательного проигрывания битов
EI          ; Восстановить прерывания для работы клавиатуры
RET         ; Выход в BASIC
```

Для коллекции звуков можно имитировать проигрывание пилот-тона. Обратиться к куску программы не получится, поэтому придётся вырезать и дорабатывать:
Для синтеза легендарного звука выполните следующую программу в God Mode:

```
Debugger
Dec
Go To 23613
23613 ← 65364

SP ← 65364
PC ← 40000

Go To 40000
40000 ← 33 160 15 243 62 2 71 16 254 211 254
40011 ← 238 31 6 164 45 32 245 5 37 242 71 156
40023 ← 251 195 162 18
```

Trace

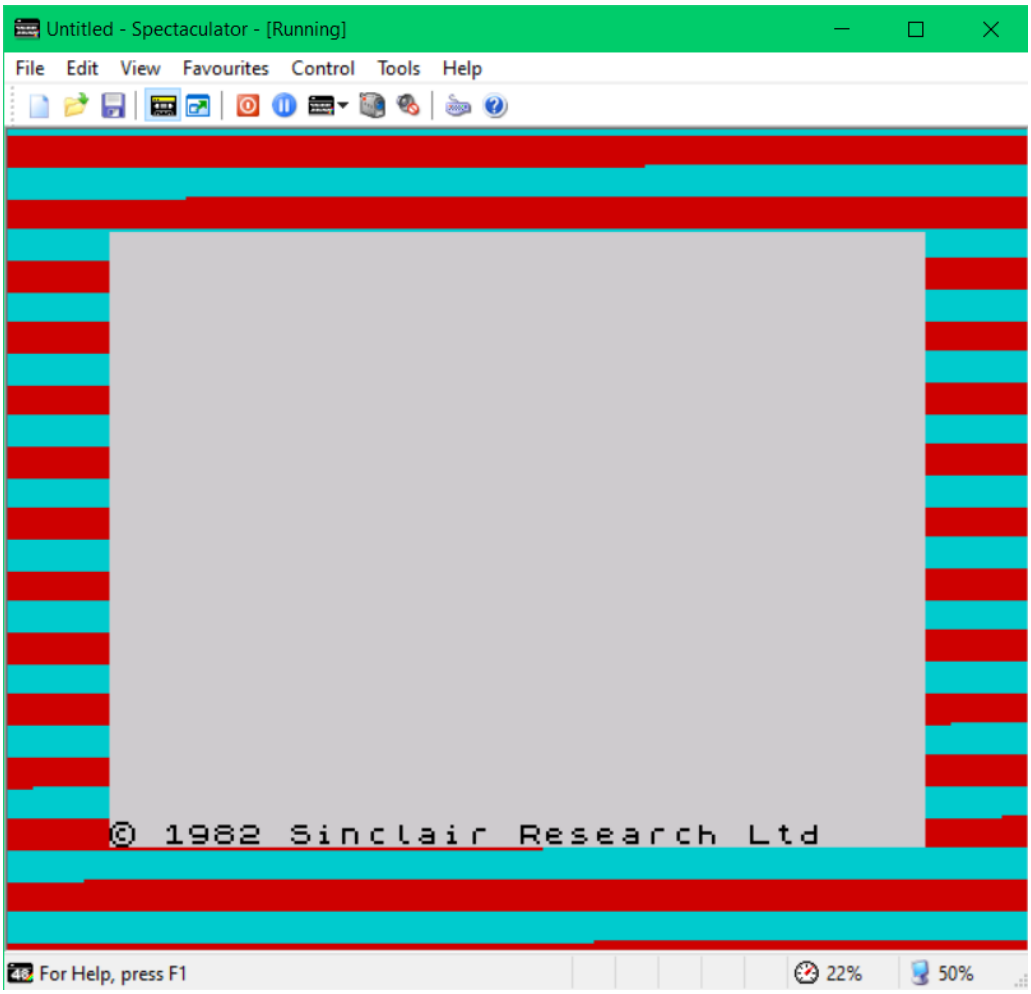


Рис. 472. Имитированный пилот-тон со звуком и выводом на экран.

…Тёмной ночью, после работы
Ты принимаешь сигнал.
Сердце заводишь на все обороты
И принимаешь сигнал…

На экране забегали полосочки и зазвучал оригинальный пилот-тон. Заменяя значение по адресу 40005 (от 0 до 7) можно изменить цвет первой полосы. Цвет другой полоски заменяется по адресу 40012 (25…31)

Вариант для ассемблера *EmuZWin*:

```
            ORG  40000

            LD HL, 4000       ; Продолжительность звука
            DI
            LD   A, 2         ; Микс звук и запись «Выкл» + цвет полоски 2
            LD   B, A
SIGNAL:     DJNZ SIGNAL
            OUT  (254), A
            XOR  31           ; Микс звук и запись «Вкл» + цвет полоски 5
            LD   B, 164       ; Размер одной полосочки и тон звучания
            DEC  L
            JR   NZ, SIGNAL
            DEC  B
            DEC  H
```

```
        JP   P, SIGNAL
        EI
        RET
```

На этом цветомузыкальную главу предлагаю закончить, ну а книга не кончается. Впереди будет еще достаточно интересных открытий.

Глава 63
Каналы и потоки в чемодане с инструментами

Краткое содержание: CHANS (23631/32) , CURCHL (23633/34), STRMS (23568) OPEN #, CLOSE #

А теперь новинка! Эта тема до осени 2024 года считалась непонятной и странной, поэтому обходилась стороной. Как оказалось, из-за неудачной концепции и аналогий. Подробности будут позже, а сейчас для ознакомления предлагаю начать с примера, который попрошу ввести нетипичным для этой книги методом.

Итак, натуральным способом с клавиатуры, введите следующую BASIC программу:

```
1 INPUT #2; INK 1;AT 11,4;"В СОРРЕНТО ";
2 LPRINT #2; INK 2;"НЕТ ";
3 PRINT INK 4;"ТОРРЕНТОВ"
```

Ввели?

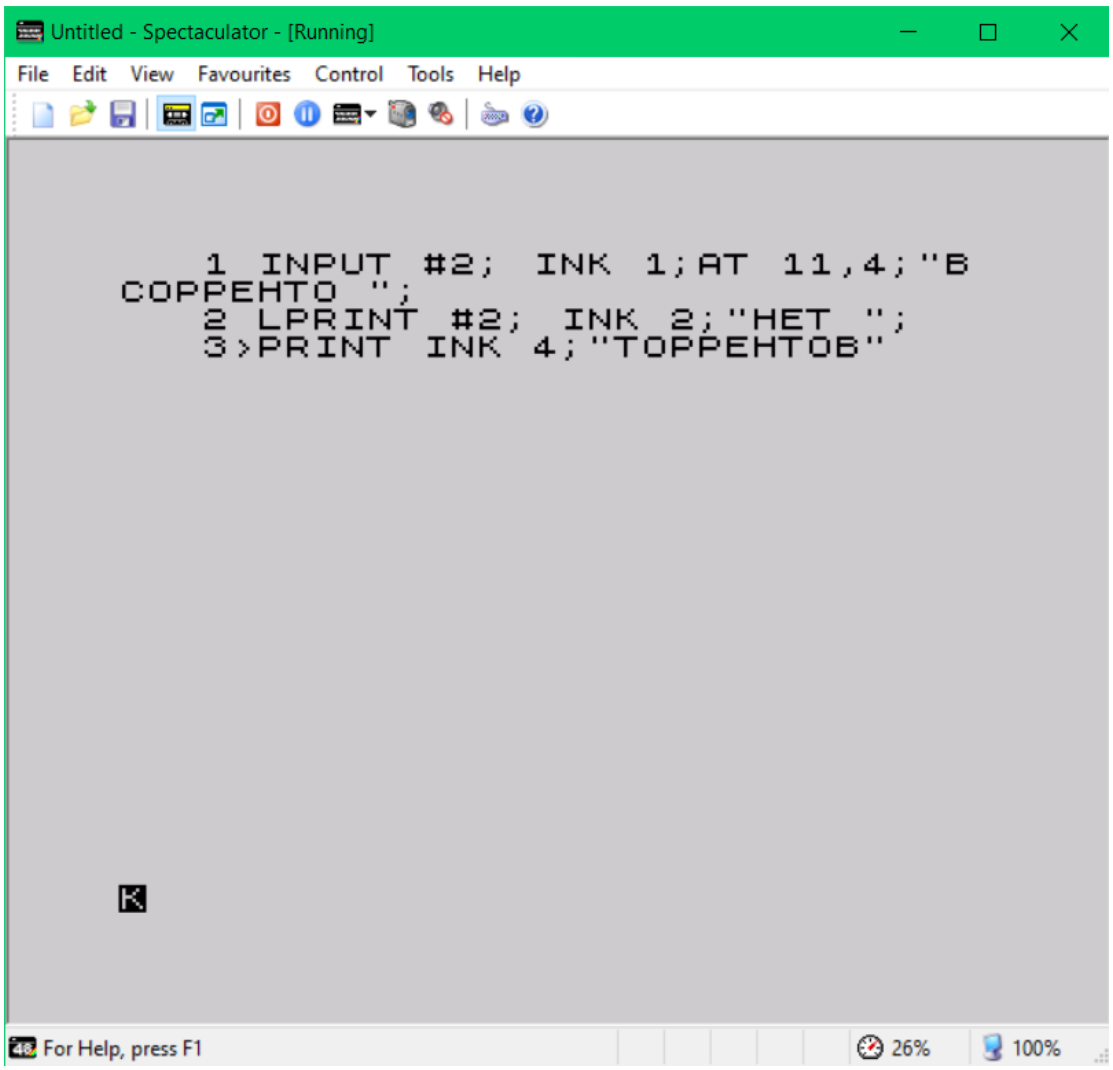


Рис. 473. Spectaculator. Программа на BASIC, введенная натуральным способом.

Спокойно, я не перегрелся под конец книги. А теперь еще и запустите этот шедевр традиционным способом, набрав команду RUN, а затем ENTER:

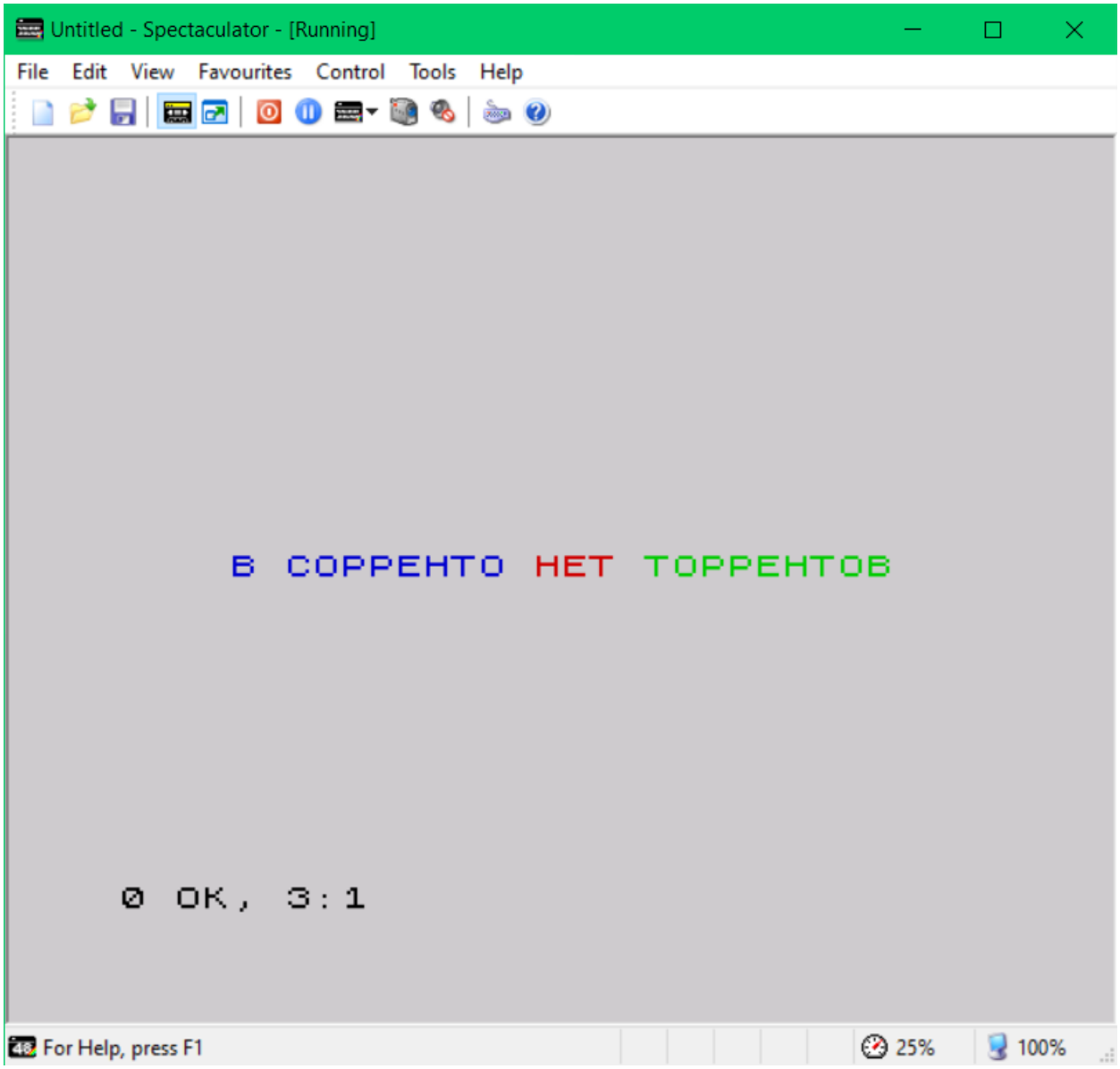


Рис. 474. Spectaculator. Вывод текста на экран всратой программой.

Поздний вечер в торрентах,
Нас закачкой не балует...

Потрясающе! Нет, то, что в некоторых европейских странах запрещены торренты и они могут блокироваться провайдерами, это понятно. Удивляет другое: по неведомым причинам, две первые команды превратились в банальный PRINT и вывели текст на экран.

Предлагаю разобраться и посмотреть, почему так прозошло. Далеко ходить не нужно, достаточно поставить малиновую точку в конец программы JUMP-C-R на адрес 7198, и посмотреть, куда оттуда разойдутся дорожки перед выполнением каждой строки:

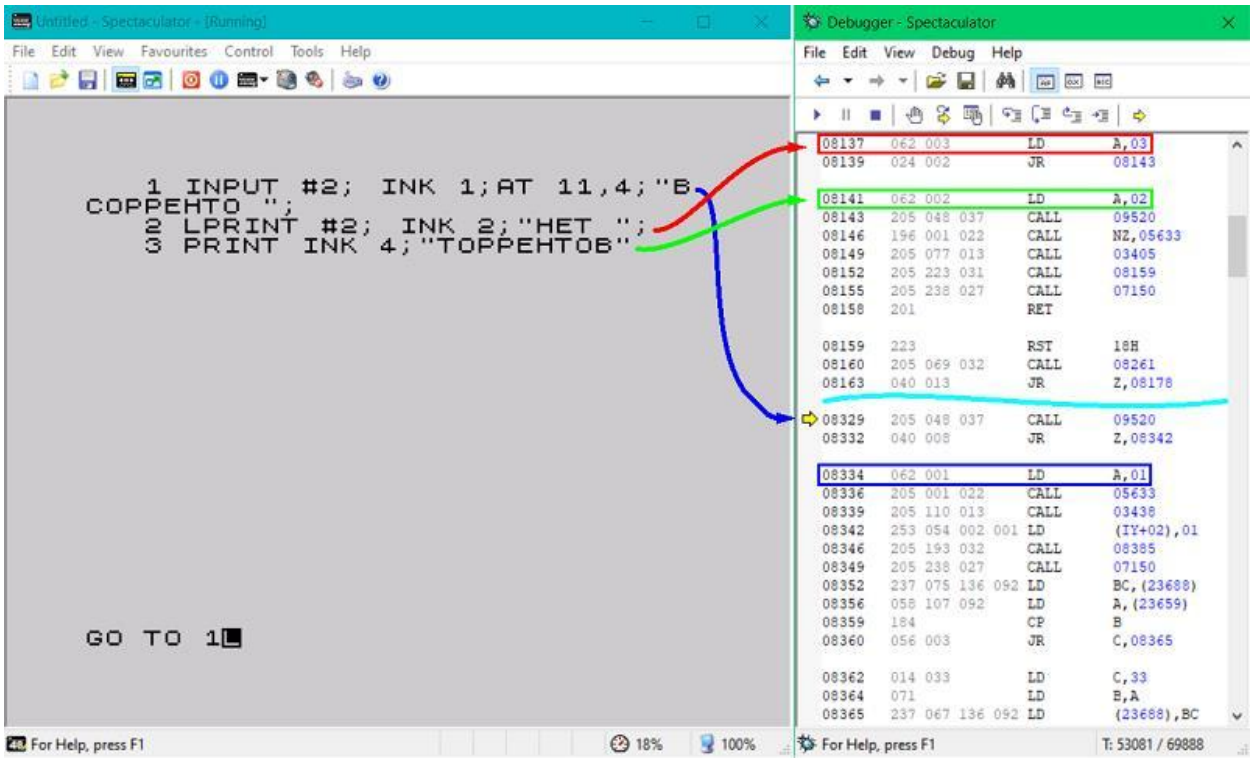


Рис. 475. Адреса программ вывода команд INPUT, LPRINT и PRINT.

Наглядно видно, что команды `PRINT` и `LPRINT` задаются одной и той же программой. Разница лишь в том, что перед выполнением `LPRINT` в «A» заносится 3. При выполнении `PRINT` – этим числом будет 2. А при старте программы, формирующей команду `INPUT`, задается число 1.

Правда есть какая-то логика в этих значениях? А теперь предлагаю вспомнить, в какие части экрана выводятся изображения по умолчанию. Например, `INPUT A`, в ожидании ввода, выведет мигающий курсор в нижние строки экрана. Результат действия команды `PRINT "A"` появится в левом верхнем углу экрана. Ну а команда `LPRINT "A"`, по задумке, должна напечатать букву на принтере.

Так вот, циферка с квадратиком `#2`, корректирует базовые установки, переназначая вывод изображения от работы команд на текущий сеанс выполнения. Таким образом, чистый `INPUT` тоже, что `INPUT #1`, команда `PRINT` это `PRINT #2`, а пустой `LPRINT` трактуется, как `LPRINT #3`.

Такими свойствами обладают все команды, которые выводят и вводят изображение на экран. Поэтому для команд `LIST` и `LLIST` эта фишка также работает.

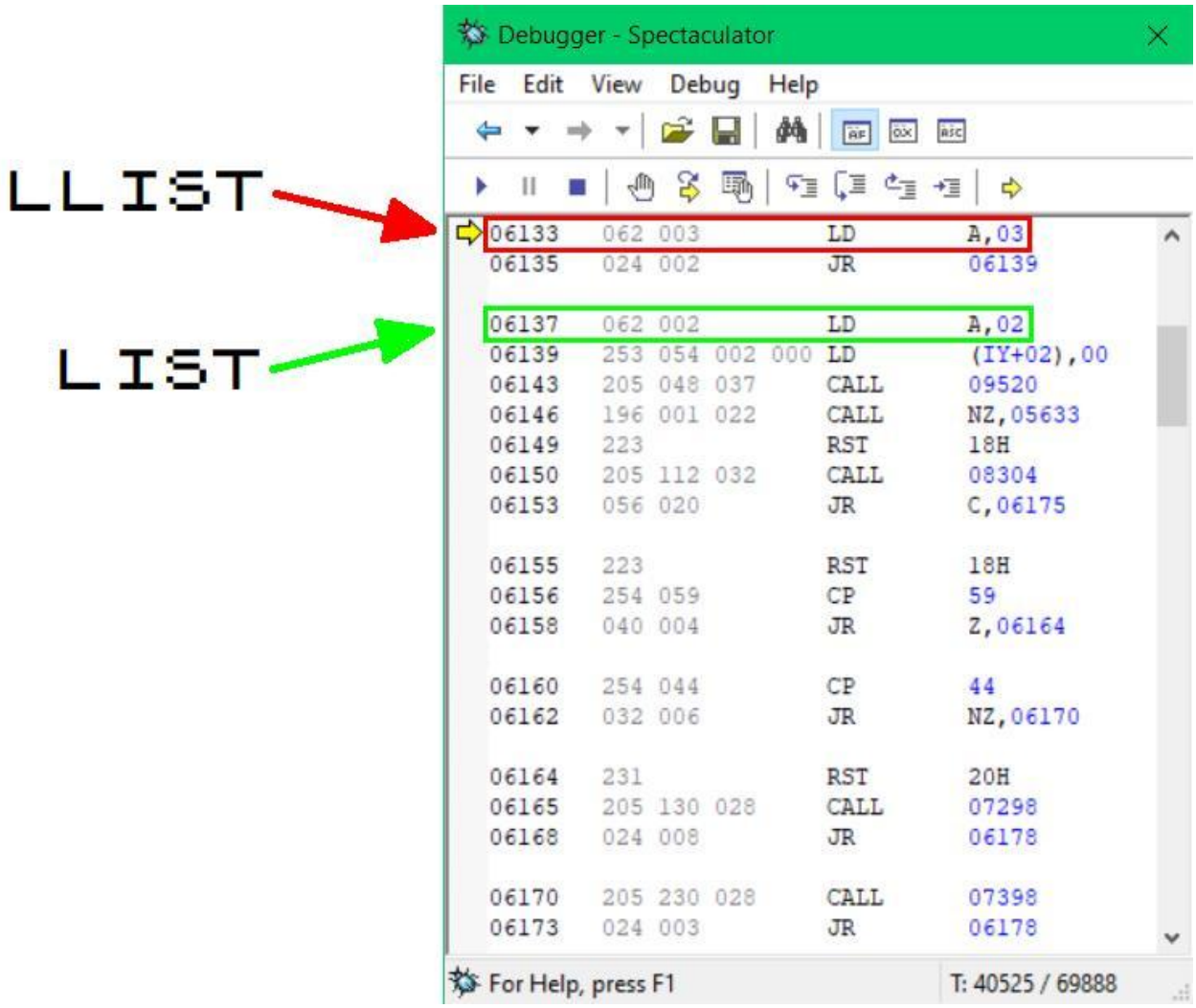


Рис. 476. Точки входа для команд LIST и LLIST.

Команда `LLIST #2` продублирует обычный `LIST`. Ну а если ввести `LIST #1`, то текст программы прокрутится в нижних строках экрана и исчезнет, затеревшись сообщением `0 OK` об успешном выполнении.

Значение `#0` будет аналогично `#1` и переназначит вывод графики в нижние строки. Например, `LIST #0` или `PRINT #0` тоже, что и `LIST #1` или `PRINT #1`.

И вроде бы все понятно, но есть некая недосказанность. В этом примере я использовал `INPUT` для вывода изображения без остановки для ввода. А что если таким образом вывести мигающий курсор, ожидающий ввода в верхний угол экрана?

Наберите и введите `INPUT #2;a`

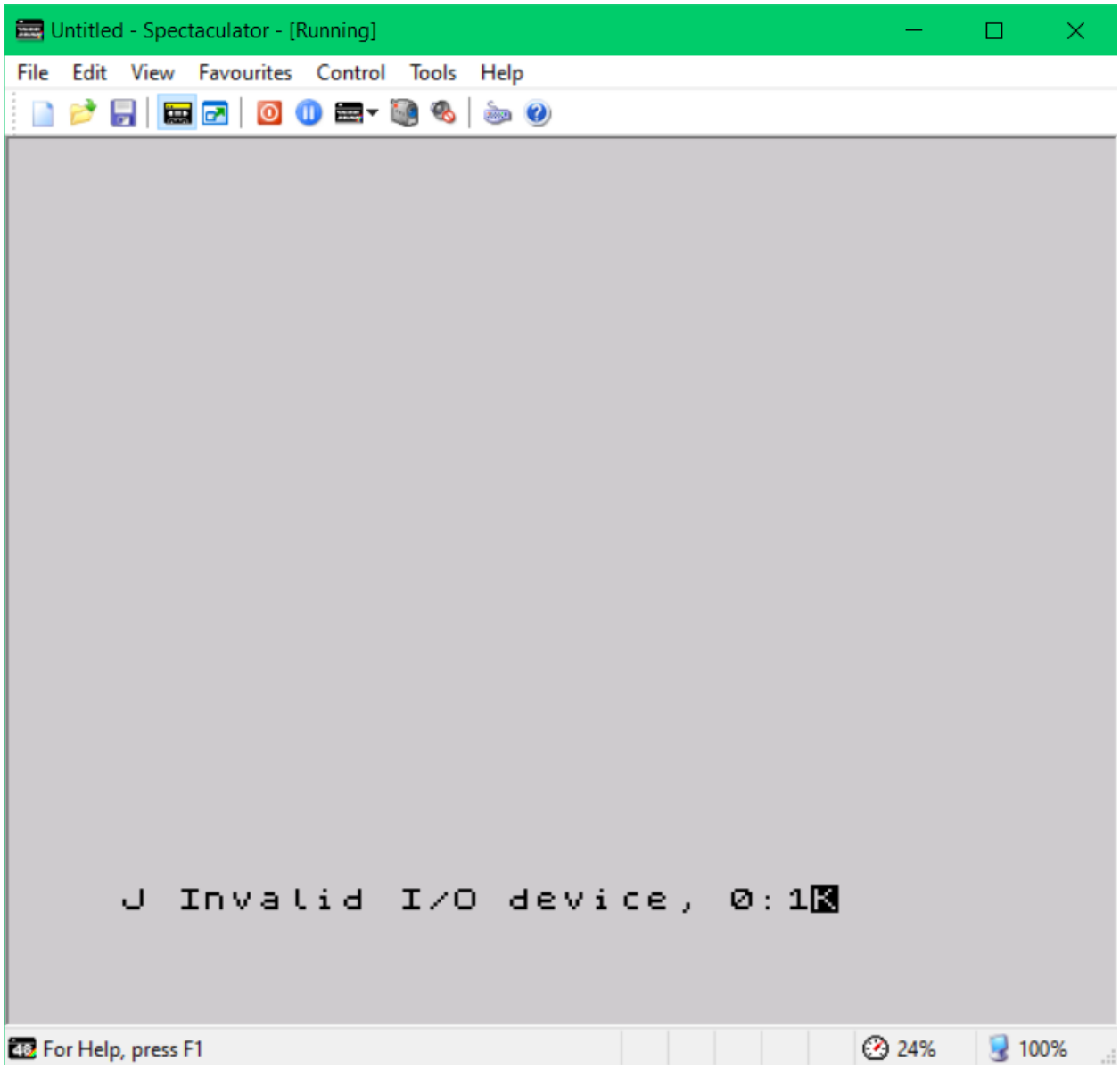


Рис. 477. Ошибка при попытке ввести INPUT #2;a.

Ага, хрен там! Сообщение об ошибке, да еще и с ошибкой в виде мигающего курсора позади сообщения. Про ошибку «Неверное устройство ввода-вывода» они не забыли, а вот про установку бита №5 (23612 ← 32) для подавления вывода на экран после сообщения забыли.

Компьютер издевательски показывает средний палец, как бы насмехаясь. Одно хорошо, следом за сообщением вы можете что-нибудь напечатать в отместку, не боясь его затереть:

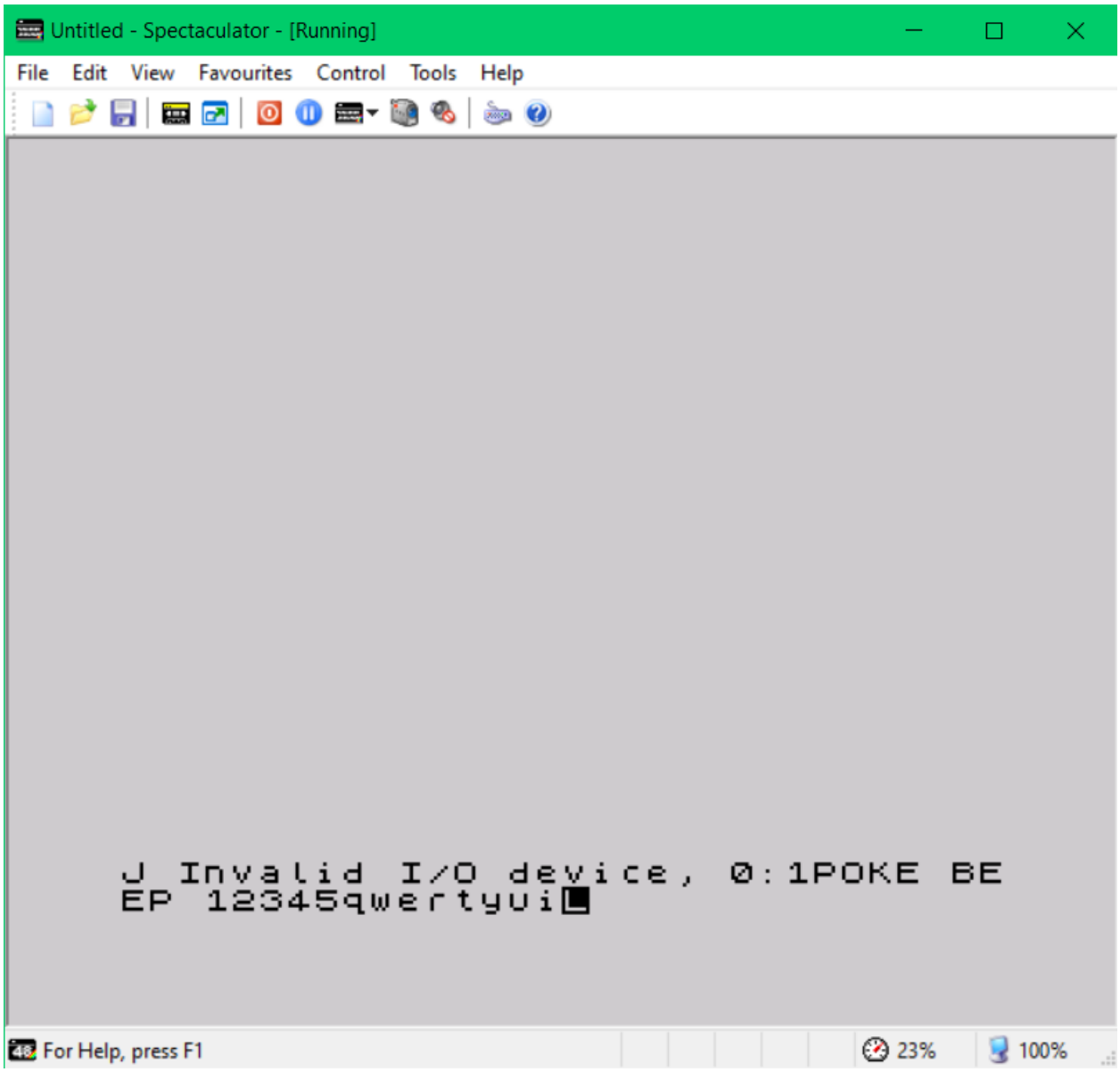


Рис. 478. Печать символов после сообщения об ошибке «J Invalid I/O Device».

Все это, конечно лирика и пройденный материал первого раздела книги, но общая картина как-то не радует.

Хорошо, а если ввести `PRINT #4`

Рис. 479. Ошибка при попытке ввести PRINT #4.

О как, еще лучше! «Неверный поток». Такого ругательства от Spectacucator'а я еще не получал. Ну что же, для коллекции не хватает естественным образом спровоцировать вывод «`8 End of file`».

Команда `INKEY$` также работает только со значениями `#0` и `#1`. Поскольку команда не рассчитана на автономное выполнение, то использовать её можно только в паре с `LET`, `PRINT` или `IF-THEN`. Голую команду натуральным способом ввести не получится.

Результаты поверхностных опытов предлагаю уместить в такую табличку:

	INPUT	PRINT	LPRINT	LIST	LLIST	INKEY$
#0	✅	✅	✅	✅	✅	✅
#1	✅	✅	✅	✅	✅	✅
#2	(без ввода)	✅	✅	✅	✅	❌
#3	(без ввода)	✅	✅	✅	✅	❌

Непонятные сообщения со словами «Ввод», «Вывод», «Поток», в совокупности со странным поведением команды `INPUT` плавно подвело к теме «Каналы и потоки».

Первый раз с термином «Канал» я познакомился в ноябре 1996 года. Тогда в стремительно удаляющуюся эпоху Спектрума, в книжном магазине на Литейном, я чудом успел урвать синюю книжку «Машинные коды». Несмотря на кучу ошибок, на тот момент, она стала глотком свежего воздуха. Плохо понимая ассемблер, и не зная структуру ПЗУ, меня удивляло начало любой программы вывода символов, которое начиналось с диковинного «Открыть канал S».

■ RST 0010

В системе SPECTRUM вся печать символов на экране выполняется при использованиии этой команды. С открытым каналом S подпрограмма PRINT-OUTPUT по адресу 09F4 действует как программа вывода.

Инструкция RST 0010 очень мощная и может использоваться для печати любого символа, изменения текущего положения печати использованием AT и TAB, печати расширенных ключевых слов и временных цветовых элементов. Следующая программа показывает это:

```
7D00  3E 02      LD A,+02          открыть канал S
7D02  CD 01 16   CALL CHAN-OPEN
7D05  08 18      LD B,+18
7D07  CD 44 0E   CALL CL-LINE      очистить экран
7D0A  3E 16      LD A,+AT
7D0C  D7         RST 0010          оператор
7D0D  3E 08      LD A,+05          PRINT AT 5,0
7D0F  D7         RST 0010
7D10  3E 00      LD A,+00
7D12  D7         RST 0010
7D13  3E 41      LD A,'A'
7D15  D7         RST 0010
```

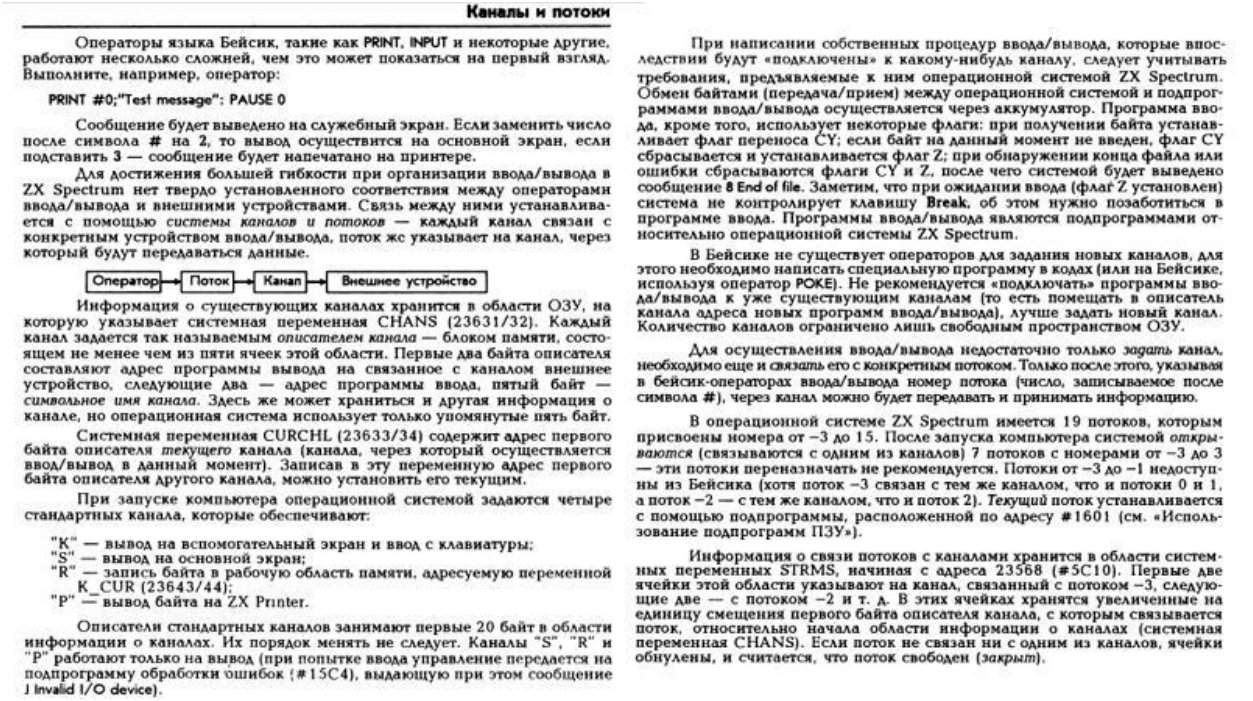
Рис. 480. Фрагмент программы из книги «Машинные коды» 1993 года.

Потом настало III тысячелетие. Эпоха Спектрума ушла в небытиё. Старая компьютерная литература резко обесценилась и стала не дороже обычной макулатуры.

Ну а у меня всё только ещё начиналось. В октябре 2000 года, я открыл для себя эмуляторы компьютера *ZX-Spectrum*, которые не только заработали на *Pentium'e*, но и издавали тот самый ностальгический звук загрузки, имитируя считывание с магнитофона. Чуть позже знакомый электронщик с работы отдал старые книжки, среди которых оказалась легендарная «*ZX-Spectrum для пользователей и программистов*» Николая

Родионова. И вот в этой книге снова попалась информация о каналах и потоках. На этот раз подробнее.

Заканчивались 2000-е годы. Возвращаясь с работы в переполненном Икарусе 142-го маршрута на Юго-Запад, я частенько пытался понять главу «*Каналы и потоки*».



Каналы и потоки

Операторы языка Бейсик, такие как PRINT, INPUT и некоторые другие, работают несколько сложней, чем это может показаться на первый взгляд. Выполните, например, оператор:

```
PRINT #0;"Test message": PAUSE 0
```

Сообщение будет выведено на служебный экран. Если заменить число после символа # на 2, то вывод осуществится на основной экран, если подставить 3 — сообщение будет напечатано на принтере.

Для достижения большей гибкости при организации ввода/вывода в ZX Spectrum нет твердо установленного соответствия между операторами ввода/вывода и внешними устройствами. Связь между ними устанавливается с помощью *системы каналов и потоков* — каждый канал связан с конкретным устройством ввода/вывода, поток же указывает на канал, через который будут передаваться данные.

Оператор → Поток → Канал → Внешнее устройство

Информация о существующих каналах хранится в области ОЗУ, на которую указывает системная переменная CHANS (23631/32). Каждый канал задается так называемым *описателем канала* — блоком памяти, состоящем не менее чем из пяти ячеек этой области. Первые два байта описателя составляют адрес программы вывода на связанное с каналом внешнее устройство, следующие два — адрес программы ввода, пятый байт — *символьное имя канала*. Здесь же может храниться и другая информация о канале, но операционная система использует только упомянутые пять байт.

Системная переменная CURCHL (23633/34) содержит адрес первого байта описателя *текущего* канала (канала, через который осуществляется ввод/вывод в данный момент). Записав в эту переменную адрес первого байта описателя другого канала, можно установить его текущим.

При запуске компьютера операционной системой задаются четыре стандартных канала, которые обеспечивают:

"K" — вывод на вспомогательный экран и ввод с клавиатуры;
"S" — вывод на основной экран;
"R" — запись байта в рабочую область памяти, адресуемую переменной K_CUR (23643/44);
"P" — вывод байта на ZX Printer.

Описатели стандартных каналов занимают первые 20 байт в области информации о каналах. Их порядок менять не следует. Каналы "S", "R" и "P" работают только на вывод (при попытке ввода управление передается на подпрограмму обработки ошибок (#15C4), выдающую при этом сообщение 1 Invalid I/O device).

При написании собственных процедур ввода/вывода, которые впоследствии будут «подключены» к какому-нибудь каналу, следует учитывать требования, предъявляемые к ним операционной системой ZX Spectrum. Обмен байтами (передача/прием) между операционной системой и подпрограммами ввода/вывода осуществляется через аккумулятор. Программа ввода, кроме того, использует некоторые флаги: при получении байта устанавливает флаг переноса CY; если байт на данный момент не введен, флаг CY сбрасывается и устанавливается флаг Z; при обнаружении конца файла или ошибки сбрасываются флаги CY и Z, после чего системой будет выведено сообщение 8 End of file. Заметим, что при ожидании ввода (флаг Z установлен) система не контролирует клавишу **Break**, об этом нужно позаботиться в программе ввода. Программы ввода/вывода являются подпрограммами относительно операционной системы ZX Spectrum.

В Бейсике не существует операторов для задания новых каналов, для этого необходимо написать специальную программу в кодах (или на Бейсике, используя оператор POKE). Не рекомендуется «подключать» программы ввода/вывода к уже существующим каналам (то есть помещать в описатель канала адреса новых программ ввода/вывода), лучше задать новый канал. Количество каналов ограничено лишь свободным пространством ОЗУ.

Для осуществления ввода/вывода недостаточно только *задать* канал, необходимо еще и *связать* его с конкретным потоком. Только после этого, указывая в бейсик-операторах ввода/вывода номер потока (число, записываемое после символа #), через канал можно будет передавать и принимать информацию.

В операционной системе ZX Spectrum имеется 19 потоков, которым присвоены номера от −3 до 15. После запуска компьютера системой *открываются* (связываются с одним из каналов) 7 потоков с номерами от −3 до 3 — эти потоки переназначать не рекомендуется. Потоки от −3 до −1 недоступны из Бейсика (хотя поток −3 связан с тем же каналом, что и потоки 0 и 1, а поток −2 — с тем же каналом, что и поток 2). *Текущий* поток устанавливается с помощью подпрограммы, расположенной по адресу #1601 (см. «Использование подпрограмм ПЗУ»).

Информация о связи потоков с каналами хранится в области системных переменных STRMS, начиная с адреса 23568 (#5C10). Первые две ячейки этой области указывают на канал, связанный с потоком −3, следующие две — с потоком −2 и т. д. В этих ячейках хранятся увеличенные на единицу смещения первого байта описателя канала, с которым связывается поток, относительно начала области информации о каналах (системная переменная CHANS). Если поток не связан ни с одним из каналов, ячейки обнулены, и считается, что поток свободен (*закрыт*).

Рис. 481. Глава «Каналы и потоки» из книги «ZX-Spectrum для пользователей и программистов».

«Внешние устройства», «Каналы», «Потоки», «Передача данных», «Подключение», «Связывание каналов».... Да-а-а-а. Вроде бы написано неплохо, но всё равно нихрена непонятно. От главы веет какой-то высокоуровневой объектно-ориентированной тоской 2000-х, вперемешку с типовым сисадминством, но никак не старым добрым Spectrum'ом 48K.

За последние 20 лет я много раз открывал книгу на этой странице, читал пару абзацев, и закрывал, так и ничего не поняв. Никак не получалось преодолеть невидимый психологический барьер. Да и надобности вникать не было… до какого-то момента. Этим моментом стало написание данной книги. Это при изучении теоретического программирования на ассемблере можно было не вникать во многое, но когда дело дошло до расковыривания ПЗУ, стало понятно, что халява закончилась. Все программы тесно связаны между собой. Незнание одного, казалось бы, неинтересного фрагмента программы, влекло за собой полное непонимание нужного мне куска.

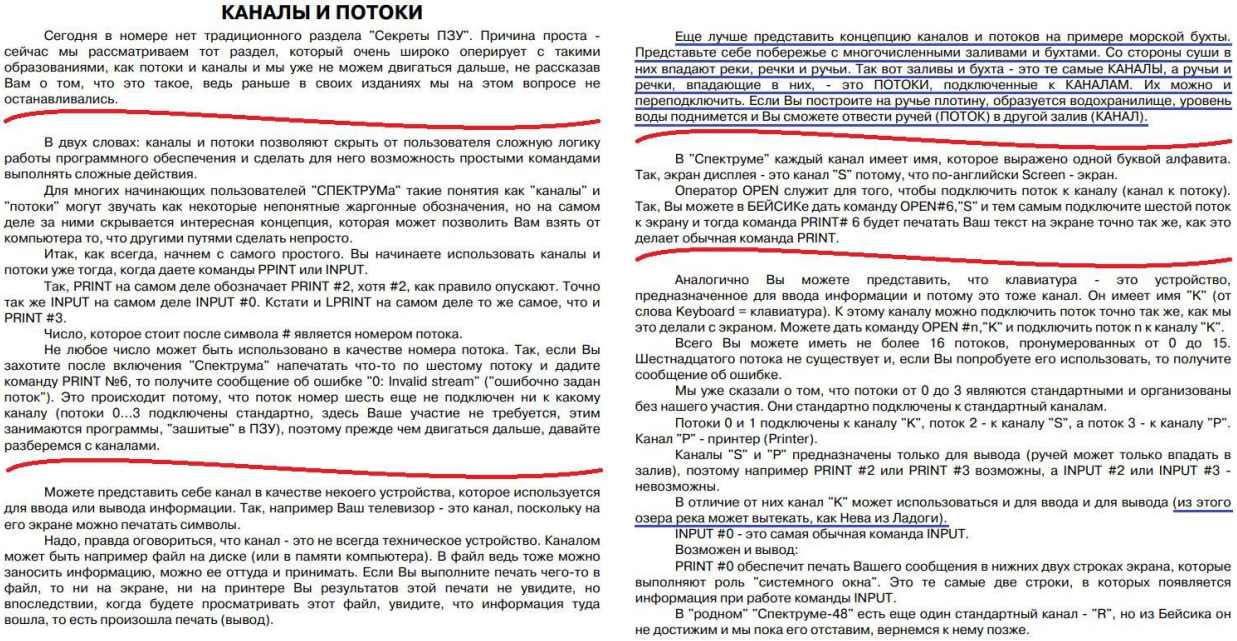
Достаточно серьёзно столкнулся с этой темой в этой книге, когда писал главу «Возвращение Жёлтой Стрелочки в город BASIC» летом 2024 года. Продвигаясь к BASIC системе, Стрелочка формировала две загадочных таблицы в области системных переменных.

Напомню, первая таблица была переписана из адреса 5551. Она заполнила 21 байт области системных переменных с адреса 23734 по 23754 включительно и вплотную подпёрла область `PROG (23635)`. На начало этого массива указывает переменная `CHANS (23631)`. Чуть позже сформировался второй массив, размером 38 байт и расположился с адреса 23568 по 23605. Он назывался `STRMS`. Как они формируются, можно почитать в той самой главе в самом начале этой книги.

Тогда по поводу их предназначения я деликатно промолчал, как и о некоторых других моментах, которые осознал к концу книги в результате опытов с памятью и самим файлом *spectaculator.exe*.

Чуть позже я попытался поверхностно вникнуть в концепцию «Каналов с потоками», но снова потерпел неудачу. И вот однажды к концу 2024 года, во время поездки в Австрию у меня дошли руки до коллекции журнала «ZX-Ревю» за 1991-1997 годы, комплекты которого я скачал из интернета и скопировал на телефон.

Сидя в поезде, я пролистывал содержание журналов. И вот в номере за ноябрь-декабрь 1991 года мне снова попались каналы с потоками. Я воодушевился, открыл статью, пролистал 10 страниц и снова закрыл.



КАНАЛЫ И ПОТОКИ

Сегодня в номере нет традиционного раздела "Секреты ПЗУ". Причина проста - сейчас мы рассматриваем тот раздел, который очень широко оперирует с такими образованиями, как потоки и каналы и мы уже не можем двигаться дальше, не рассказав Вам о том, что это такое, ведь раньше в своих изданиях мы на этом вопросе не останавливались.

В двух словах: каналы и потоки позволяют скрыть от пользователя сложную логику работы программного обеспечения и сделать для него возможность простыми командами выполнять сложные действия.

Для многих начинающих пользователей "СПЕКТРУМа" такие понятия как "каналы" и "потоки" могут звучать как некоторые непонятные жаргонные обозначения, но на самом деле за ними скрывается интересная концепция, которая может позволить Вам взять от компьютера то, что другими путями сделать непросто.

Итак, как всегда, начнем с самого простого. Вы начинаете использовать каналы и потоки уже тогда, когда даете команды PPINT или INPUT.

Так, PRINT на самом деле обозначает PRINT #2, хотя #2, как правило опускают. Точно так же INPUT на самом деле INPUT #0. Кстати и LPRINT на самом деле то же самое, что и PRINT #3.

Число, которое стоит после символа # является номером потока.

Не любое число может быть использовано в качестве номера потока. Так, если Вы захотите после включения "Спектрума" напечатать что-то по шестому потоку и дадите команду PRINT №6, то получите сообщение об ошибке "0: Invalid stream" ("ошибочно задан поток"). Это происходит потому, что поток номер шесть еще не подключен ни к какому каналу (потоки 0...3 подключены стандартно, здесь Ваше участие не требуется, этим занимаются программы, "зашитые" в ПЗУ), поэтому прежде чем двигаться дальше, давайте разберемся с каналами.

Можете представить себе канал в качестве некоего устройства, которое используется для ввода или вывода информации. Так, например Ваш телевизор - это канал, поскольку на его экране можно печатать символы.

Надо, правда оговориться, что канал - это не всегда техническое устройство. Каналом может быть например файл на диске (или в памяти компьютера). В файл ведь тоже можно заносить информацию, можно ее оттуда и принимать. Если Вы выполните печать чего-то в файл, то ни на экране, ни на принтере Вы результатов этой печати не увидите, но впоследствии, когда будете просматривать этот файл, увидите, что информация туда вошла, то есть произошла печать (вывод).

Еще лучше представить концепцию каналов и потоков на примере морской бухты. Представьте себе побережье с многочисленными заливами и бухтами. Со стороны суши в них впадают реки, речки и ручьи. Так вот заливы и бухта - это те самые КАНАЛЫ, а ручьи и речки, впадающие в них, - это ПОТОКИ, подключенные к КАНАЛАМ. Их можно и переподключить. Если Вы построите на ручье плотину, образуется водохранилище, уровень воды поднимется и Вы сможете отвести ручей (ПОТОК) в другой залив (КАНАЛ).

В "Спектруме" каждый канал имеет имя, которое выражено одной буквой алфавита. Так, экран дисплея - это канал "S" потому, что по-английски Screen - экран.

Оператор OPEN служит для того, чтобы подключить поток к каналу (канал к потоку). Так, Вы можете в БЕЙСИКе дать команду OPEN#6,"S" и тем самым подключите шестой поток к экрану и тогда команда PRINT# 6 будет печатать Ваш текст на экране точно так же, как это делает обычная команда PRINT.

Аналогично Вы можете представить, что клавиатура - это устройство, предназначенное для ввода информации и потому это тоже канал. Он имеет имя "K" (от слова Keyboard = клавиатура). К этому каналу можно подключить поток точно так же, как мы это делали с экраном. Можете дать команду OPEN #n,"K" и подключить поток n к каналу "K".

Всего Вы можете иметь не более 16 потоков, пронумерованных от 0 до 15. Шестнадцатого потока не существует и, если Вы попробуете его использовать, то получите сообщение об ошибке.

Мы уже сказали о том, что потоки от 0 до 3 являются стандартными и организованы без нашего участия. Они стандартно подключены к стандартным каналам.

Потоки 0 и 1 подключены к каналу "K", поток 2 - к каналу "S", а поток 3 - к каналу "P". Канал "P" - принтер (Printer).

Каналы "S" и "P" предназначены только для вывода (ручей может только впадать в залив), поэтому например PRINT #2 или PRINT #3 возможны, а INPUT #2 или INPUT #3 - невозможны.

В отличие от них канал "K" может использоваться и для ввода и для вывода (из этого озера река может вытекать, как Нева из Ладоги).

INPUT #0 - это самая обычная команда INPUT.

Возможен и вывод:

PRINT #0 обеспечит печать Вашего сообщения в нижних двух строках экрана, которые выполняют роль "системного окна". Это те самые две строки, в которых появляется информация при работе команды INPUT.

В "родном" "Спектруме-48" есть еще один стандартный канал - "R", но из Бейсика он не достижим и мы пока его отставим, вернемся к нему позже.

Рис. 482. Фрагменты статьи «Каналы и потоки» из журнала ZX-Ревю №№11-12 за 1991 год.

Нет, начало на двух первых страницах еще понять как-то можно. С удивлением обнаружил, что стиль повествования у авторов журнала, чем-то похож на мой. Приятно тронули сравнения «*…из этого озера река может вытекать как Нева из Ладоги*».

Из прочтенного я вынес только душевные сравнения, которые подчеркнул синим цветом. По существу вопроса это такая же непонятная концепция высокоуровневых терминологий, как и в справочнике «*ZX-Spectrum для пользователей и программистов*».

«*…чтобы скрыть от пользователя сложную логику программного обеспечения…*» - пожалуй да, всё как всегда. Суть в том, чтобы все засекретить, скрыть и напустить тумана важности… И вообще лучше не выходить из комнаты, не совершать ошибку.

А вот дальше пошел тихий ужас. Статья больше похожа на научную работу для защиты докторской степени, которую без литровки колы не поймешь. И ладно бы только это. Большая часть посвящалась «Спектруму 128», который мне не симпатизировал еще с детства.

Посмотрев заумную и пространную статью в журнале, я понял, что объяснение в книге «*ZX-Spectrum для пользователей и программистов*» куда адекватнее.

Продолжая изучать журнал «ZX-Ревю», я добрался до середины 1992 года и понял, что думаю так не один:

ВОЗВРАЩАЯСЬ К НАПЕЧАТАННОМУ

КАНАЛЫ И ПОТОКИ

В 12-м номере "ZX-РЕВЮ" мы подняли вопрос о концепции каналов и потоков. Среди наших читателей есть мнение о том, что при всей нужности и важности этого вопроса, сама статья грешила определенной академичностью и, если читать ее от начала и до конца, то не все могли с ней разобраться. Большинство взяло то, что лежало на поверхности, а остальное отложили для разбора на долгое время.

В связи с этим мы с радостью принимаем предложение, высказанное нашим постоянным корреспондентом из г. Балашова Саратовской области Пашориным В. Г. о введении постоянной рубрики "Возвращаясь к напечатанному", и начинаем с того, что дадим рекомендации, которые предлагает наш уважаемый автор для тех, кто хотел бы не только в теории ознакомиться поближе с каналами и потоками, но и применить свои знания на практике.

Это достойное развитие начатой темы, и нам будет очень приятно, если полезное начинание будет подхвачено и новая рубрика вызовет появление новых публикаций.

* * *

Рис. 483. Фрагмент из журнала ZX-Ревю №№5-6 за 1992 год.

«*Грешила определённой академичностью*»? Да ладно? Неужели даже авторы впоследствии признали, что в журнале за предыдущий год была академическая статья, насквозь пропитанная «культом» 128К и неприменимая к базовой модели 48К.

Наконец-то нашелся адекватный читатель, который объяснил вкратце и нормальным языком. На самом деле, лучше короткий, но толковый пост на пару страниц, чем заумные профессорские исследования на двадцать.

После короткой журнальной статьи я уловил суть. Продолжая копать дальше, я и понял, что эта тема должна найти свой укромный уголок в этой книжке. С этого момента решил детально изучить структуру этих таблиц и что с ними можно сотворить. Ну а сделать из этого главу и окончательно разобраться дошли руки только летом 2025 года.

Проанализировав ключевые подпрограммы, которые имеют отношение к теме «Каналы и потоки», я понял, что книжная концепция крайне неудачная. Она не совсем корректная, не отражает структуры работы подпрограмм, по крайней мере, в 48К режиме, и поэтому, долгое время вводила в заблуждение.

Предлагаю на время забыть все то, что написано в старых журналах, открыть отладчик и Стрелочкой походить по программам. Для начала, предлагаю вспомнить, из чего сформировываются блоки.

Итак, во время послесбросовой загрузки, из адреса 5551 в 23734 копируется таблица из 21 байта, которая состоит из 5-ти блоков по 5 байт каждый. Блоки стоят друг на друге. Самой конечной, в роли фундамента, выступает заглушка «128», которая лежит на области для BASIC программы. На рисунке она отделена зелёной чертой.

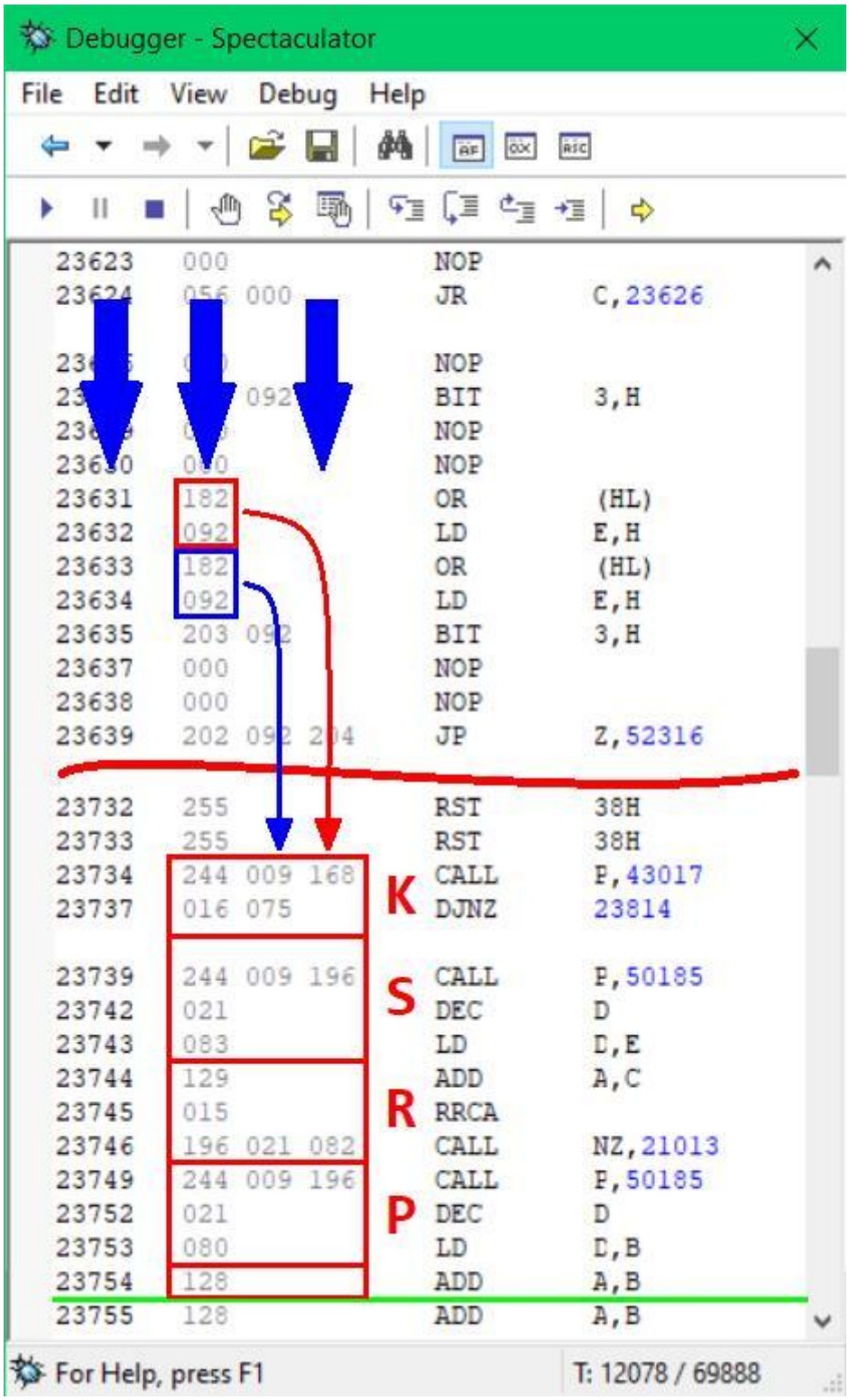


Рис. 484. Структура таблицы CHANS и связей с ней.

Это первая таблица, которая задаёт те самые дополнительные «гибкие» свойства командам `INPUT`, `PRINT`, `LPRINT`, `LIST` , `LLIST` и `INKEY$`.

Из области ПЗУ как такового прямого обращения к этим адресам не замечено. Управление массивом осуществляется через ячейки-посредники `CHANS (23631/32)` и `CURCHL (23633/34)` (*жарг. Чурчхелла*). Вот к ним напрямую по этим адресам ссылается несколько программ. Переменная `CHANS (23631/32)` является указателем на начало массива. Единожды создавшись во время загрузки `4664 LD (23631),HL` более нигде не модифицируется, а только считывается.

Комплект ячеек CURCHL (23633/34) является указателем, какой из 4-х блоков используется в данный момент. Он указывает на заглавный байт одного из блоков. Таким образом, с течением времени он может принимать одно из 4-х значений: 23734, 23739, 23744 или 23749. Модификация этого указателя прямым текстом производится в единственном месте, 5653 LD (23633),HL в подпрограмме CHAN-FLAG.

Таким образом, в данный массив при желании можно добавлять свои блоки, отодвинув вниз BASIC области. Как переместить BASIC методом God Mode рассказывалось в главах о номерных строках и будет напомнено ниже, в примере.

Ну а структура отдельно взятого блока следующая:

- 2 байта – адрес перехода для выполнения 1-й программы (вывод)
- 2 байта – адрес перехода для выполнения 2-й программы (ввод)
- 1 байт – имя блока, состоящее из буквенного символа

Таким образом, весь массив будет выглядеть так:

Адрес	Значения	Примечание	Индекс для команд BASIC
Блок K – Keyboard – Нижние строки экрана			
23734	244, 9	Подпрограмма PRINT-OUT (2548)	#0, #1
23736	168, 16	Подпрограмма KEYBOARD INPUT (4264)	#0, #1
23738	75	Код буквы K	
Блок S – Screen – Основная часть экрана			
23739	244, 9	Подпрограмма PRINT-OUT (2548)	#2
23741	196, 21	Подпрограмма-заглушка REPORT-J (5572). Вывод Invalid I/O Device	#2
23743	83	Код буквы S	
Блок R – Взятие строки на редактирование и работа команды STR$			
23744	129, 15	Подпрограмма ADDCHAR (3969)	
23746	196, 21	Подпрограмма-заглушка REPORT-J (5572). Вывод Invalid I/O Device	
23748	82	Код буквы R	
Блок P – Printer – Как бы вывод на мифический принтер			
23749	244, 9	Подпрограмма PRINT-OUT (2548)	#3
23751	196, 21	Подпрограмма-заглушка REPORT-J (5572). Вывод Invalid I/O Device	#3
23753	80	Код буквы P	
23754	128	Маркер-заглушка конца блока	

Теперь понятно, почему при попытке ввести INPUT #2 с функцией набора с клавиатуры (точка с запятой + буква), выдается ошибка? Да потому что по адресу 23741 поставлен переход на программу печати ошибки и не более того. Для задания работоспособности нужно вместо заглушки поставить адрес для перехода.

Манипулируя этими данными можно создать много интересных эффектов. Любопытно ещё и то, что программы получатся достаточно простые. Первым делом предлагаю восстановить справедливость для INPUT #2. Для этого введите такую короткую программу:

```
Debugger
Dec
Go To 23741
23741 ← 4264
```

`Trace`

А теперь натуральным способом введите BASIC строку `INPUT #2;a`

Рис. 485. Ввод строки с INPUT #2.

И вместо сообщения «`J Invalid I/O device`», в верхнем углу экрана теперь замигал курсор:

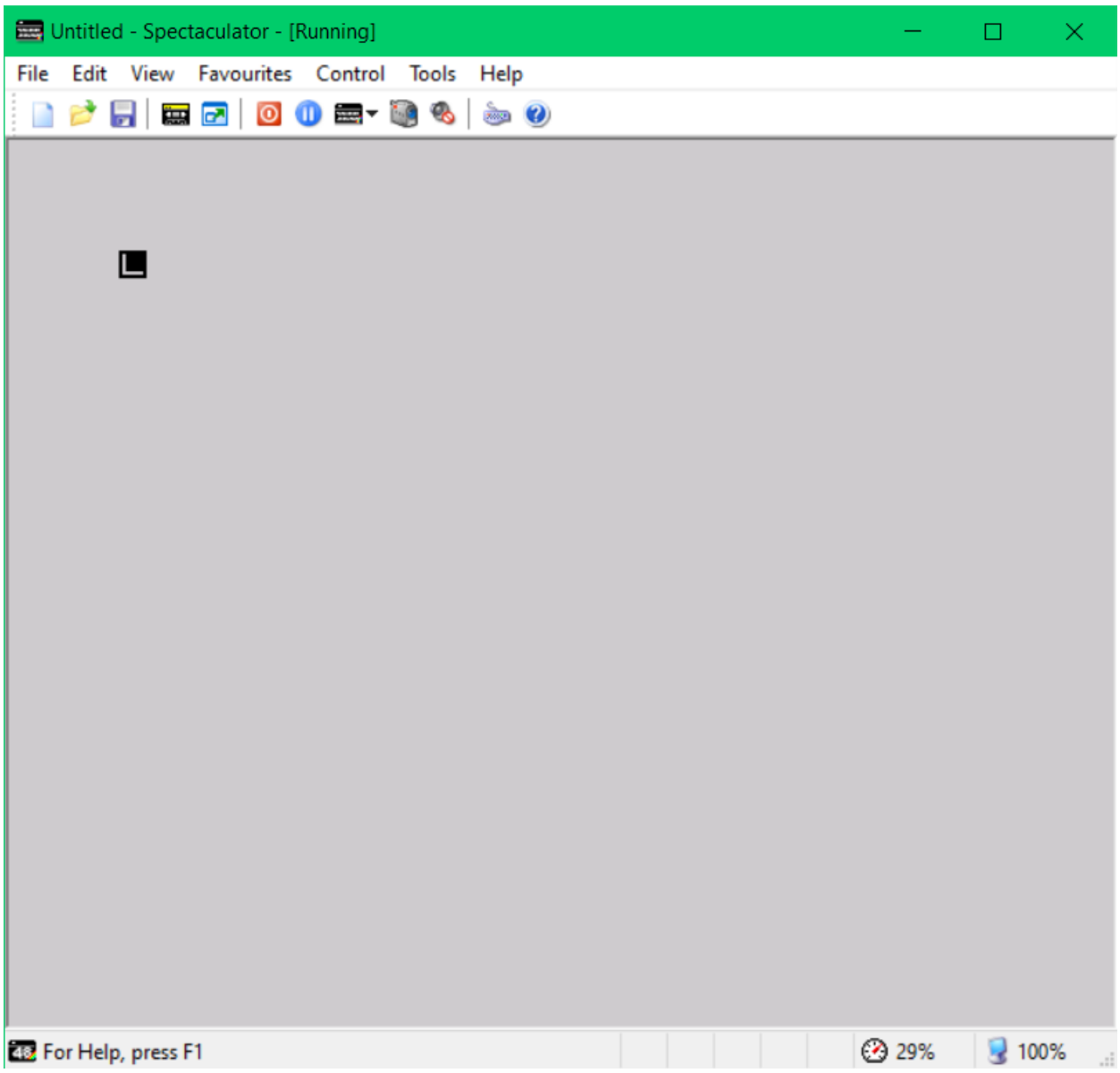


Рис. 486. Курсор, ожидающий ввода в верхнем углу экрана.

Ага, в этих ваших литературах пишут, что нельзя вывести в верхнюю область экрана? Всё можно. А теперь попробуйте что-нибудь набрать и ввести:

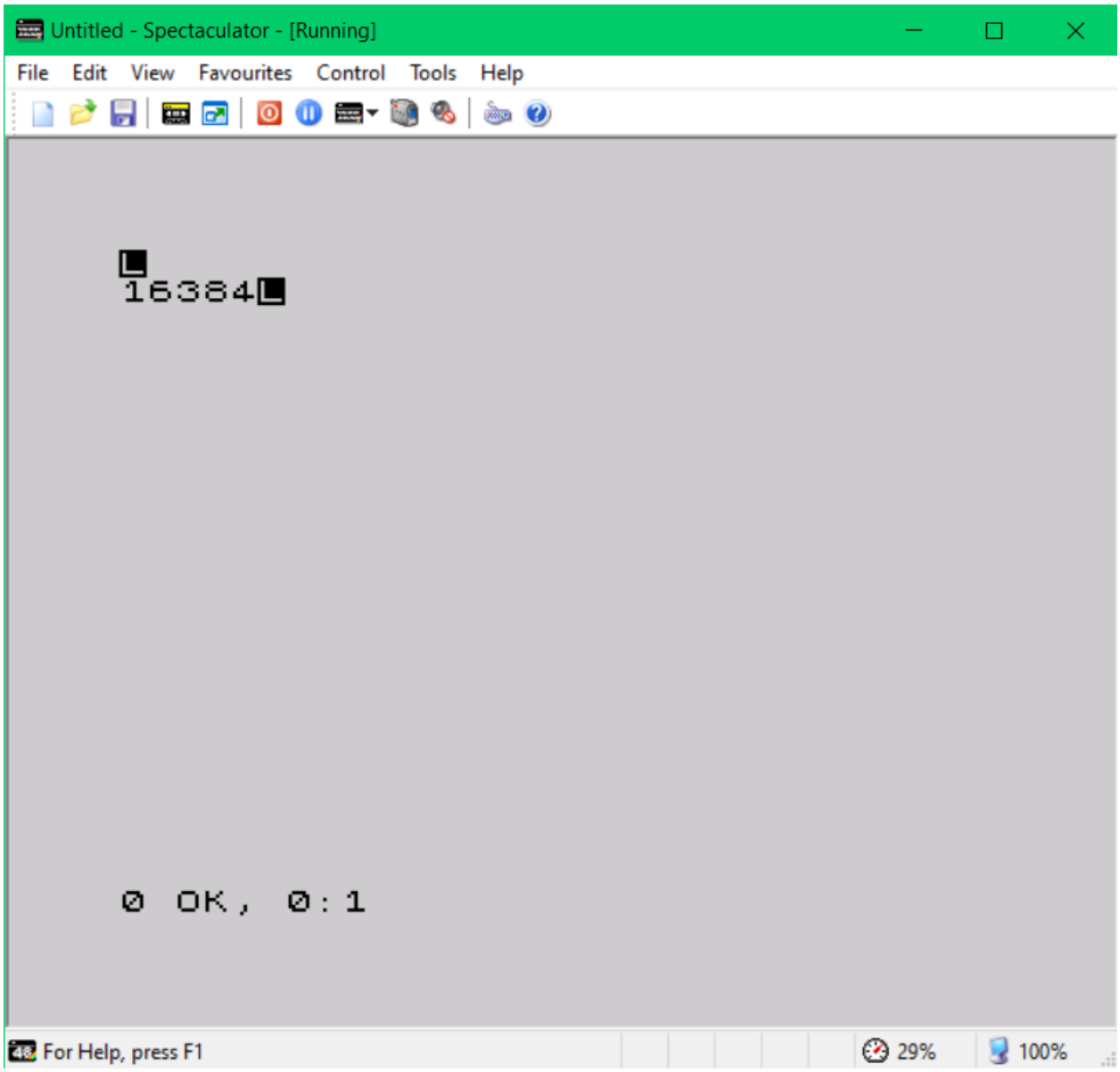


Рис. 487. Результат ввода данных с искажением через INPUT #2.

Пусть с определенными глюками, но строка корректно выполнилась и даже систему почти не перекособенило.

Нажмите «*Reset*» и можно сделать опыт №2. Предлагаю в блочках «K» и «S» поменять местами буквы их имён. В первую очередь введите искусственную BASIC программу:

```
Debugger
Dec
Go To 23560
23560 ← 13
23611 ← 32
23627 ← 23795
23641 ← 23796
23649 ← 23798 23798 23798
23755 ← 0 1 28 0 245
23760 ← ""СТРОКА В ВЕРХНЕМ СЕКТОРЕ
23785 ← 34 13 0 2 4 0 242 195 167 13 128 13 128
```

```
Trace
```

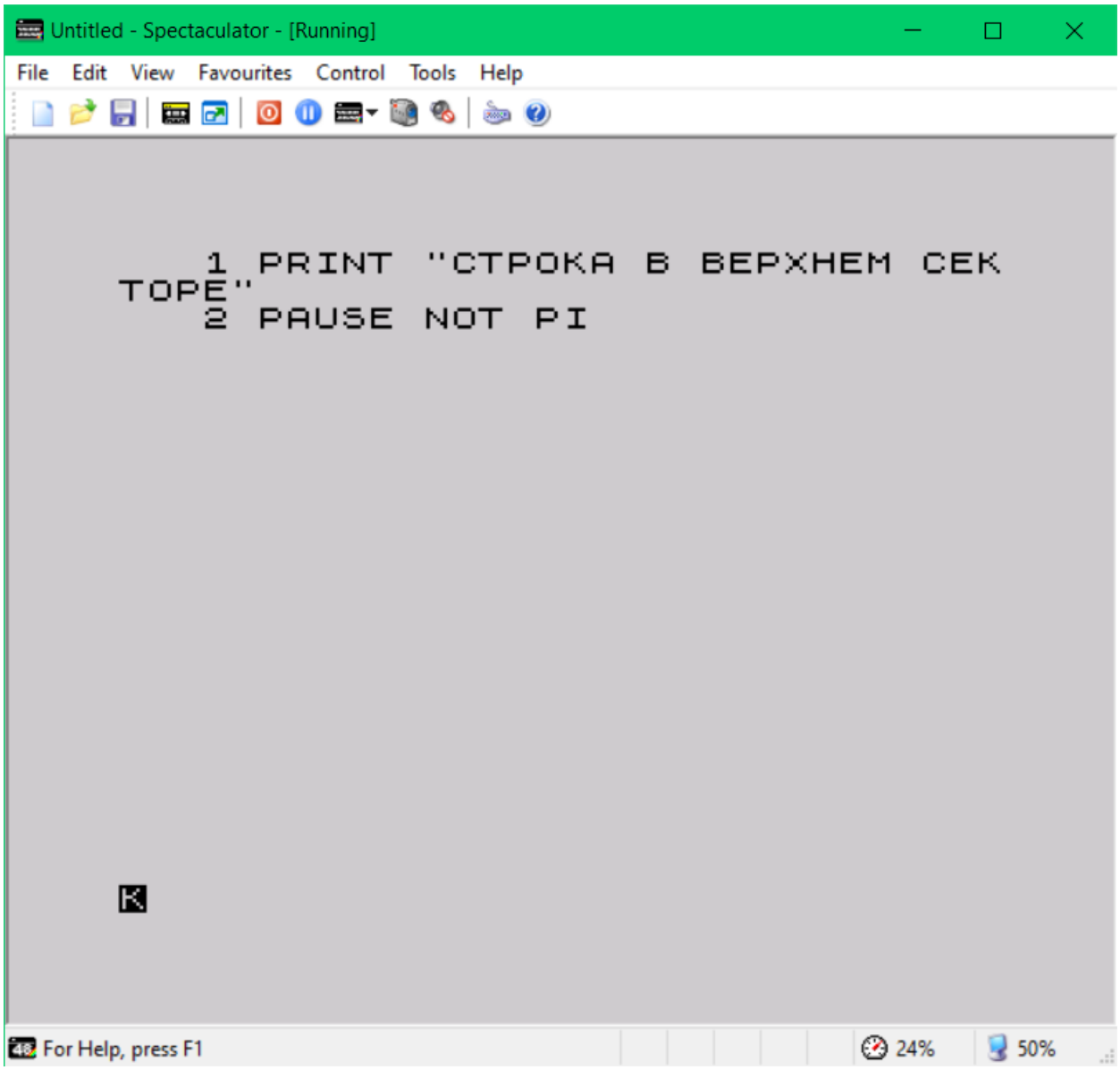


Рис. 488. Ввод BASIC программы для опыта.

А теперь добавьте следующую программу:

```
Debugger
Dec
23560 ← 13
23611 ← 32
Go To 23738
23738 ← 83
23743 ← 75
Trace
```

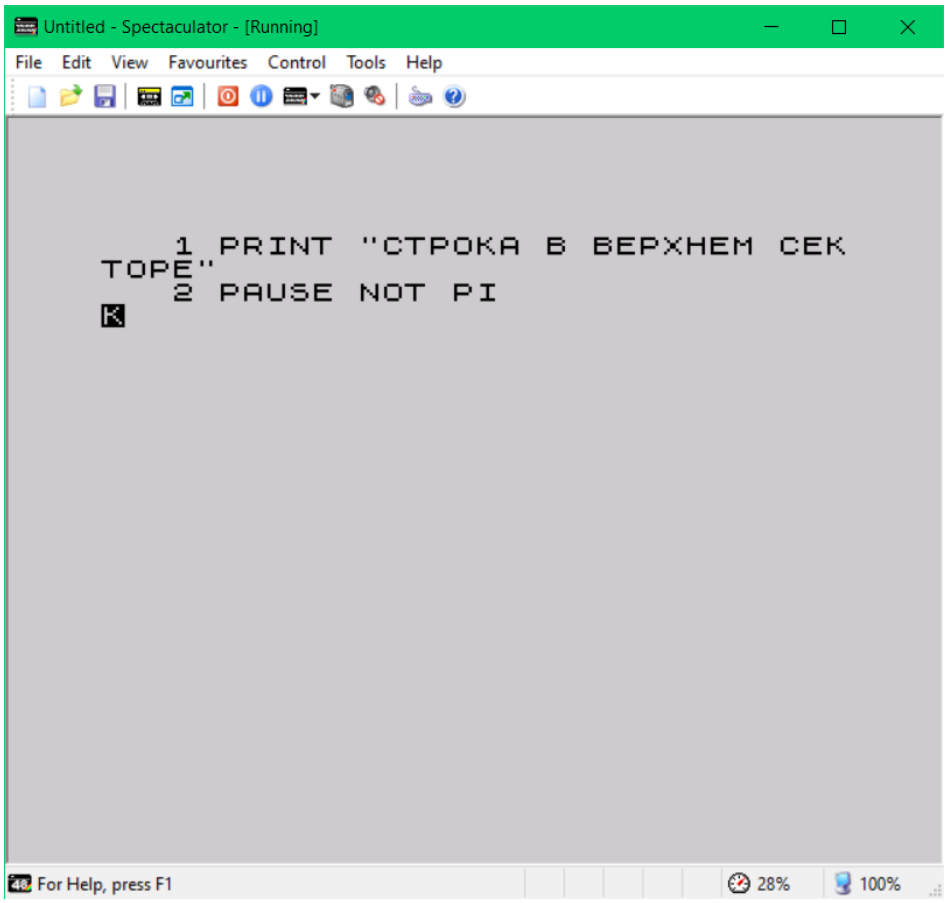


Рис. 489. Эффект курсора наверху.

А теперь запустите программу натуральным способом командой RUN:

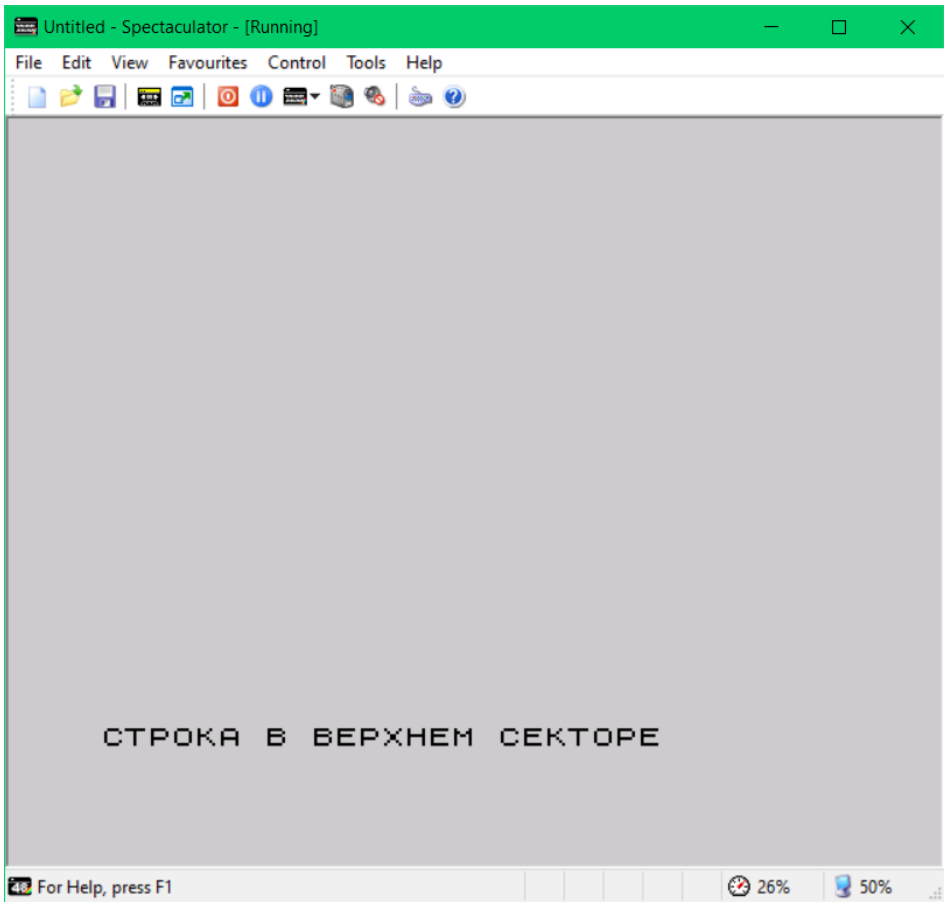


Рис. 490. Команда PRINT, выполнившаяся в нижней строке.

Все перевернулось. По команде `PRINT` произошел вывод в нижние строки. Нажмите любую клавишу:

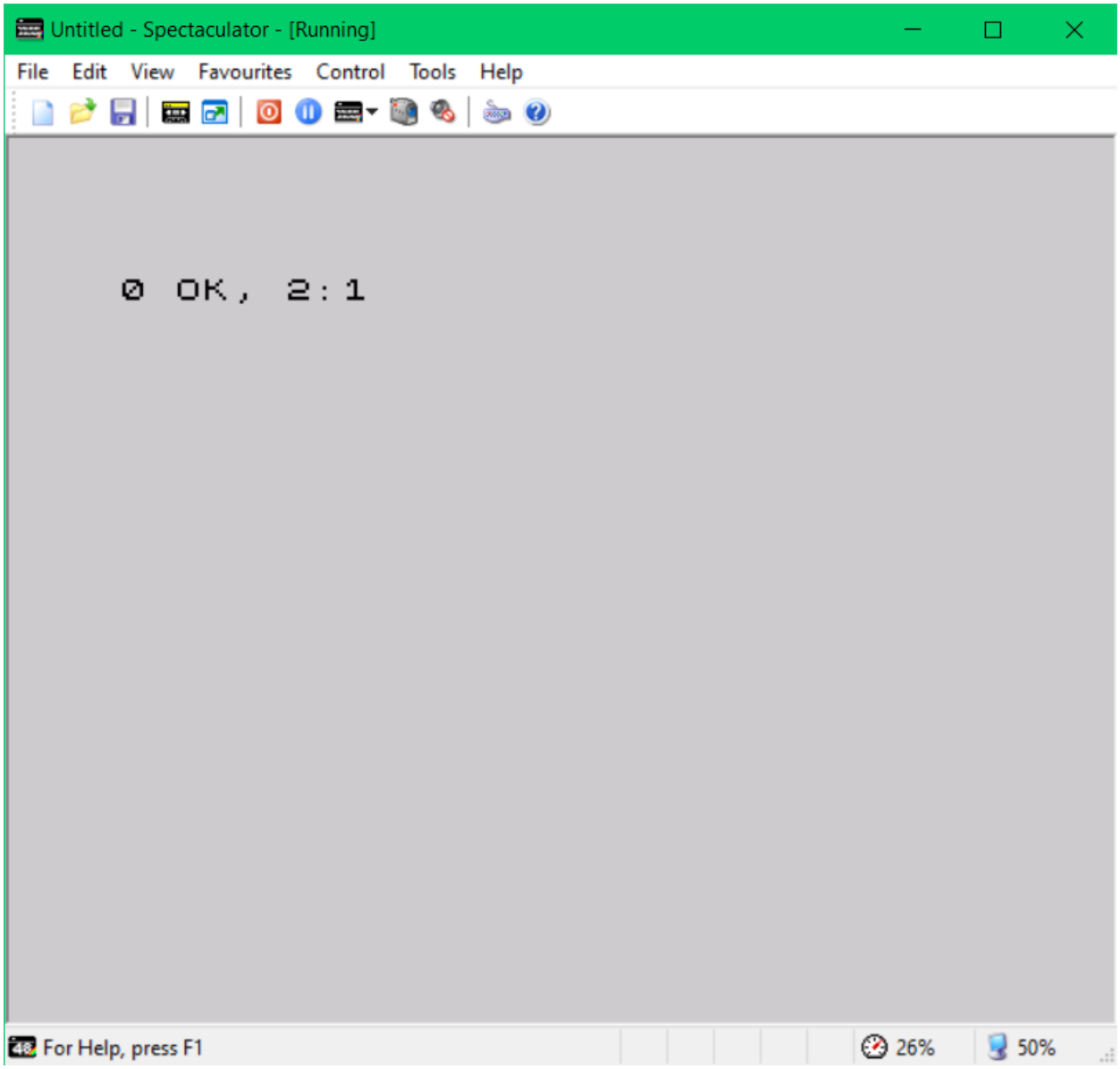


Рис. 491. Сообщение «0 OK» в верхней части экрана.

Сообщение «`0 OK`» выдалось в верхнюю часть экрана. Когда вы снова нажмёте любую клавишу, курсор продолжит держаться наверху экрана. Учтите, что затирание после ввода и редактирование работает не совсем корректно, поэтому после данного эксперимента верните буквы блоков на место или снова нажмите «*Reset*».

Как это можно использовать на практике? Допустим, для блокировки вывода заголовков во время загрузки. Например, поставьте на загрузку в виртуальный магнитофон какую-нибудь запись и введите:

```
POKE 23743,80: LOAD ""CODE : POKE 23743,83
```

Редактирование блоков это хорошо, но может возникнуть вопрос: а чем закрепляются эти самые числа после квадратика? А вот для этого существует таблица `STRMS` (*жарг. «стремный термос»*), которая также размещается в системных переменных.

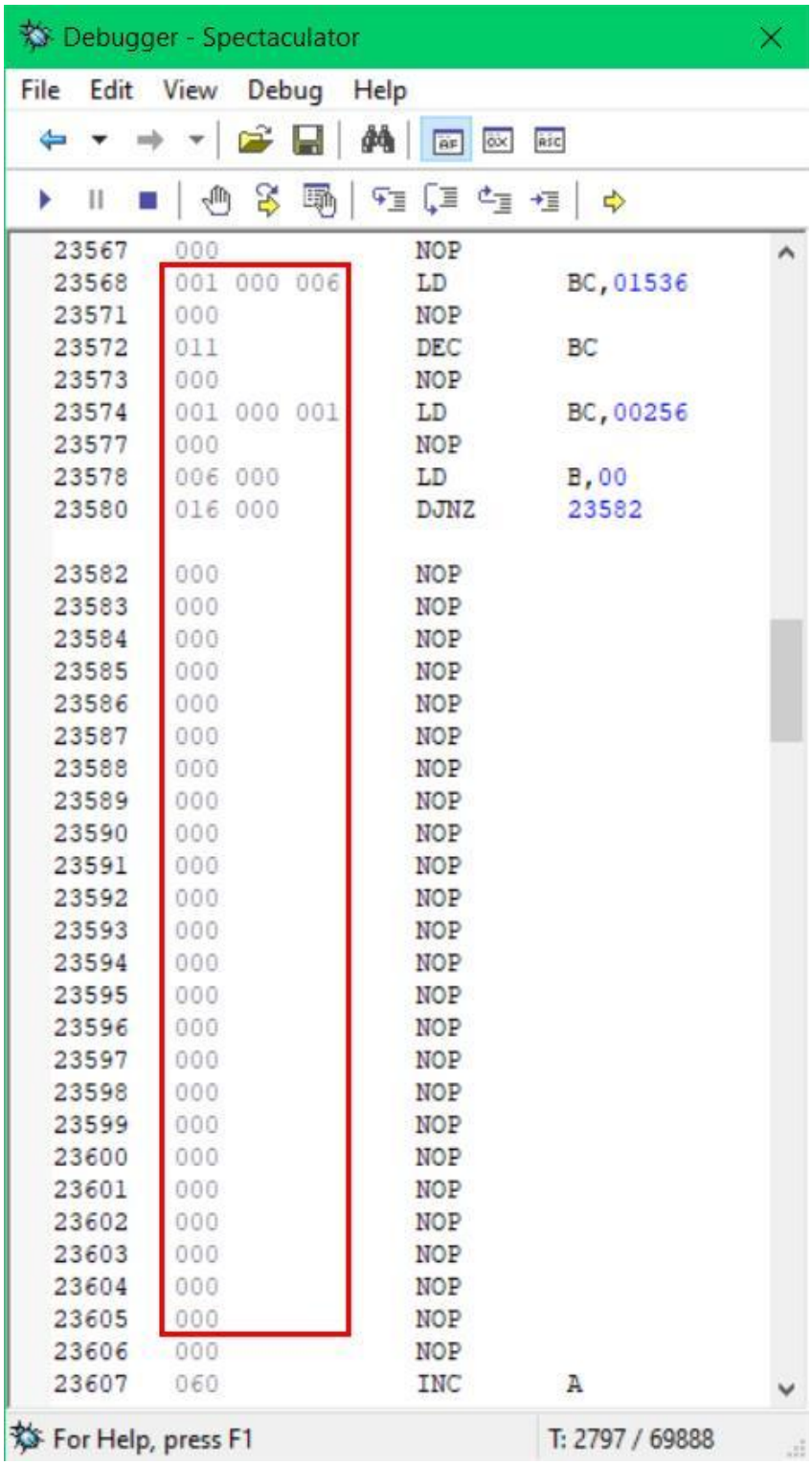


Рис. 492. Массив STRMS. Общий вид.

А вот с этим массивом картина печальная. Собственного указателя-посредника он не имеет. Все обращения идут по прямому адресу в это конкретное место. Для перемещения придется править все адреса, найденные в области ПЗУ. При формировании во время загрузки, из 5574 копируются только первые 14 байт. Остальная нижняя часть из 24-х ячеек оставлена для пользователей.

А теперь о структуре. Этот массив является указательно-распределительной таблицей к блокам CHANS.

Таблица состоит из 24-х секций. Каждая секция, соответственно, всего по 2 байта. По сути, отдельную секцию можно назвать парой. Первый байт паре означает смещение,

которое нужно прибавить к числу 23733, чтобы полученная сумма указывала на заглавный байт одного из блоков массива CHANS. Например, 1 будет означать 23733+1=23734. Это адрес начала блока «K». Смещение 6, будет означать сумму 23739, и укажет на начало блока «S». Из контекста понятно, что число для смещения должно быть 1, 6, 11 или 16.

Второй байт в паре – это банальный разделитель, состоящий из нуля.

На первый взгляд предназначение данной таблицы не понятно, но после краткого осмысления всё встаёт на свои места. В действительности STRMS закрепляет эти самые числа в BASIC командах, которые ставятся после квадратиков для корректировки свойств.

Есть у STRMS одно неоспоримое преимущество. Если CHANS можно редактировать только с помощью POKE, то значения пользовательских пар STRMS, по задумке разработчиков, можно частично корректировать командами OPEN #и CLOSE #. Использование этих команд в природе не замечено (кроме как с USR 0 для защиты), поэтому с какого-то момента они получили прозвище «легендарные».

Итак, вот как следует понимать данный массив:

№№ пары	Адрес пары	Дан-ные	Смещение от 23733	Указание на блок в CHANS	№№ пары для OPEN # и CLOSE #	Жаргонное название
253	23568	1, 0	1	K	-	VIP-1
254	23570	6, 0	6	S	-	VIP-2
255	23572	11, 0	11	R	-	VIP-3
0	23574	1, 0	1	K	#0	Базовая-1
1	23576	1, 0	1	K	#1	Базовая-2
2	23578	6, 0	6	S	#2	Базовая-3
3	23580	16, 0	16	P	#3	Базовая-4
4	23582	0, 0	0	Изначально не задан	#4	Народная-1
5	23584	0, 0	0	Изначально не задан	#5	Народная-2
6	23586	0, 0	0	Изначально не задан	#6	Народная-3
7	23588	0, 0	0	Изначально не задан	#7	Народная-4
8	23590	0, 0	0	Изначально не задан	#8	Народная-5
9	23592	0, 0	0	Изначально не задан	#9	Народная-6
10	23594	0, 0	0	Изначально не задан	#10	Народная-7
11	23596	0, 0	0	Изначально не задан	#11	Народная-8
12	23598	0, 0	0	Изначально не задан	#12	Народная-9
13	23600	0, 0	0	Изначально не задан	#13	Народная-10
14	23602	0, 0	0	Изначально не задан	#14	Народная-11
15	23604	0, 0	0	Изначально не задан	#15	Народная-12

Структура массива STRMS.

Обратите внимание, что корректировка спецкомандами BASIC доступна лишь с 4-й по счету пары. Понятно, что с помощью POKE можно легко отредактировать три верхние VIP пары, Например: POKE 23568,0. Но раз они так защищены от простого BASIC-пользователя, значит тут что-то нечисто.

Так оно и есть. Обнуление первого параметра вызовет искажение с последующим сбросом, второго невозможность ввести номерную строку и курсор с мигающим знаком вопроса. Обнулив пару «VIP-3» вы лишите себя возможности редактирования BASIC строки натуральным способом. VIP-ы они такие. Тронешь, потом неприятностей не оберешься.

Но ведь это не аргумент. Классовое неравенство не есть хорошо, поэтому будет интересно узнать, где именно происходит оберегание тонкой душевной организации VIP пар от команд народного BASIC'а. И вот для этого очередной раз придеться отправиться с Желтой Стрелочкой на прогулку, чтобы изучить то, из чего состоит команда OPEN #, на примере OPEN #4,"S".

Все начинается в 5942, куда Стрелочка попадает после распределения с `JUMP-C-R`. Там число, стоящее в команде после `#` преобразуется и кладется в «A». Оттуда она выходит в подпрограмму `STR-DATA (5918)`. Вот тут и находится важная часть с запретами и ограничениями:

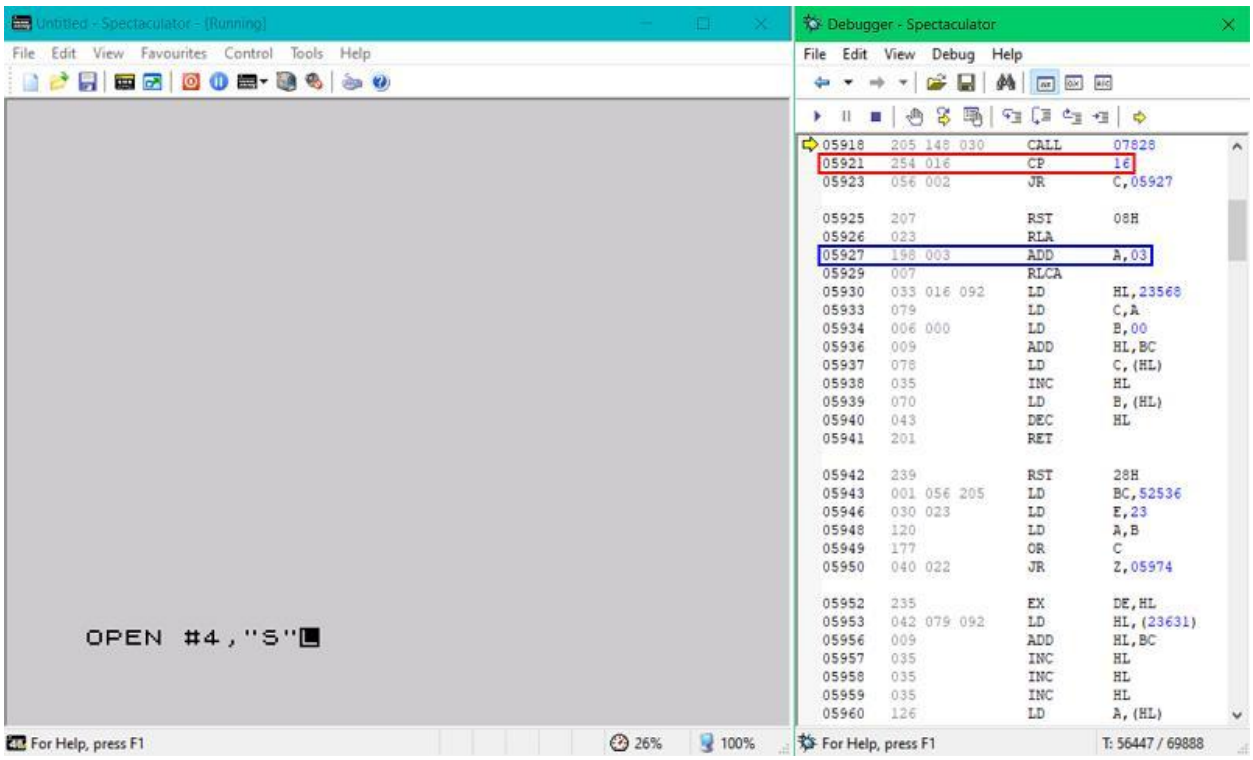


Рис. 493. Фрагмент программы STR-DATA. Ограничения параметров для команды OPEN #.

Начинается проверка (`5921 CP 16`). Если таким числом оказалось 16 и выше, то это перебор. Для `CLOSE #16` пары не существует, поэтому программа останавливается и выдается ошибка «`O Invalid stream`» (5925).

Если число в заданном диапазоне, то для проскока VIP пар к нему добавляется трёшка (`5927 ADD A,3`) и организуется нужное защитное смещение. Далее в «HL» берется адрес таблицы и копируется старое значение смещения из требуемой пары в «BC». Потом в зависимости от класса пар («Базовая» или «Народная») выполняются немного разные вычисления. В конечном итоге Стрелочка попадает на адрес 5977, где из «DE» записывает новые значения в выбранную пару массива `STRMS`.

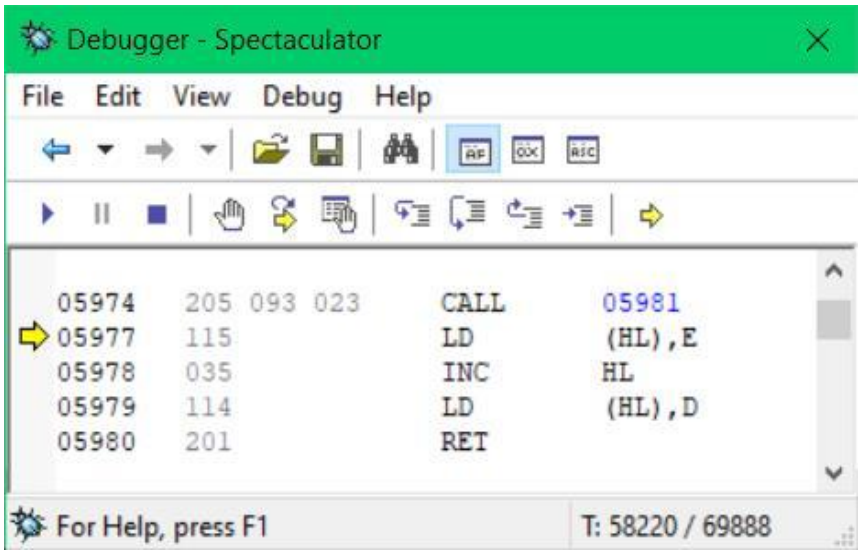


Рис. 494. Итоговая запись результата работы команды OPEN # в пару STRMS.

Таким образом, OPEN # это узкоспециализированный аналог команды POKE с ущемленными правами.

В таблице приведены все 48 вариантов применения команды OPEN #

Адрес пары в STRMS	Доступ командой OPEN #	POKE аналог
23574	OPEN #0,"K"	POKE 23574,1
	OPEN #0,"S"	POKE 23574,6
	OPEN #0,"P"	POKE 23574,16
23576	OPEN #1,"K"	POKE 23576,1
	OPEN #1,"S"	POKE 23576,6
	OPEN #1,"P"	POKE 23576,16
23578	OPEN #2,"K"	POKE 23578,1
	OPEN #2,"S"	POKE 23578,6
	OPEN #2,"P"	POKE 23578,16
23580	OPEN #3,"K"	POKE 23580,1
	OPEN #3,"S"	POKE 23580,6
	OPEN #3,"P"	POKE 23580,16
23582	OPEN #4,"K"	POKE 23582,1
	OPEN #4,"S"	POKE 23582,6
	OPEN #4,"P"	POKE 23582,16
23584	OPEN #5,"K"	POKE 23584,1
	OPEN #5,"S"	POKE 23584,6
	OPEN #5,"P"	POKE 23584,16
23586	OPEN #6,"K"	POKE 23586,1
	OPEN #6,"S"	POKE 23586,6
	OPEN #6,"P"	POKE 23586,16
23588	OPEN #7,"K"	POKE 23588,1
	OPEN #7,"S"	POKE 23588,6
	OPEN #7,"P"	POKE 23588,16
23590	OPEN #8,"K"	POKE 23590,1
	OPEN #8,"S"	POKE 23590,6
	OPEN #8,"P"	POKE 23590,16
23592	OPEN #9,"K"	POKE 23592,1
	OPEN #9,"S"	POKE 23592,6
	OPEN #9,"P"	POKE 23592,16
23594	OPEN #10,"K"	POKE 23594,1
	OPEN #10,"S"	POKE 23594,6
	OPEN #10,"P"	POKE 23594,16
23596	OPEN #11,"K"	POKE 23596,1
	OPEN #11,"S"	POKE 23596,6
	OPEN #11,"P"	POKE 23596,16
23598	OPEN #12,"K"	POKE 23598,1
	OPEN #12,"S"	POKE 23598,6
	OPEN #12,"P"	POKE 23598,16
23600	OPEN #13,"K"	POKE 23600,1
	OPEN #13,"S"	POKE 23600,6
	OPEN #13,"P"	POKE 23600,16
23602	OPEN #14,"K"	POKE 23602,1
	OPEN #14,"S"	POKE 23602,6
	OPEN #14,"P"	POKE 23602,16
23604	OPEN #15,"K"	POKE 23604,1
	OPEN #15,"S"	POKE 23604,6
	OPEN #15,"P"	POKE 23604,16

Варианты использования команды OPEN #

Команда CLOSE # по смыслу должна быть полной противоположностью OPEN #, но изучение маршрута Стрелочки в отладчике показали совсем иную картину.

Допустим, введена команда CLOSE #3:

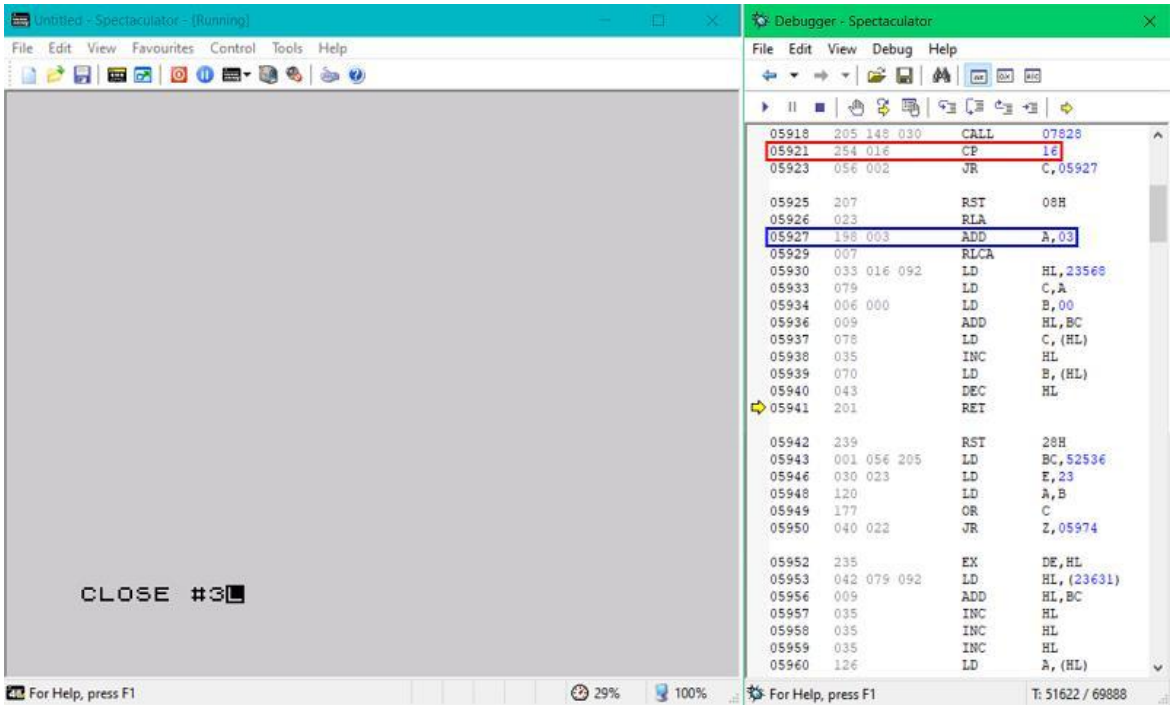


Рис. 495. Процесс выполнения команды CLOSE #3.

Попав на адрес 5861, Стрелочка тут же перескакивает на знакомую программу `STREAM-DATA (5918)`. Как и в случае с `OPEN #`, верхний предел значения задаётся по адресу 5921, а нижний в 5927. В этом плане у команд одинаковые корни.

А вот по выходу обратно в 5864 начинаются отличия. Обратите внимание, путем хитрых вычислений к финальной записи данных в 5885 вместо предполагаемого обнуления в «BC» подготовлено число 16:

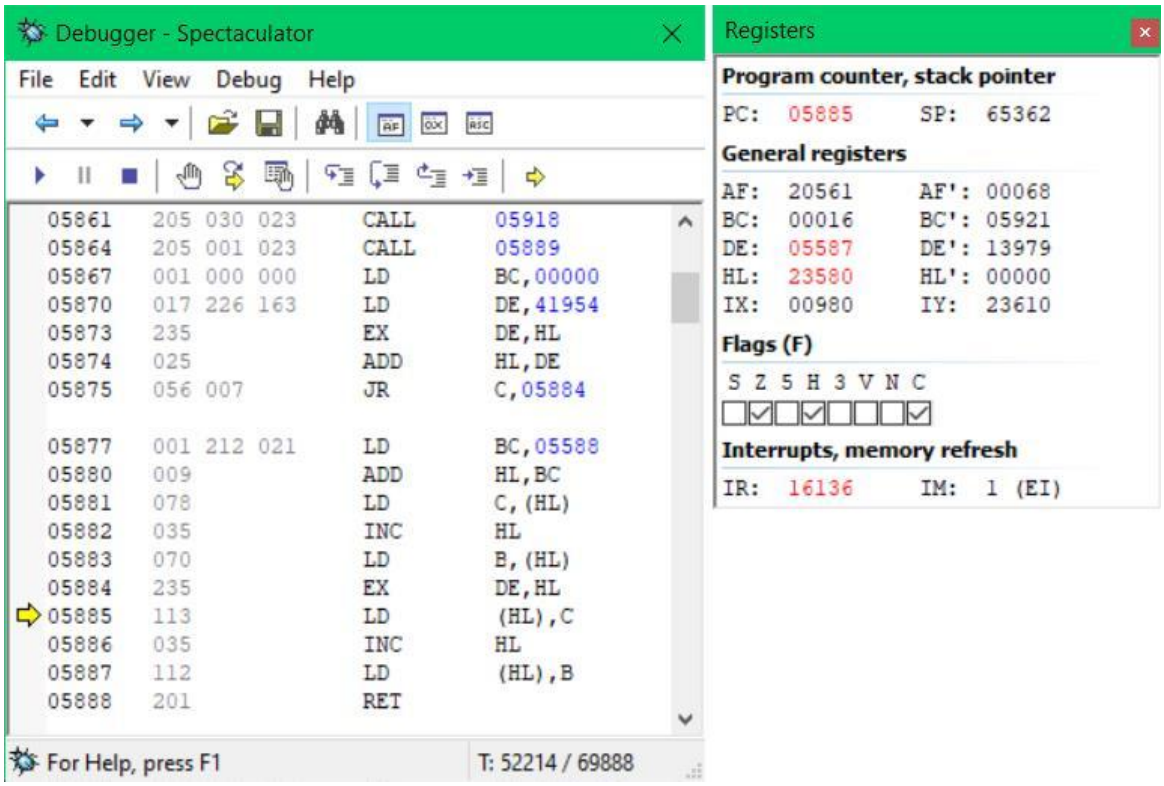


Рис. 496. Итоговая запись результата работы команды CLOSE # в пару STRMS.

Таким образом, при использовании команды `CLOSE #0...3` «Базовым» парам вместо обнуления возвращаются их родные значения. А «Народные» пары при работе

`CLOSE #4…15` обнуляются, поскольку изначально при старте компьютера они пустовали.

Таким образом, команда `CLOSE #` получается ещё более узкого профиля. Она лишь восстанавливает первоначальные значения в таблице `STRMS`.

Можно было бы закончить с этой командой, если бы не одна деталь. Как показала практика `CLOSE #` нельзя использовать, когда в паре обнулен байт смещения. Иными словами, `CLOSE #4…15` без предварительного `OPEN #` вызовет зависание. Это серьезная ошибка разработчиков, которая была известна еще в каменном веке и упоминалась в книгах «*Тайники ZX-Spectrum*» и «*Полное описание ПЗУ ZX-Spectrum*».

```
ТАБЛИЦА ЗАКРЫТЫХ ПОТОКОВ

1716              DEFB  4B 05              - канал 'K', смещение +05, адрес 171C
1718              DEFB  53 03              - канал 'S', смещение +03, адрес 171C
171A              DEFB  50 01              - канал 'P', смещение +01, адрес 171C

Примечание: В конце таблицы нет маркера конца.

ПОДПРОГРАММА 'CLOSE STREAM' ('Закрыть поток')

171C CLOSE-STR    POP   HL                 Выбор указателя на информацию
                  RET                      о канале и возврат.
```

9.12 Ошибка CLOSE

Попытка отключения потока 4...15 от канала до его подключения ведет к непредвиденным эффектам с рестартом системы включительно. А все потому, что в ROM в таблице, содержащей данные о каналах с адресом #1716 забыли разместить указатель конца таблицы.

Рис. 497. Упоминание ошибки CLOSE #. Вырезка из старых книг.

Чтобы понять причину более детально, нужно смотреть на практике:

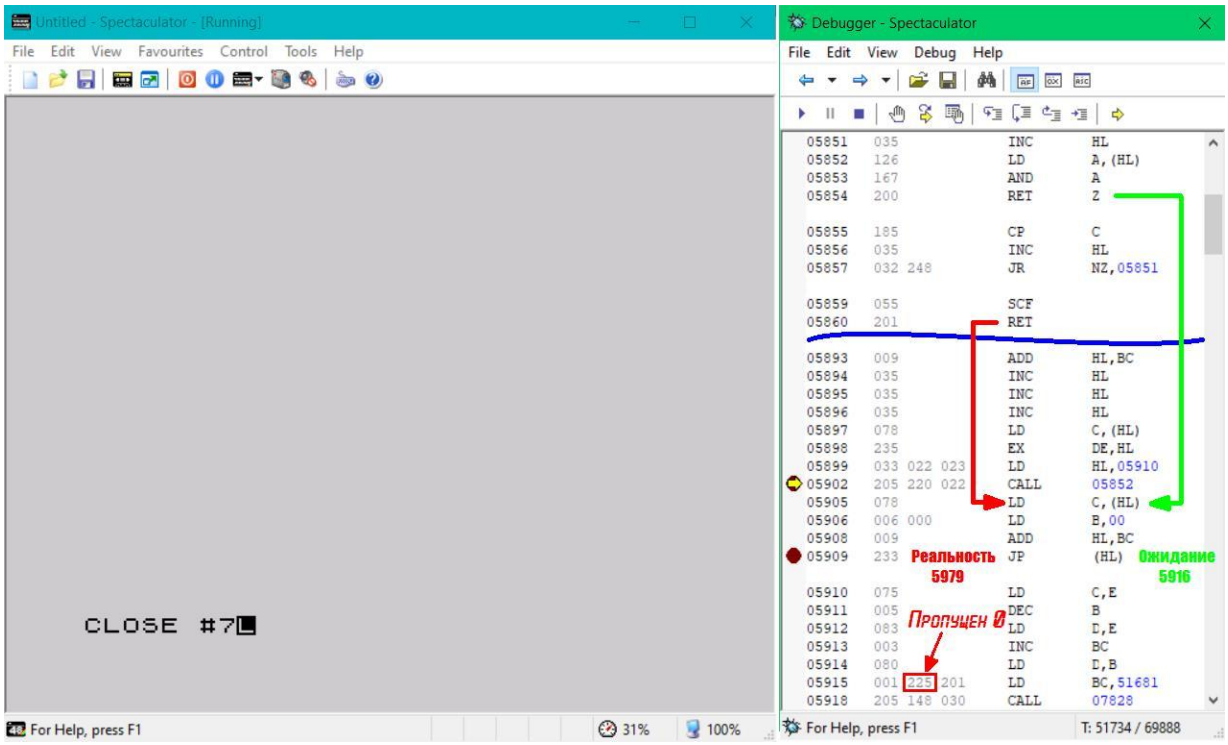


Рис. 498. Место сбоя выполнения CLOSE #4…15.

Сбой формируется по адресу 5902. Войдя в универсальную программу `INDEXER (5851)` начинается перебор таблицы, в конце которой, по адресу 5916 должен был стоять

0 (вместо 225). По плану, обнаружив его, Стрелочка должна вернуться по RET Z (5854) со значением в «HL» 5916. По возврату из программы индексации, записав в «C» число 0 из ячейки «HL» (5905 LD C,(HL). В сумме с нулем получается итоговый адрес 5916, по которому должен осуществиться переход.

В реальности при отсутствии 0 все это заканчивается плачевно. Не найдя конец таблицы, перебор продолжается до ячейки 5923. Сложив набранный адрес с имеющимся значением, получается 5979. Естественно в 5909 Стрелочка уходит по JP (HL) немного не туда, не забрав со столбика SP лишнего значения. Таким образом, она попадает в системные переменные, на адрес STRMS 23582...23604, в который нужно всего лишь записать байт.

Ну а дальше начинает выполняться белиберда из текущих системных переменных, налетает на встречные DI с RET'ами, после чего обычно наглухо виснет на HALT по адресу 4867. Сценарий будет меняться в зависимости от текущего состояния массива системных переменных.

На языке God Mode эта ошибка корректируется вводом следующей простенькой программы:

```
Debugger
Dec
Add Breakpoint 5909
HL ← 5916
Remove Breakpoint 5909
Trace
```

И снова несколько примеров. Чуть выше рассматривался метод замены имён блоков, в результате которого курсор и сообщения выводились в верхнюю часть экрана. Подменой смещений в VIP парах, можно добиться аналогичного эффекта. Для этого просто введите программу:

```
Debugger
Dec
Go To 23568
23568 ← 6
23570 ← 1
Trace
```

После запуска курсор замигает в верхних строчках. После завершения экспериментов, восстановите текущие значения или нажмите *Reset (CTRL+R)*.

Таблицу STRMS можно также использовать для подавления вывода заголовков во время загрузки «Пи-и-и Тшу-у-у». Для этого в паре «VIP-2», перед загрузкой блока можно поставить значение 16 (адрес блока «P»), а после загрузки снова восстановить 6:

```
POKE 23570,16: LOAD ""CODE : POKE 23570,6
```

А теперь предлагаю применить накопленные знания и создать собственное цветное сообщение об ошибке. Оно будет всякий раз выскакивать и останавливать BASIC программу при вводе или запуске, в которой обнаружится комбинация INPUT #2 с переменной для ввода.

Введите следующую нехитрую программу:

```
Debugger
Dec
Go To 23741
23741 ← 30000
30000 ← 253 54 55 0 253 54 2 33 17 76 117
30011 ← 1 30 0 205 60 32 49 84 255
```

```
30020 ← 237 115 61 92 241 195 169 18 16 2
30030 ← X Hexep mym INPUT #2 nucamb!
Trace
```

Пока еще ничего не произошло, но контексту нетрудно догадаться, что будет немножко весело.

После запуска натуральным способом введите следующую BASIC программу:

```
1 PRINT "gocka"
2 INPUT #2;a$
3 PRINT "u mpu cocka"
```

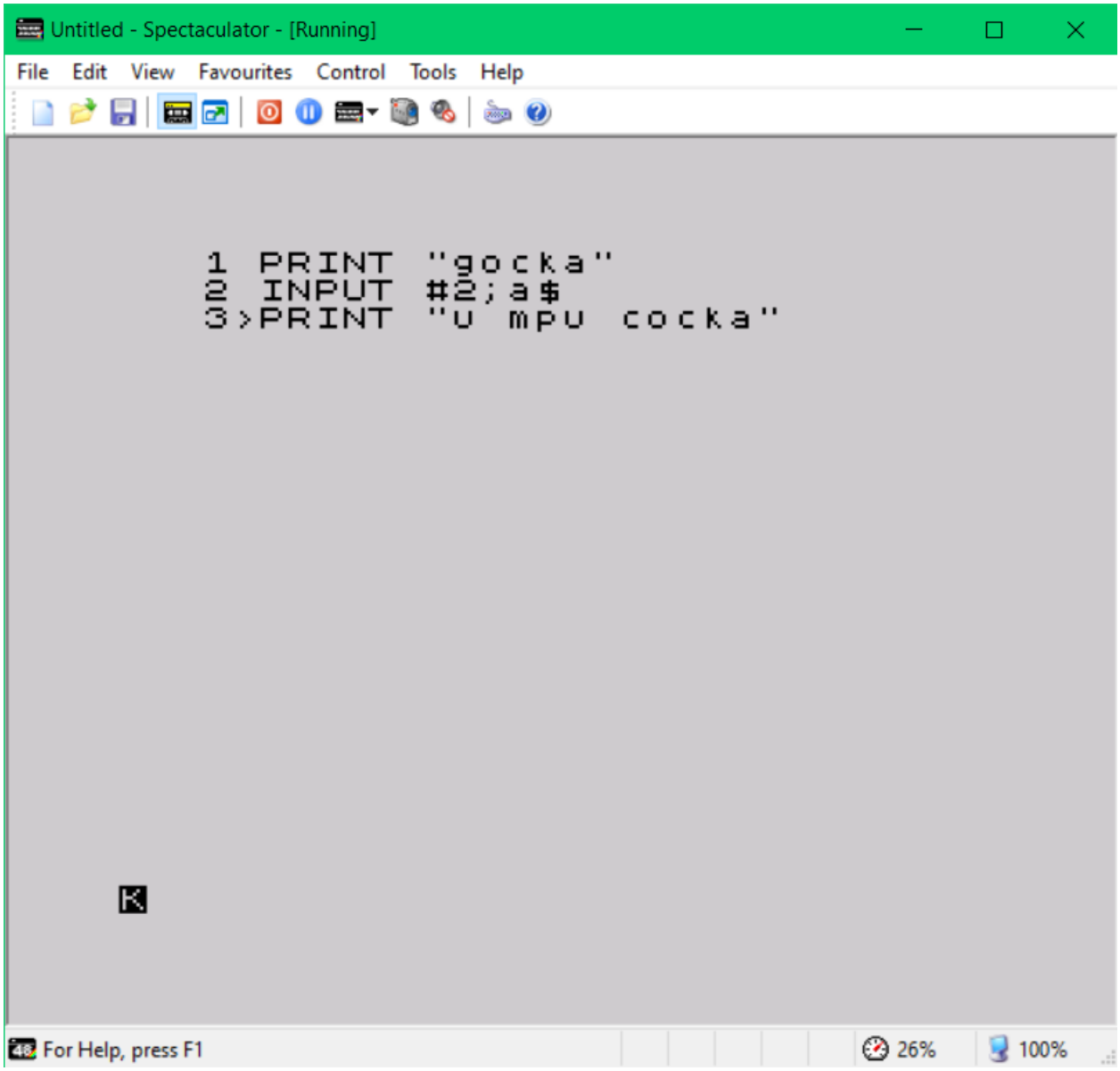


Рис. 499. Введенная BASIC программа для тестирования сообщения об ошибке.

А теперь запустите:

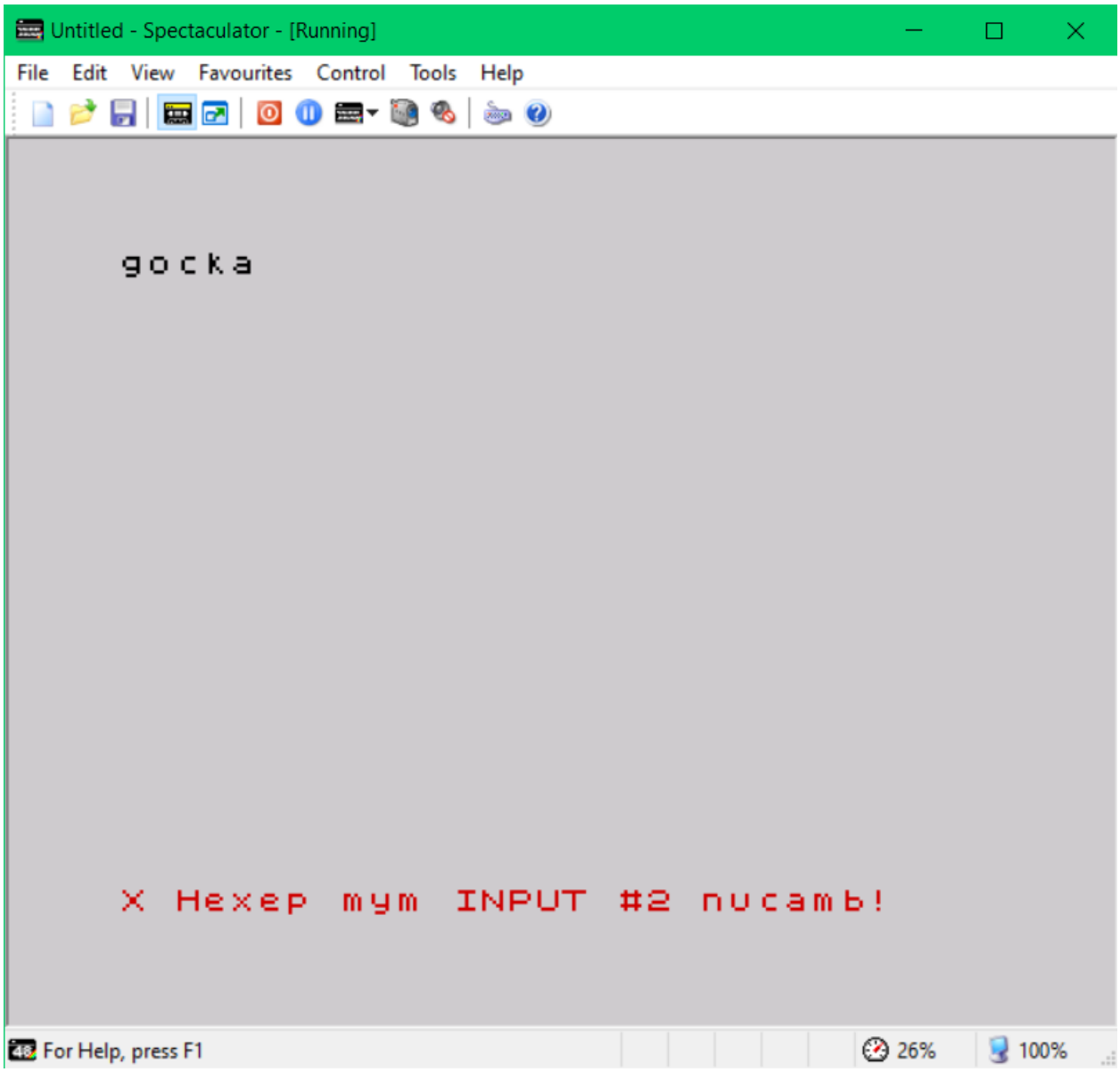


Рис. 500. Собственное цветное сообщение об ошибке.

В итоге только «доска» и вот такое красное сообщение с кодом «Хэ». Оно появится всякий раз, когда встретится `INPUT #2` с любой переменной и будет вежливо намекать о том, что вы немного не правы.

Но такой метод прокатит не везде. Если вы попытаетесь подменить адреса блока «K» или первый адрес в «S», то вызов собственной программы будет работать ровно до момента нажатия `ENTER` или ввода корректной номерной строки.

Предлагаю снова посмотреть в отладчик:

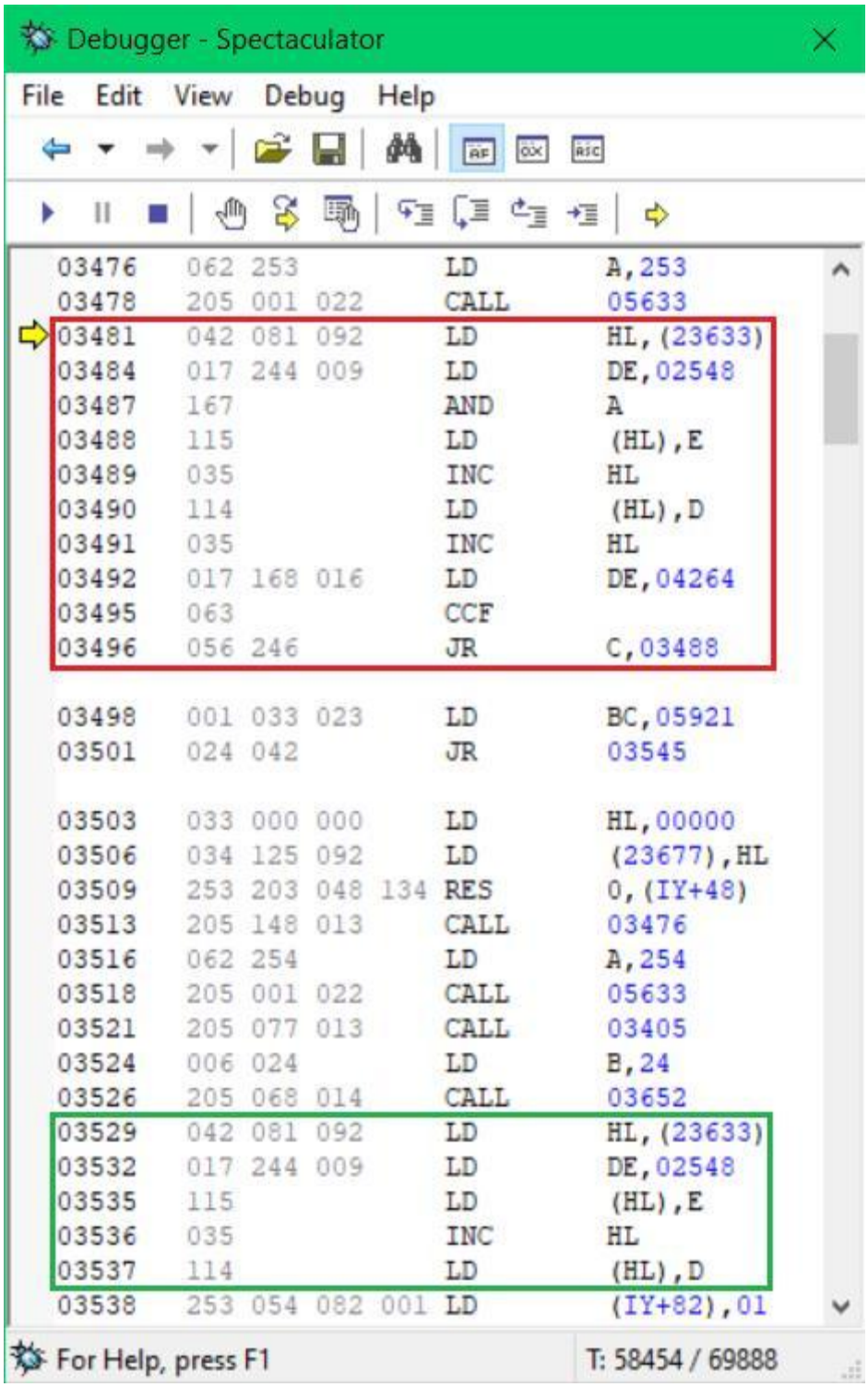


Рис. 501. Программы CL-CHAN и CL-ALL.

В восстановлении блока «K» виноват фрагмент программы `CL-CHAN` по адресу 3481-3496. Пока печатаются символы без ввода, программа на подмену будет работать, но при нажатии `ENTER` вся сказочка закончится.

С блоком «S» всё аналогично, только в данном случае обновляется адрес первой программы в блоке, заточенной под работу `PRINT`.

«А нахухуа обновлять второй?» – подумали разработчики прошивки. Ведь во втором адресе банальная заглушка с выводом ошибки «`J Invalid I/O device`». Кто же в здравом уме будет менять тот адрес. Вот именно на этом и основан прошлый пример с выводом цветного сообщения.

Ну а прежде, чем приступить к созданию собственных блоков, нужно внимательно изучить работу основных подпрограмм, которые влияют на CHANS и STRMS. Опыт прошлого раздела с автозагрузками показал, что применяя метод тыка, желаемый эффект можно искать годами.

Прежде всего, нужно выяснить, какие команды позволят изменить себе свойства, принимая число с # квадратом. Я упоминал (L)PRINT'ы, (L)LIST'ы, INPUT и INKEY$. А вдруг есть что-то ещё? Литературе, типа дизассемблера ПЗУ, конечно можно доверять, но лучше перепроверить на практике. Книга на то и книга, но мнение Желтой Стрелочки в отладчике имеет больший авторитет.

Узнать это очень просто. Для этих целей имеется специальная программа ALTER STREAM по адресу 8304. Все команды, склонные к смене ~~ориентации~~ свойств, будут проходить через эту подпрограмму.

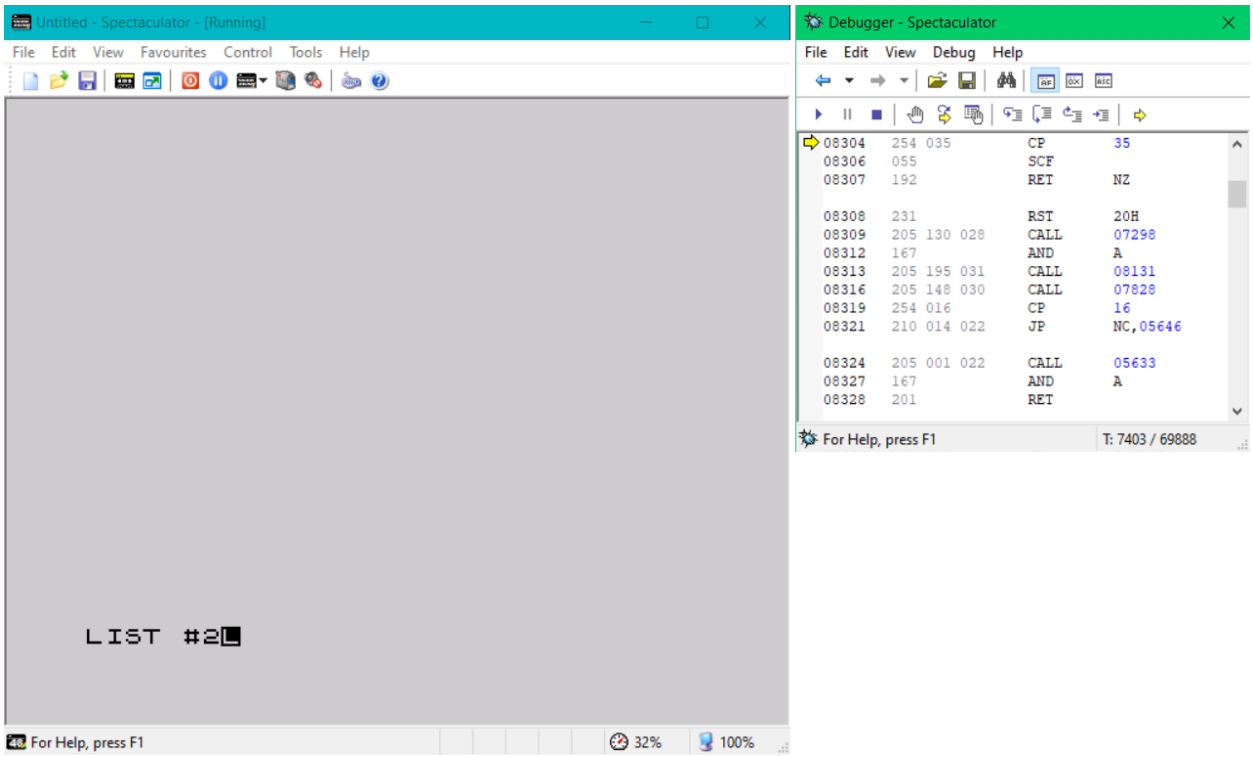


Рис. 502. Программа ALTER STREAM во время ввода LIST #2.

После проверки на символ # сдвигается BASIC-курсор CHADD (23645/46), чтобы посмотреть какое значение в строке стоит следом. После извлечения, обработки и пересылки в «BC» происходит проверка числа на допустимый диапазон (8319 CP 16). Как и в случае с OPEN #, остальные команды также должны иметь значение после квадрата не больше 15. Если обнаруживается #16 и больше, то выводится ошибка «O Invalid stream».

Если число в расчетном диапазоне, то перед выходом, для коррекции значения, последний раз вызывается… культовая программа CHAN-OPEN по адресу 5633.

О да! Эта живая легенда, которая тревожила душу с первых дней появления книги «Машинные коды» осенью 1996 года. Что же такое «Открыть канал», да и как на самом деле выглядит этот «открывательный» агрегат CHAN-OPEN по адресу 5633. Итак, в «A» лежит номер пары STRMS, после чего Стрелочка заходит ТУДА!

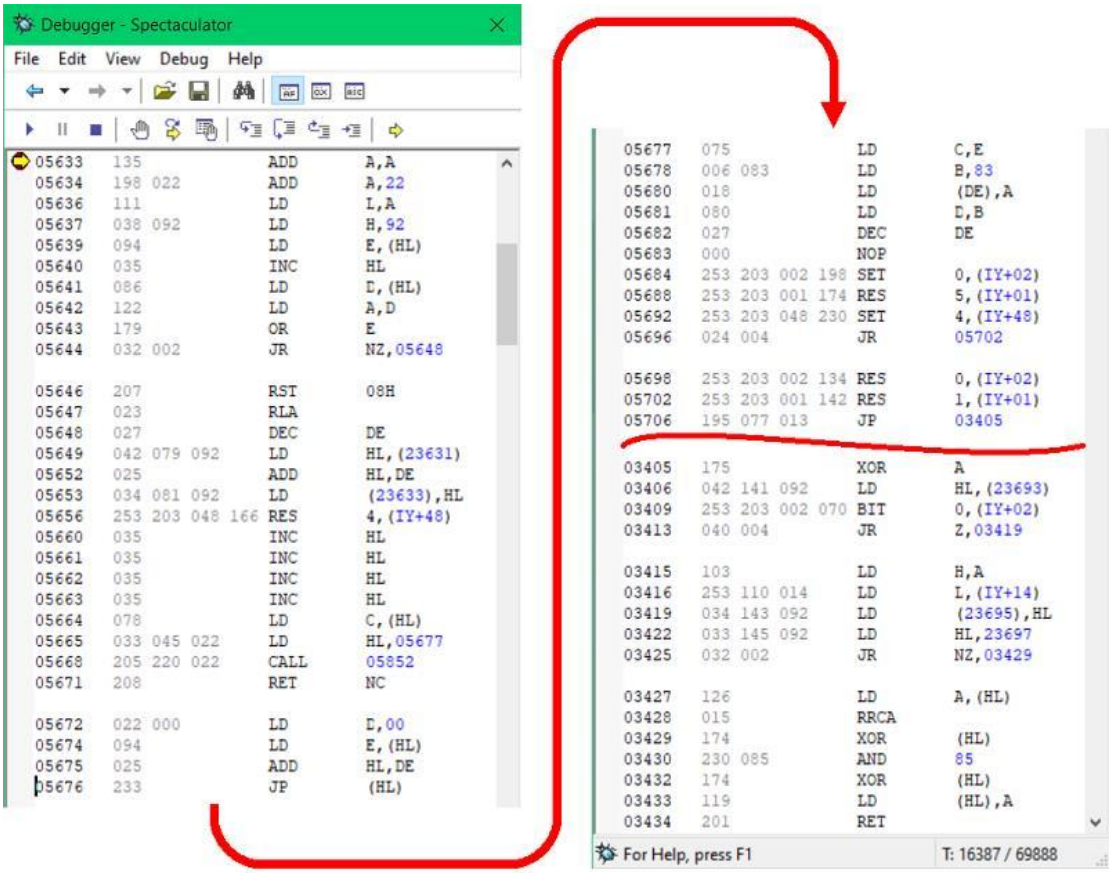


Рис. 503. Легендарная программа CHAN-OPEN.

Удвоив номер пары STRMS и прибавив 22, получается смещение, которое незамедлительно записывается в «L» (5636). В «H» добавляется старшая составляющая адреса 92 (92*256=23552). Таким образом, на выходе в «DE» получается адрес пары STRMS, из которого следует прочитать смещение на складе блоков в CHANS. В конкретном случае из номера #2 рождается адрес 23578.

Следом проводится быстрая проверка. Если пара ячеек содержит нули (5643), значит нужно выдать сообщение «Invalid stream».

Путём вычитания лишней единицы, полученное смещение корректируется и складывается с набором ячеек CHANS (23631), чтобы слепить адрес начала требуемого блока (5652). Полученное местонахождение блока переписывается в пару CURCHL (23633/34), которая является указателем.

Авансом выключив тумблер №4, который отвечает за курсор K (FLAGS2 23658 IY+48), пролистывается адрес блока до его буквенного имени (5663). Из полученного адреса в «C» забирается буковка (K, S, R или P).

Берется адрес таблицы смещений для задания действий (5665 LD HL,5677). По печально известной программе INDEXER (5852), на которой зависает CLOSE #, ищется смещение. В этот раз всё проходит нормально, так как в этой таблице не забыли поставить нолик в конце.

Подкорректировав выбранное значение, по полученному адресу происходит переход 5676 JP (HL) на набор переключения битов для требуемого блока из CHANS.

В данном случае, стрелочка попадёт на адрес 5698. Там выключится тумблер №0 (TV_FLAG 23612) для отмены вывода в нижние строки и отключается тумблер №1 (FLAGS 23611), отвечающий за вывод на принтер, после чего Стрелочка переходит в подпрограмму TEMPORARY COLOUR ITEMS по адресу 3405. Там смотрится, в какую часть экрана выводится символ. Если в нижнюю, то для контрастности корректируется цвет выводимых символов, в зависимости от цвета рамки. После проверки цветов происходит выход (3434 RET) туда, откуда она была вызвана.

Знакомый материал из первой половины этой книги? А как это мудрёно называется в литературе «Открыть канал».

На повестке дня остаётся самый последний и самый важный вопрос: где и на каком моменте происходит перехват и переход на альтернативные адреса, заданные в блоке CHANS? И этой программой является… WAIT-KEY (5588), а точнее её самые нижние уровни, где живёт Стрелочка при загрузке BASIC и бездействии системы.

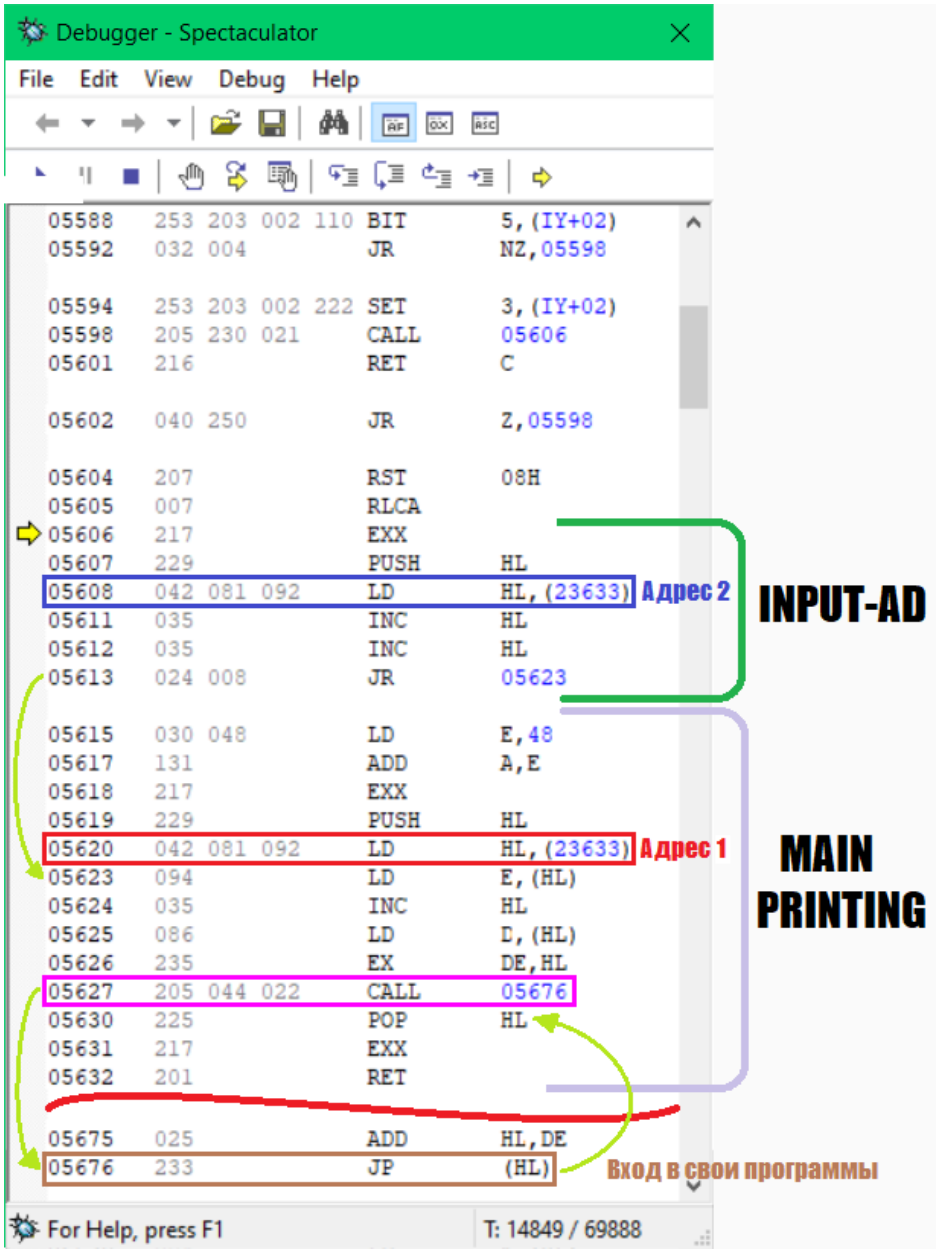


Рис. 504. Программа WAIT-KEY. Схема перехода по адресам из блока CHANS.

Вот так. Новое – это хорошо забытое старое из главы 5 «*Тайны подземелья EDITOR*». Правда забавно, как хитро переплетены все программы, а с ними главы и сюжет этой книги? А вот теперь предлагаю взглянуть на эту программу с другого ракурса.

Сначала рассмотрю перехват по команде PRINT. Этот адрес первым стоит в блоке CHANS.

Предположим, при наборе PRINT #5 с адреса 8141 пойдёт долгая подготовка к выполнению команды по плановому сценарию. Настаёт момент, когда в программе ALTER STREAM (8304) из BASIC строки подцепляется корректирующая приставка #5. В программе CHAN-OPEN по вытащенному значению задаётся смещение в паре STRMS. Следом находится и сам новый блок, который записывается в CURCHL (23633).

Далее берется буква самодельного блока. В попытке идентифицировать, она сверяется с базой данных в ПЗУ. Поскольку информацию о новом блоке не найти, то в 5671 по RET NC происходит преждевременный выход без дальнейшего переключения тумблеров и подготовки цветов. После каскада выходов продолжается подготовка к выполнению команды PRINT.

Настаёт момент, когда Стрелочка подходит к кульминационной точке выполнения вывода символа по команде RST 16. В зависимости от ситуации, адресом отправки может стать 8256 и 8186. Дело в том, что команда PRINT вызывает блок вывода дважды. Первый раз для вывода символов в кавычках (8252), а второй когда в конце строки нужно вывести ENTER для перехода на следующую. RST 16 – это не что иное, как часть подпрограммы MAIN PRINTING. В общем, из улавливающего адреса 16 происходит перенаправление в 5618.

Дальнейшие события уже можно наблюдать на картинке. Из CURCHL (23633) забирается адрес созданного блока, вычисляется его нахождение и кладётся в «HL». Обратившись к кончику хвоста CHAN OPEN (5627 CALL 5676), по JP (HL) происходит переход на модуль с адресом собственной программы.

Таким образом, перехват происходит внутри процедуры RST 16 и вставленный модуль становится частью программы ПЗУ. Дальше полная свобода действий: от возврата назад для продолжения прерванной программы до собственных идей.

Если в своей программе поставить просто RET, то главная функция текущей команды PRINT со всеми параметрами будет проигнорирована. Ничего так и не напечатав, произойдет выход наверх для выполнения следующей команды в очереди. А вот она будет выполнена в полном объёме.

С переходом по команде INPUT на адрес №2 история очень похожая. После распознания BASIC оператора, Стрелочка заходит на адрес 8329, где начинается долгая подготовка к выполнению сценария команды.

После захода на ALTER STREAM (8304) с установкой указателя, в 5671 происходит преждевременный выход по команде RET NC, потому что имя самодельного блока в таблице ПЗУ не находится. Незапланированный выход, также как в ситуации с PRINT'ом, происходит без дальнейшего переключения тумблеров и проверки цветов. Продолжается подготовка выполнения команды INPUT.

Настаёт торжественный момент, когда по адресу 8530 происходит переход в программу EDITOR (CALL 3884), которая используется для BASIC редактора. В данном случае она вызывается отдельным слоем для создания образа команды INPUT.

Далее события будут развиваться как в главе 5, но с некоторой оговоркой. Точно также, в 3896 по CALL 5588 Стрелочка проваливается в WAIT-KEY. С этого момента, развитие ситуации показано на картинке выше.

В 5598 включив тумблер №3 TV_FLAG (23612), Стрелочка спускается этажом ниже в INPUT-AD (5606). Из CURCHL (23633/34) забирается адрес созданного блока и пролистывается первый адрес, предназначенный для PRINT. Далее Стрелочка добирается до подпрограммы MAIN PRINTING и продолжает вычисление адреса для перехода.

Аналогично ситуации с PRINT, на 5627 происходит обращение к хвостику CHAN OPEN и в 5676 по JP (HL) переход на свой модуль программы.

Перехват адреса произойдёт ровно на моменте, когда вот-вот должен замигать курсор и начаться опрос клавиатуры стандартным BASIC редактором, который сейчас, как шпион отыгрывает легенду, замаскировавшись под команду INPUT.

В этой ситуации одним RET'ом не отделаться, поскольку для прохода дальше нужен какой-нибудь опрос клавиши (*опять см. главу 5*). Начнется замкнутый цикл метания «Своя программа-WAIT-KEY» с зависанием, если в собственной программе не создать условия для продолжения работы сбившейся системы.

Программа WAIT-KEY на отметке 5601, бездействие вклиненной программы тупо не пропустит. Нужно, как минимум, ставить галочку ☑ C для возврата в уровень EDITOR'а и обратно в комплекс INPUT. Естественно, далее вся сложная система уже собьётся и нужно корректировать маршрут и сопровождать Стрелочку до следующей команды.

Какова же реальная концепция этих «Каналов и потоков»? Представьте себе простой инструментальный набор отвёрток с ключами-трещотками. Несмотря на пару десятков головок и отверточных биток, модульных оснований для всех съемных переходников две штуки. Плюс дополнительно в набор прилагается адаптер-посредник. Извлекая из ячеек головки и втыкая в любое основание через адаптер, вы можете собрать либо ключ с нужным калибром, либо отвертку с подходящим шлицем.

Рис. 505. Типовой набор модульных инструментов. Фото из интернета.

Примерно так и с программами ввода-вывода в ПЗУ *Spectrum'a*, которыми формируются команды PRINT / LPRINT и INPUT. Вот эти два основания – базовый комплекс PRINT и INPUT. Адаптер, это программа WAIT-KEY, а сменные насадки это подключаемые модули, адреса которых извлекаются из блока CHANS и подсоединяется с помощью трамплинчика 5627 CALL 5676 → 5676 JP (HL).

Разработчики предоставили возможность продвинутому пользователю самому менять насадки, но только те которые будут состыковываться с адаптером и основанием. Ведь в случае с инструментами всё точно также. Обратные стороны насадок должны иметь стандартные разъёмы, чтобы плотно вставлялись в основание и сохраняли работоспособность в процессе монтажа. Закончив пользоваться инструментом – разбери и положи в набор.

Похоже, разработчики системы сами не смогли дать удачную жизненную аналогию работе своего детища. Ну а авторы литературы тех лет, почитав техдокументацию на компьютер, в знак уважения просто высасывали сравнения из пальца, вместо того, чтобы подобрать более подходящие ассоциации. Ведь изобретатель всегда прав?

Да, возможно в 1982 году не существовало подобных наборов инструментов, но можно было весь этот алгоритм обернуть как-то удачнее. Например, упомянуть прожектора со сменными светофильтрами для концертных залов. Если хорошо подумать, нашлось бы несколько удачных сравнений. Ну а с каналами, которые типа «бухты с заливами», вышло как-то несуразно.

Теперь, когда все встало на свои места, предлагаю ввести программу, которая отодвигает BASIC, формирует собственные блоки в CHANS и закрепляет их за парами в STRMS. При таком подходе можно сколько угодно запускать программы с командами, не боясь самовосстановления адресов из ПЗУ во время ввода или очистки экрана.

Простое вмешательство с продолжением выполнения базовой программы приводилось в «*ZX-Ревю*» за 1992 год. Это не сильно интересно. Предлагаю более изысканную «колхозно-бунтарскую» модель, где происходит полный перехват управления с деактивацией программы из ПЗУ и корректным возвратом в BASIC или к следующей команде после двоеточия.

```
Debugger
Dec
Go To 23641

; смещение BASIC области вниз на 10 ячеек

23627 ← 23765
23635 ← 23765
23641 ← 23766 23766
23649 ← 23768 23768 23768
23765 ← 128 13 128

; дописка новых блоков к массиву CHANS

23754 ← 30000 30100
23758 ← X
23759 ← 30200 30300
23763 ← E
23764 ← 128

; закрепление нового блока за парой 5 и 9 в таблице STRMS

23584 ← 21 0
23592 ← 26 0

; программа для адреса-1 блока X (окончание текста)
```

```
30000 ← 62 254 205 1 22 17 68 117 1 10 0
30011 ← 205 60 32 217 1 0 0 217 201 16 2
30022 ← 66 32 75 65 80 69 84 69
```

; программа для адреса-2 блока X (звук)

```
30100 ← 17 5 1 33 218 4 205 181 3 49 84 255
30112 ← 237 115 61 92 253 203 2 158 195 118 27
```

; программа для адреса-1 блока E (рамка)

```
30200 ← 62 1 211 254 49 84 255 237 115 61 92
30211 ← 253 203 2 158 195 118 27
```

; программа для адреса-2 блока E (начало текста)

```
30300 ← 62 254 205 1 22 17 120 118 1 26 0
30311 ← 205 60 32 49 84 255 237 115 61 92
30321 ← 253 203 2 158 195 118 27 16 3
30330 ← НЕВЕСТА СТРАСТНО СТОНЕТ ; в конце один пробел!
Trace
```

А теперь натуральным способом введите BASIC программу:

```
1 LET a$=INKEY$#9
2 INPUT #5;numep
3 PRINT #5;"ropog pagocmu"
4 LIST #9
```

После ввода наберите RUN и перед запуском еще раз осмотрите программу:

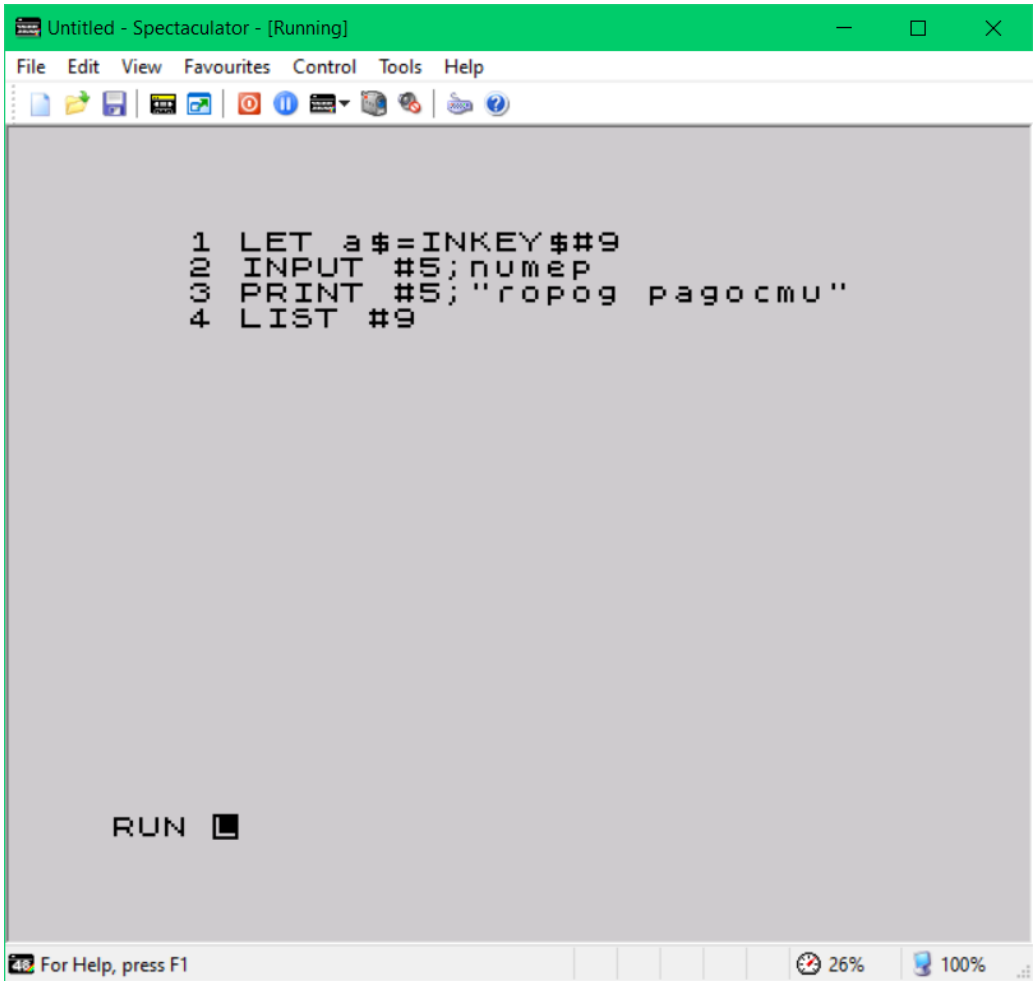


Рис. 506. BASIC программа для тестирования созданных блоков «X» и «E».

А теперь вводите ENTER:

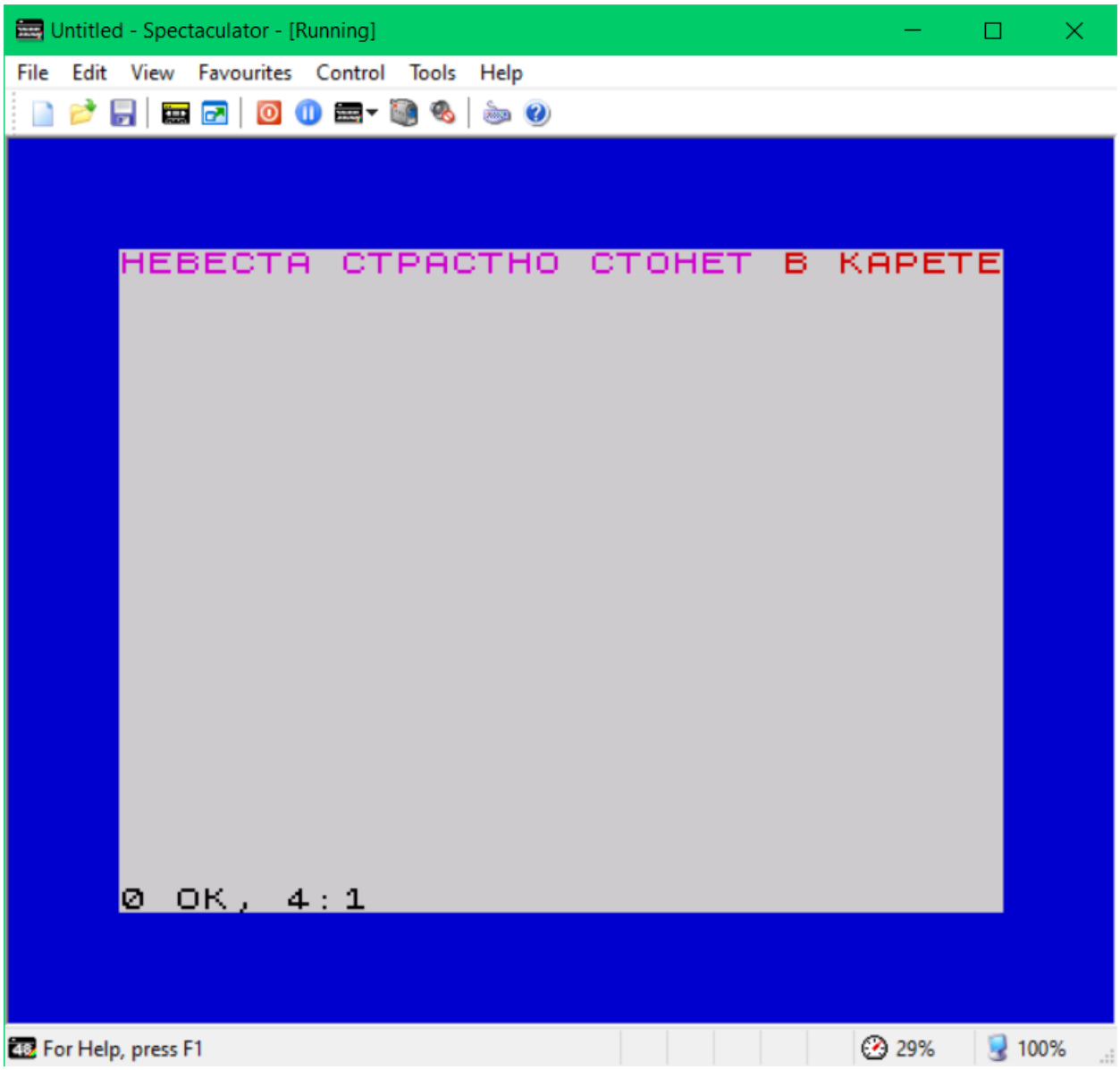


Рис. 507. Вывод группы программ по заданным адресам без USR помощи.

...Стрелка Васильевского острова. Биржевая площадь. Обычный летний день. У моста, в ожидании лохов, топчатся «Екатерина» с «Петром».

На стояночке напротив Ростральной колонны, среди «Мерсов» и «Бэх», стоит роскошная свадебная карета. У очередной свадьбы началась официальная предзастольная программа. Рослые мужчины в тёмно-малиновых пиджаках курят в тени деревьях за распределительным киоском Ленсвета. Мечтая поскорее оказаться за праздничным столом, они стоят с кислыми минами и безразлично смотрят на окружающий пейзаж. Девушки, в вызывающе коротких юбках, кучкуются у спуска к Неве и весело хихикают. Женщины постарше, в длинных роскошных платьях с глубоким декольте, чинно прогуливаются вокруг Ростральной колонны. Нанятые фотографы замерли в ожидании выхода жениха с невестой.

В этот момент карета начала тихонько покачиваться. Страстные стоны вперемежку с характерными шлепками, заставили оглянуться гостей и случайных прохожих возле широкого перехода...

Короткие раскаты гогота лишь подчеркнули пикантную ситуацию.

- Санёк... ну даёт... – усмехнулся один из друзей жениха.

А теперь по существу. Напечатав первые три слова, прозвучал BEEP 1,0, которого в тексте нет и в помине. Затем появилась концовка предложения. Самой последней по

LIST #9 выдалась синяя рамка, естественно на подмену вывода на экран текста программы. Команды полностью поменяли свои свойства, при этом BASIC система отработала корректно. Вместо LIST можно поставить LLIST, а вместо PRINT – LPRINT. После запуска результат от этого не поменяется.

По сути, это ещё один альтернативный способ запуска программ в машинных кодах без помощи USR. Итак, переход по адресу-1 в блоке CHANS будет осуществляться командами PRINT #, LPRINT #, LIST #, LLIST # и INPUT # без функции ввода переменной. Программа по адресу-2 запустится с помощью связок LET / PRINT / IF-THEN с INKEY$# или INPUT #;n , где n – любое имя переменной на конце, которое будет проигнорировано.

А что если попробовать сделать системный автостарт по ячейке 23734? Ведь после загрузки, но перед возвратом в BASIC выводится сообщение 0 OK. Но увы. По адресу 4909 вызывается очистка нижней части экрана (CALL 3438), которая попутно восстанавливает исходные адреса блока «K» и рушит все планы на открытие эксклюзивного способа автозагрузки. В общем, CHANS не даёт в этом плане ни малейшего ШАНСА.

Остался последний неохваченный момент, а именно:

> В "родном" "Спектруме-48" есть еще один стандартный канал - "R", но из Бейсика он не достижим и мы пока его отставим, вернемся к нему позже.

Рис. 508. Фрагмент из журнала ZX-Ревю №№5-6 за 1992 год.

Логика железная: если из BASIC недостижимо, значит это что-то внутреннее, таинственное и не для простых пользователей. А еще слово в «родном», как бы подтверждает теорию ~~заговора~~, что вся статья заточена под «SPECTRUM-128». Поэтому еще раз убеждаюсь, что рамках данной книги, информация бесполезна, хоть в теории, хоть на практике. Также история умалчивает, была ли информация по каналу «R» в более поздних выпусках. Бегло пролистав, годовую подписку за 1992-94 года, я ничего не нашел. Похоже, так к нему и не вернулись. Ладно, будет собственное исследование.

Итак, блок «R» отвечает за вывод строки для редактирования в нижнюю часть экрана… и за работу редкой BASIC команды STR$. Внезапно! В таблице STRMS он закреплен за парой «VIP-1» (№255). Как и большинство базовых блоков, он однобокий с «PRINT-перекосом». Поэтому рабочим адресом блока является первый. В адрес-2 засунута стандартная заглушка с сообщением об ошибке J Invalid I/O Device.

Несмотря на это есть у него отличия от других базовых блоков. Данные для «R» отсутствует в обеих таблицах ПЗУ для идентификации (5677 и той самой дефективной без маркера в 5910, из-за которой зависает CLOSE #, введенная на сухаря). Поэтому в программе CHAN OPEN, как и у любого самодельного кустарного блока, процесс настройки срежется на 5671 RET NC. Произойдет преждевременный выброс без задания цветов. Так что блок «R» можно назвать переходным видом между базовым и полностью самодельным. Его подсоединение к программе WAIT-KEY происходит с адреса 4047 в известной по первым главам программе ED-EDIT (4009).

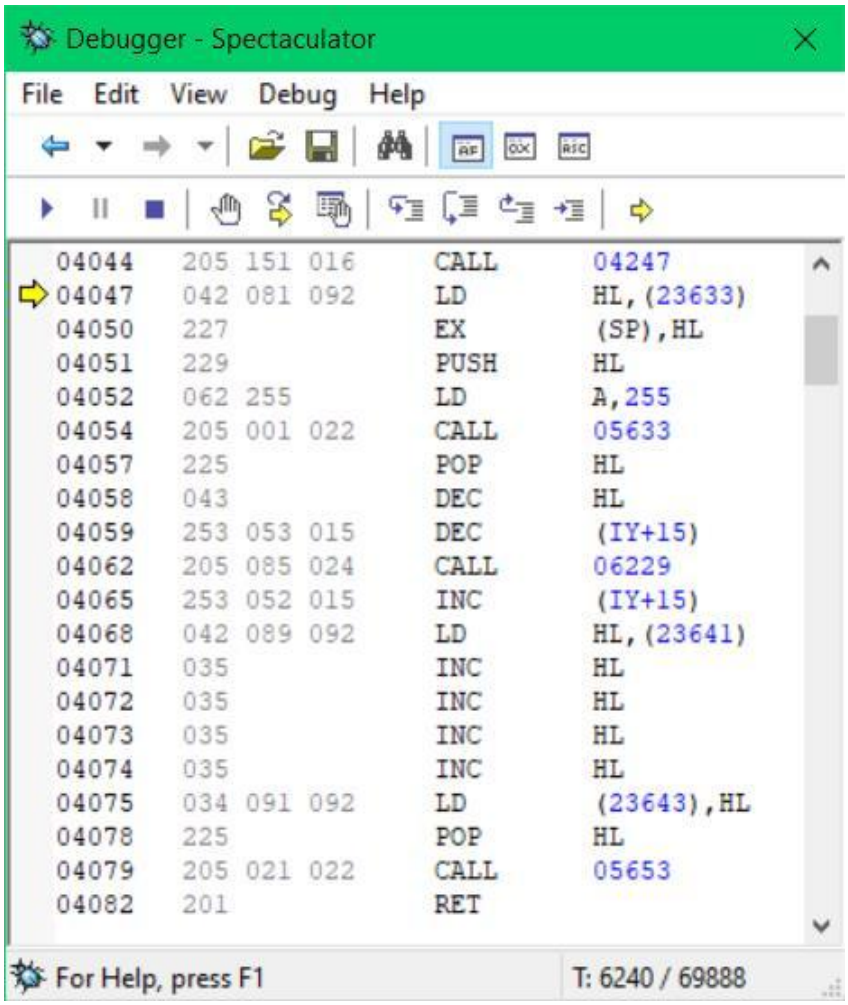


Рис. 509. Фрагмент программы ED-EDIT. Подготовка к подсоединению куска программы.

А следующая программа отвечает за команду `STR$`.

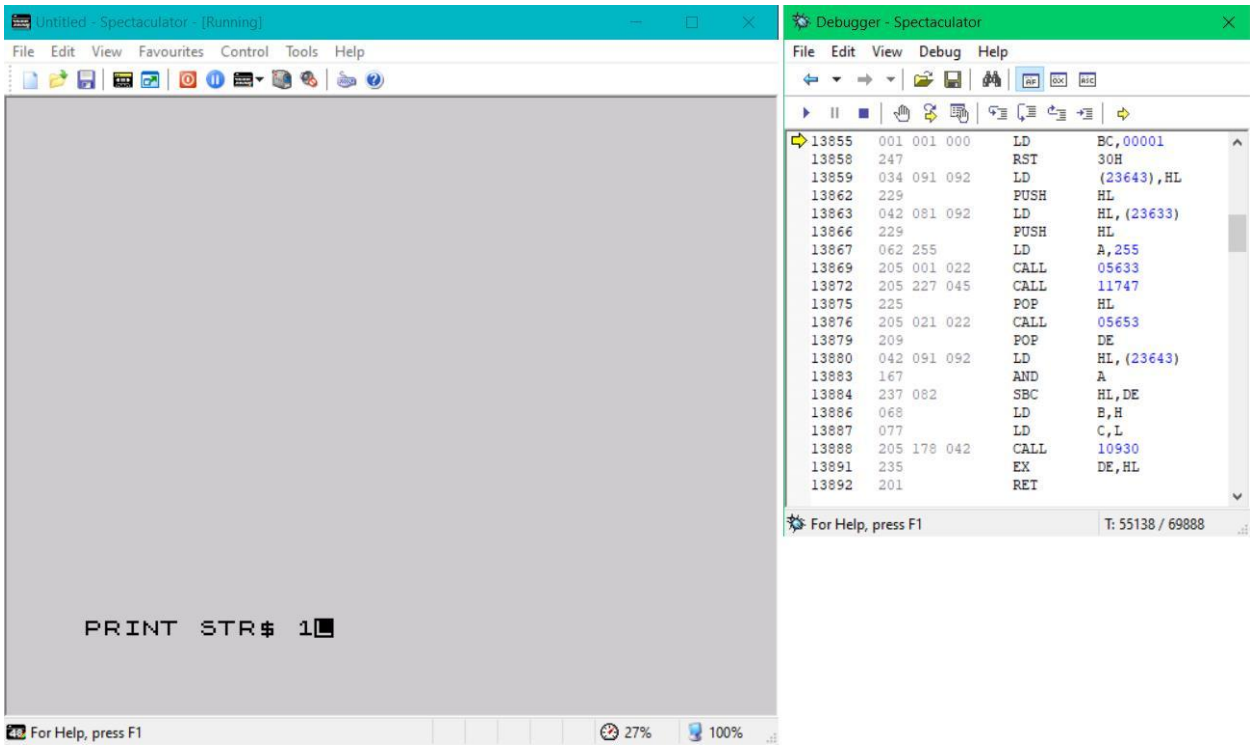


Рис. 510. Программа, формирующая команду STR$.

Схема работы программы из блока «R» ничем не отличается от вышеописанных. Все уже сказано заранее. А работу самой команды, предлагаю изучить по картинке или отладчику.

В заключение главы предлагаю пример, демонстрирующий работу блока «R». Пусть при попытке отредактировать строку будет появляться информационное сообщение, после чего произойдёт выход в BASIC с сообщением 0 OK.

```
Debugger
Dec
Go To 23744
23744 ← 30000
Go To 30000
30000 ← 62 254 205 1 22 17 73 117 1 33 0
30011 ← 205 60 32 49 84 255 237 115 61 92 241
30022 ← 195 3 19 22 11 2 16 1
30030 ← Сера РАКОМ Bucum Над ВАЗОМ
Trace
```

Следом натуральным способом введите BASIC строку:

```
1 PRINT STR$ 1
```

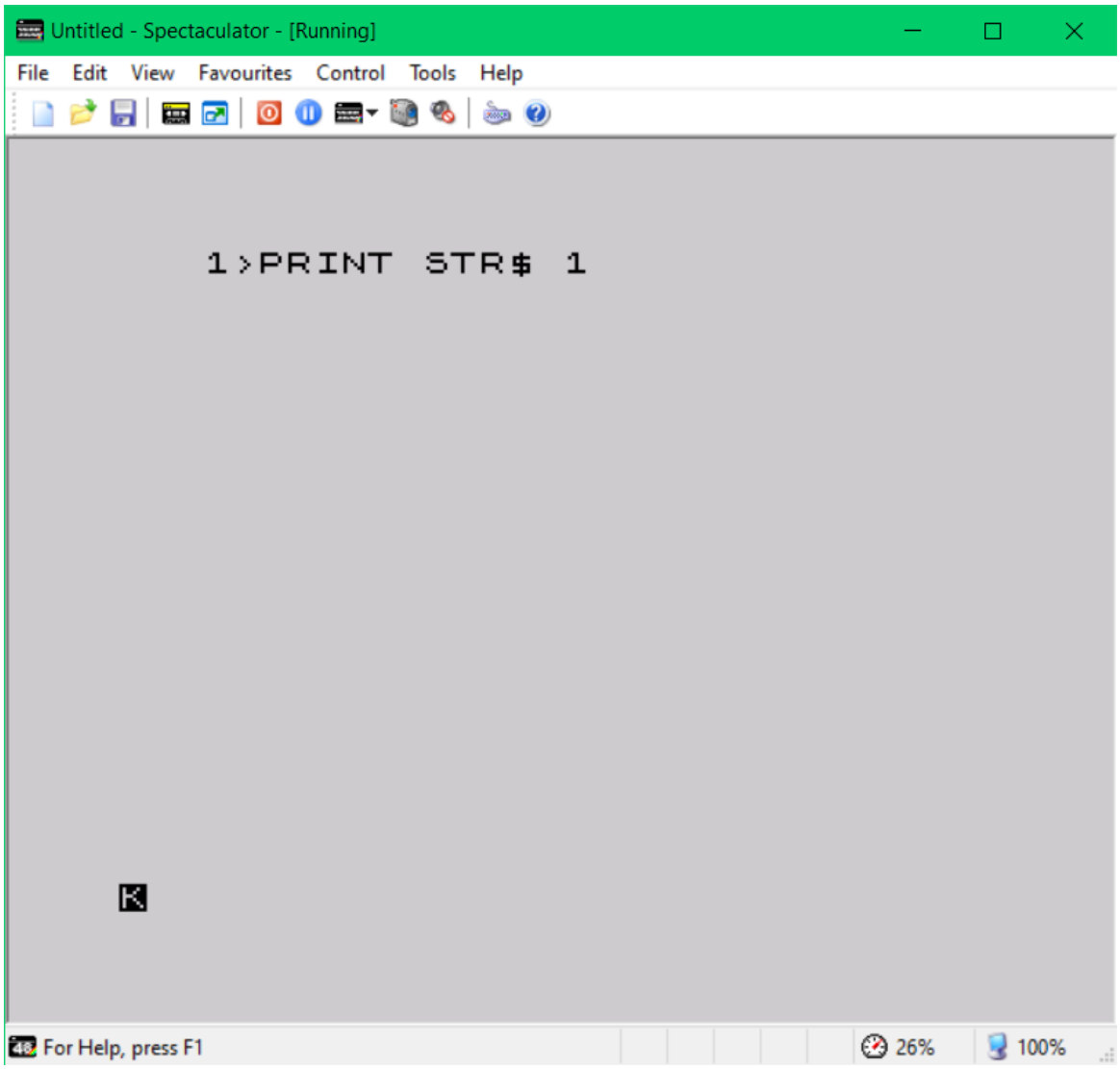


Рис. 511. BASIC строка с оператором STR$.

А теперь попробуйте взять строку на редактирование:

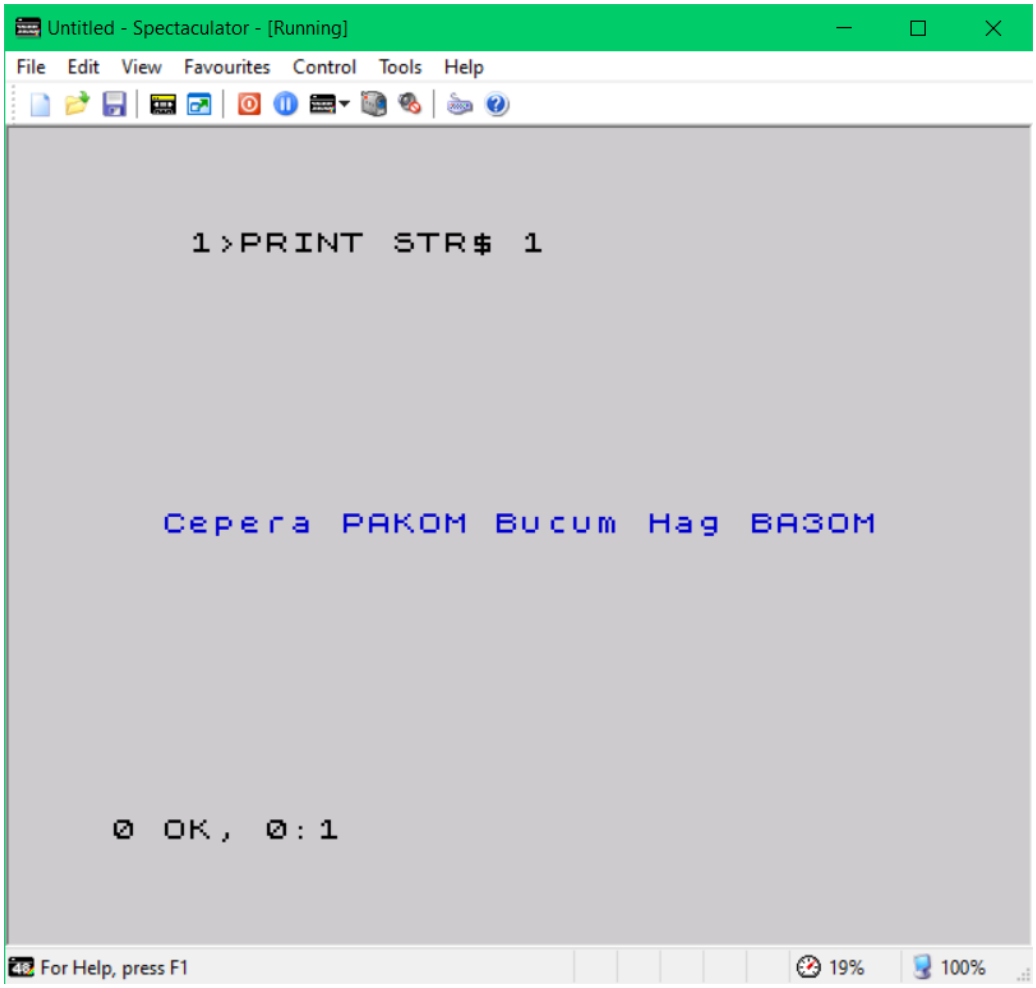
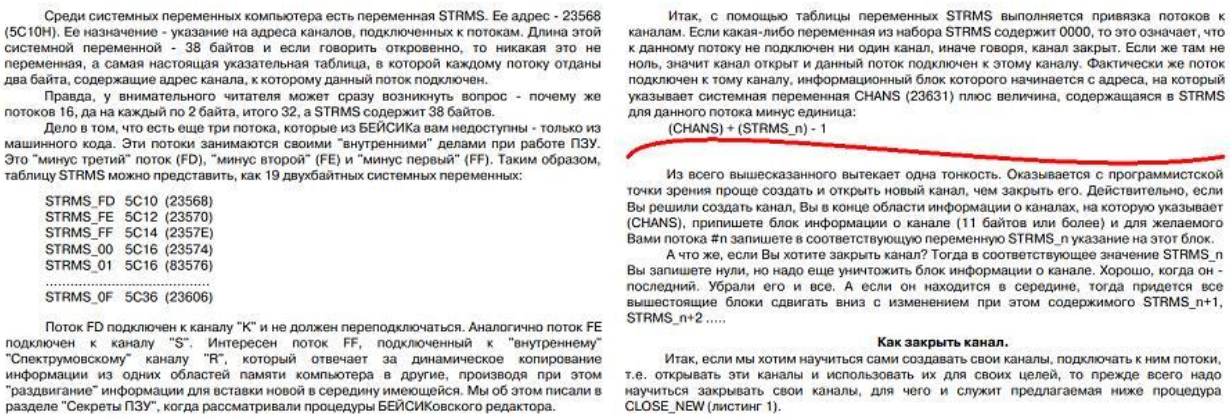


Рис. 512. Вывод собственного сообщения при попытке отредактировать строку.

Введите `RUN`, запустив строку. Эффект будет точно такой же, как от попытки редактирования строки. На экран выскочит всё то же напоминание о самостоятельном техническом осмотре и ремонте продукции отечественного автопрома. Это очередной метод, теперь уже с помощью команды `STR$` и редактирования запустить собственную программу. После опытов нажмите *Reset* и на этом пока всё.

В заключение главы, для отдыха, привожу ещё кусок ретро статьи из прошлого:



Среди системных переменных компьютера есть переменная STRMS. Ее адрес - 23568 (5C10H). Ее назначение - указание на адреса каналов, подключенных к потокам. Длина этой системной переменной - 38 байтов и если говорить откровенно, то никакая это не переменная, а самая настоящая указательная таблица, в которой каждому потоку отданы два байта, содержащие адрес канала, к которому данный поток подключен.

Правда, у внимательного читателя может сразу возникнуть вопрос - почему же потоков 16, да на каждый по 2 байта, итого 32, а STRMS содержит 38 байтов.

Дело в том, что есть еще три потока, которые из БЕЙСИКа вам недоступны - только из машинного кода. Эти потоки занимаются своими "внутренними" делами при работе ПЗУ. Это "минус третий" поток (FD), "минус второй" (FE) и "минус первый" (FF). Таким образом, таблицу STRMS можно представить, как 19 двухбайтных системных переменных:

```
STRMS_FD  5C10  (23568)
STRMS_FE  5C12  (23570)
STRMS_FF  5C14  (2357E)
STRMS_00  5C16  (23574)
STRMS_01  5C16  (83576)
.....
STRMS_0F  5C36  (23606)
```

Поток FD подключен к каналу "K" и не должен переподключаться. Аналогично поток FE подключен к каналу "S". Интересен поток FF, подключенный к "внутреннему" "Спектрумовскому" каналу "R", который отвечает за динамическое копирование информации из одних областей памяти компьютера в другие, производя при этом "раздвигание" информации для вставки новой в середину имеющейся. Мы об этом писали в разделе "Секреты ПЗУ", когда рассматривали процедуры БЕЙСИКовского редактора.

Итак, с помощью таблицы переменных STRMS выполняется привязка потоков к каналам. Если какая-либо переменная из набора STRMS содержит 0000, то это означает, что к данному потоку не подключен ни один канал, иначе говоря, канал закрыт. Если же там не ноль, значит канал открыт и данный поток подключен к этому каналу. Фактически же поток подключен к тому каналу, информационный блок которого начинается с адреса, на который указывает системная переменная CHANS (23631) плюс величина, содержащаяся в STRMS для данного потока минус единица:

(CHANS) + (STRMS_n) - 1

Из всего вышесказанного вытекает одна тонкость. Оказывается с программистской точки зрения проще создать и открыть новый канал, чем закрыть его. Действительно, если Вы решили создать канал, Вы в конце области информации о каналах, на которую указывает (CHANS), припишете блок информации о канале (11 байтов или более) и для желаемого Вами потока #n запишете в соответствующую переменную STRMS_n указание на этот блок.

А что же, если Вы хотите закрыть канал? Тогда в соответствующее значение STRMS_n Вы запишете нули, но надо еще уничтожить блок информации о канале. Хорошо, когда он - последний. Убрали его и все. А если он находится в середине, тогда придется все вышестоящие блоки сдвигать вниз с изменением при этом содержимого STRMS_n+1, STRMS_n+2 .....

Как закрыть канал.

Итак, если мы хотим научиться сами создавать свои каналы, подключать к ним потоки, т.е. открывать эти каналы и использовать их для своих целей, то прежде всего надо научиться закрывать свои каналы, для чего и служит предлагаемая ниже процедура CLOSE_NEW (листинг 1).

Рис. 513. Вырезка из журнала «ZX-Ревю» 11-12 за 1992 год.

А вот мне из вышесказанного с вырезки видится совсем другая тонкость. Зачем собственный блок делать 11 байтов, если нужно всего 5, причем достаточно любого печатного символа «на отвали», кроме «K», «S» или «P», которые распознаются по таблице ПЗУ. Дальше 5-го байта всё равно никто не смотрит. А стирать и закрывать

зачем? Ах да это же всё о «128K». Кому интересно, может почитать и ознакомиться, а я заканчиваю эту главу, с которой мучился почти месяц.

Глава 64
Устройство всратой печати

Краткое содержание: LPRINT, LLIST, #3, PR_CC (23680), 23296, переброска данных

В предыдущей главе, было выяснено, что `LPRINT` это не что иное, как `PRINT #3`. По задумке создателей, данные по этой команде отправляются как бы на принтер. Конечно, так было раньше, но какие нафиг принтеры для ZX-Spectrum в 2025 году, да ещё для программы под *Windows-10*?

Естественно, данная команда будет рассматриваться в другом ключе. Как показывает практика, перед отправкой на принтер, совершается несколько интересных действий, которые можно трансформировать в видимые эффекты.

Введена команда `LPRINT` с текстом и начинается её выполнение. Суть работы программы `CHAN-OPEN (5633)` с блоком «P» заключается в записи `CURCHL (23633)` адреса блока «P» и включения тумблера №1 в `FLAGS (23611)`. В знакомой программе `PR-STRING (8252)` начинается вывод текста в скобках. Вызывается программа `PO-FETCH (2819)`. Увидев включенный тумблер №1 на пульте `FLAGS (23611)`, вместо позиции вывода на экран, из пары ячеек `PR_CC (23680/81)` берется альтернативный адрес для вывода изображения и записывается в «HL». В стандартном случае, это буфер принтера 23296.

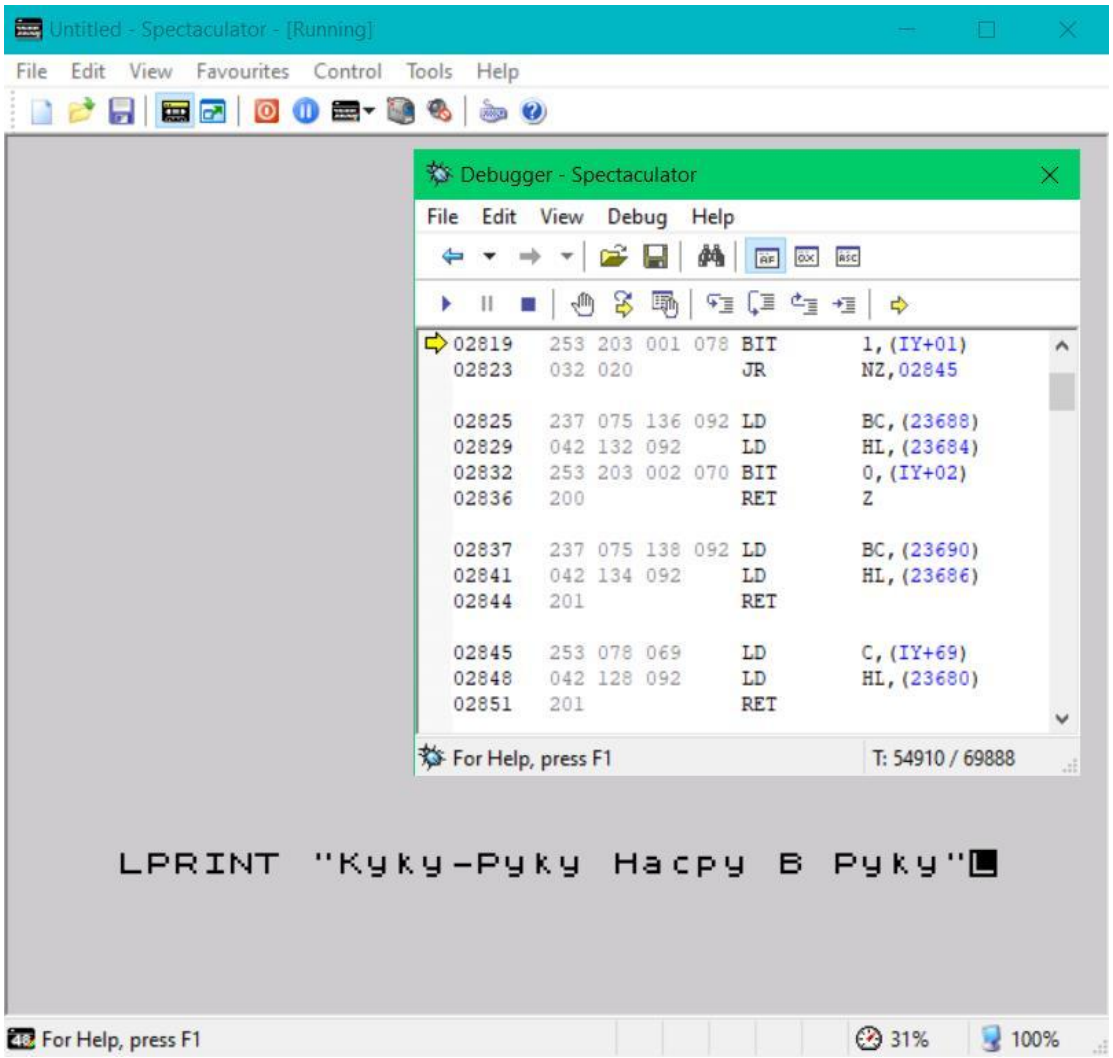


Рис. 514. Программа PO-FETCH. Выбор адреса для буфера принтера.

Далее в программе PO-ANY (2852) (*жарг. «Пони девочки Ани»*), начинается подготовка к построчному копированию в выбранную область экрана. В середине программы PR-AL-4 по адресу3004 настаёт торжественный момент записи в выбранную ячейку.

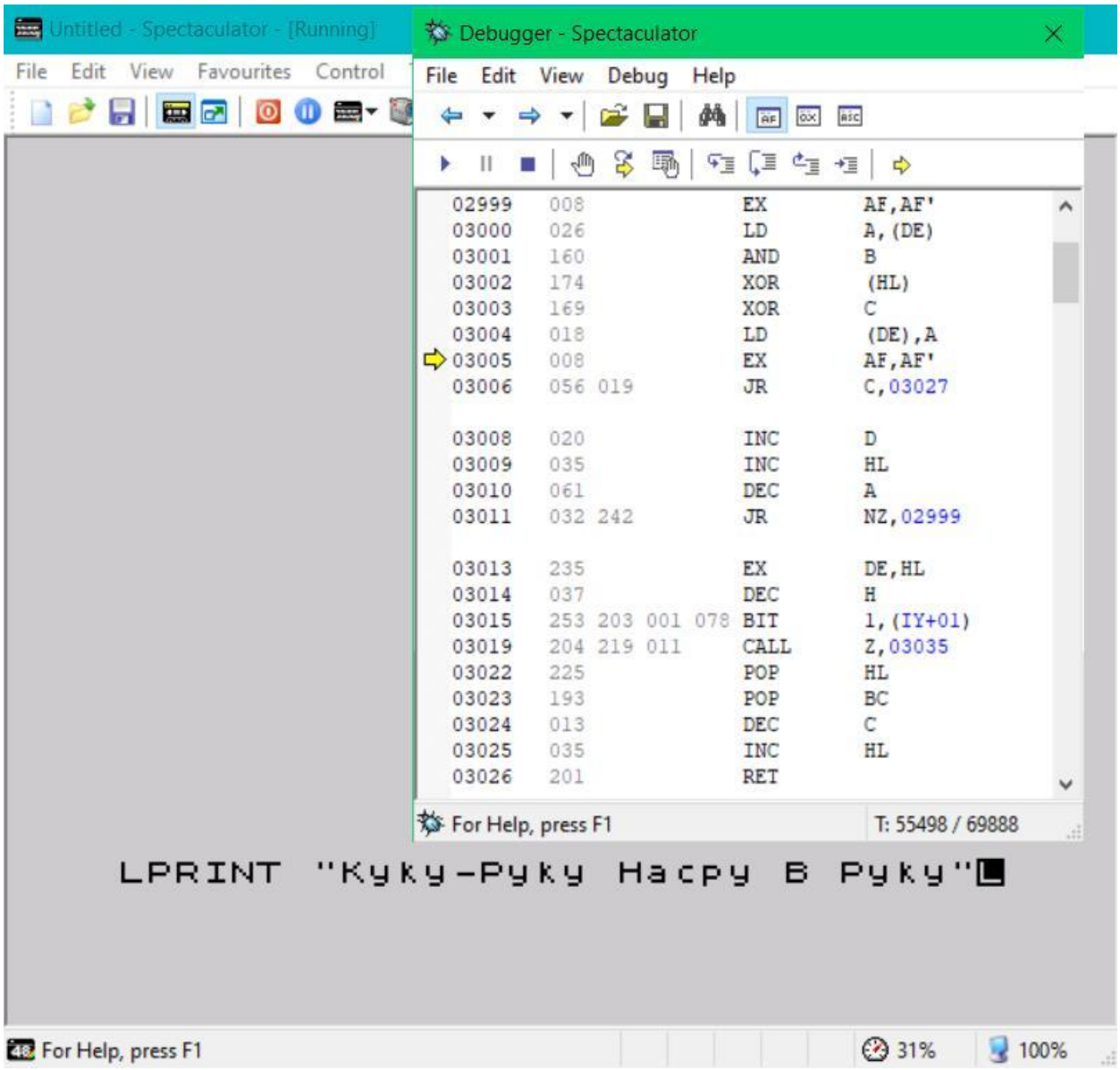


Рис. 515. Программа PR-AL-4. Запись байта символа в выбранный адрес.

После цикла печати происходит возврат из PR-STRING в точке 8255 по команде RET Z для окончания работы комплекса «PRINT». В PRINT-CR (8184) начинается печать ENTER для перевода строки. Уходя по RST 16, в WAIT-KEY снова срабатывает подсоединенная сменная насадка PRINT-OUT для работы с принтером (2548). Наступают последние вычисления, и Стрелочка наконец-то попадает на программу работы с принтером COPY-BUFF (3789).

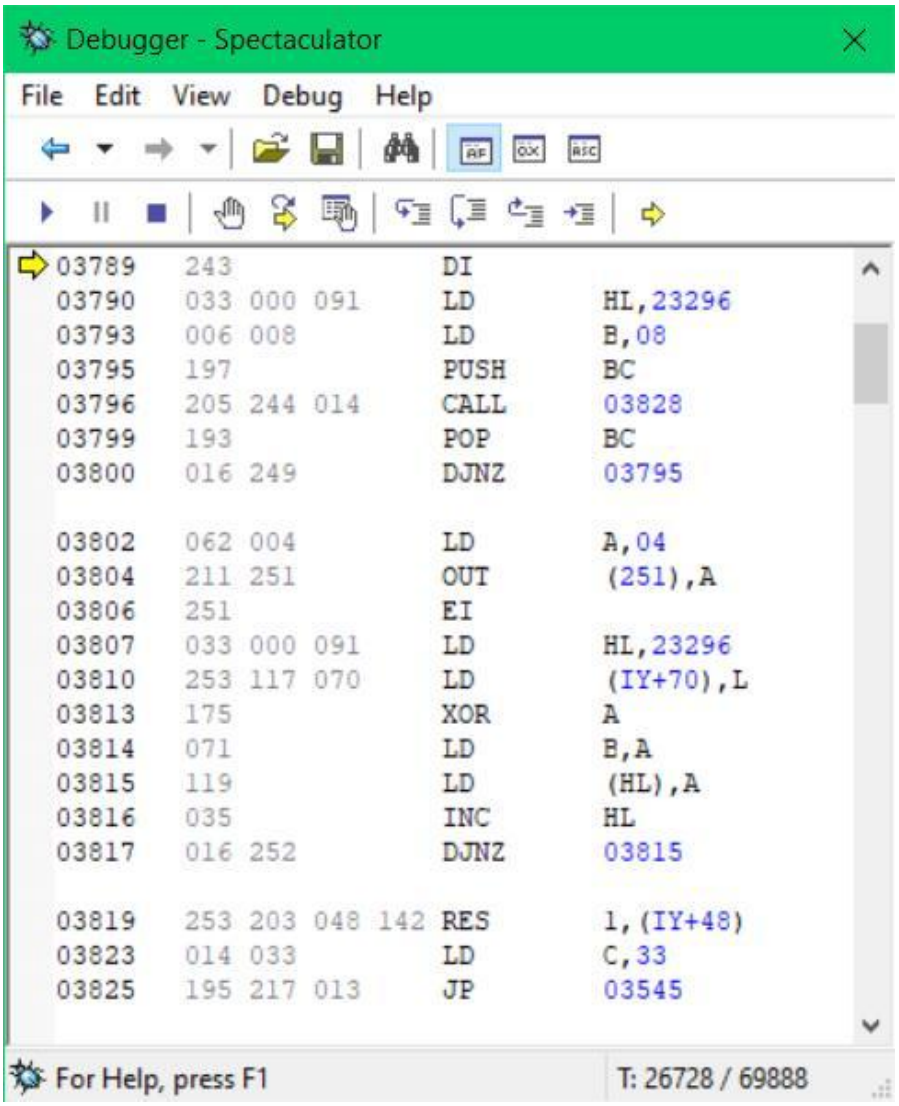


Рис. 516. Программа COPY-BUFF. Отправка данных на несуществующий принтер.

Тут строка, состоящая из 256 байт, с адреса 23296 отправляется на принтер.

Последовательно проглатывая командой `OUT (251), A (3804)`, они отправляются «господину Нихрена». После отправки на принтер, в `CLEAR-PRB (3807)` вся область 23296 очищается, а в младшую половинку переменной `PR-CC (23680)` устанавливается 0.

Подготовив адрес на заключительном этапе, Стрелочка попадает в `PO-ST-PR (2812)`, где переменной возвращается текущее значение 23296.

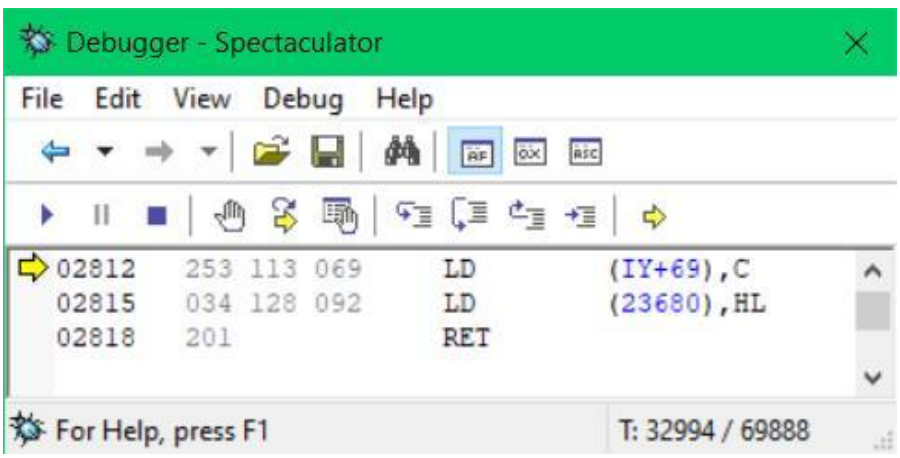


Рис. 517. Программа PO-ST-PR. Восстановление адреса буфера принтера.

Следом, после быстрой проверки (`8155 CALL 7150`) происходит постепенный выход на поверхность в комплекс программ `MAIN`.

Таким образом, с помощью функции вывода на принтер, есть возможность единоразово подкорректировать адрес переброски 256 байт данных командой `PRINT` без малиновых ● точек и вмешательства в ПЗУ. Для этого перед каждым вызовом `LPRINT` / `PRINT #3` нужно обновлять альтернативный адрес для вывода данных. Поскольку очистка после отдачи данных принтеру произведется по прямому адресу 23296 в любом случае, а не по переменной `PR_CC`, то всё записанное в альтернативное место, сохранится после отработки команд.

Предлагаю на примере ввода одиночного символа посмотреть, как он отобразится на экране. Для этого введите следующую программу:

```
Debugger
Dec
Go To 23610
23610 ← 255
23611 ← 2
23612 ← 32
23613 ← 65364
23680 ← 16384
65364 ← 4867
AF ← 22528
SP ← 65364
PC ← 5618
Trace
```

И на экране появилась…

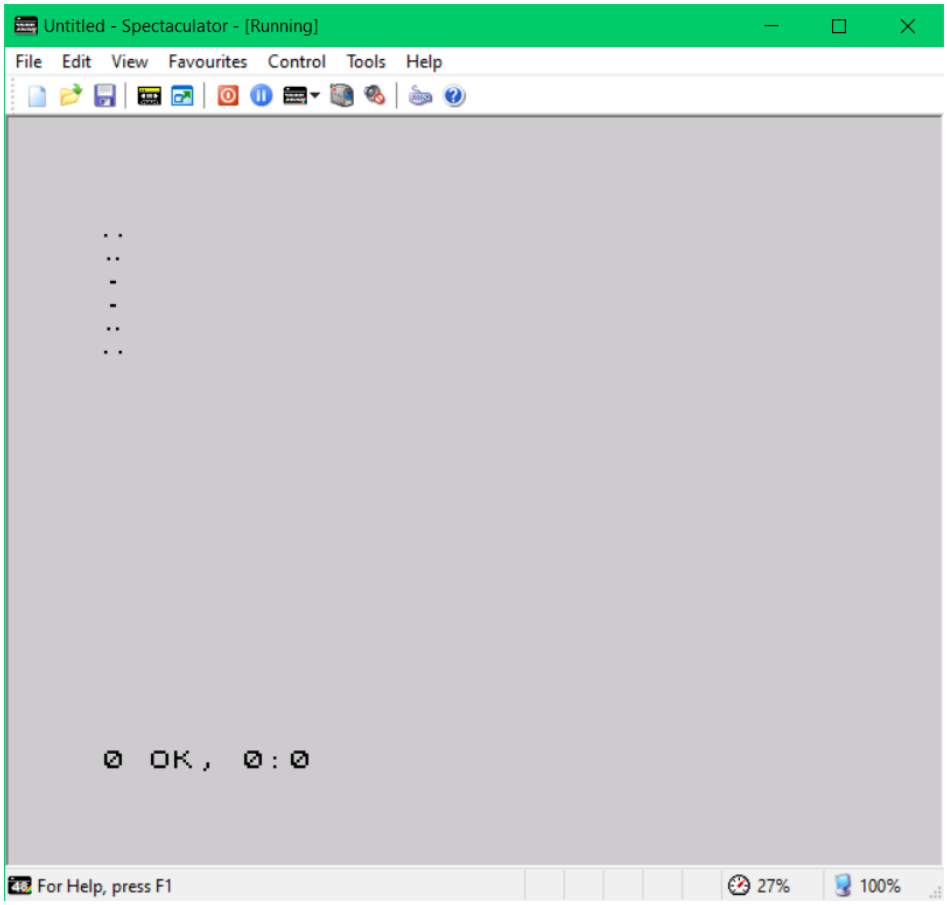


Рис. 518. Растягивание символов по вертикали после вывода командой LPRINT.

…раскромсанная буква «X». Из-за особенности структуры области экрана, полосочки, из которых состоит буква, разлепились. Между ними стало расстояние по 8 пикселей.

Для чего же это свойство использовали в некоторых программах прошлого? Для создания декоративного шрифта. В машинных кодах, это будет слишком банально, а вот на BASIC такая программа выглядит оригинально.

Скажу честно, идея была не моя и попалась в каком-то из номеров журнала «*ZX-Ревю*». Я лишь доработал.

Натуральным способом введите следующую BASIC программу;

```
1 FOR a=64 TO 71
2 POKE 23681,a
3 LPRINT "  КРАН ЗАСТРЕВАЕТ В АРКЕ МОСТА"
4 NEXT a
```

А теперь также естественно запустите, нажав `RUN` и `ENTER`:

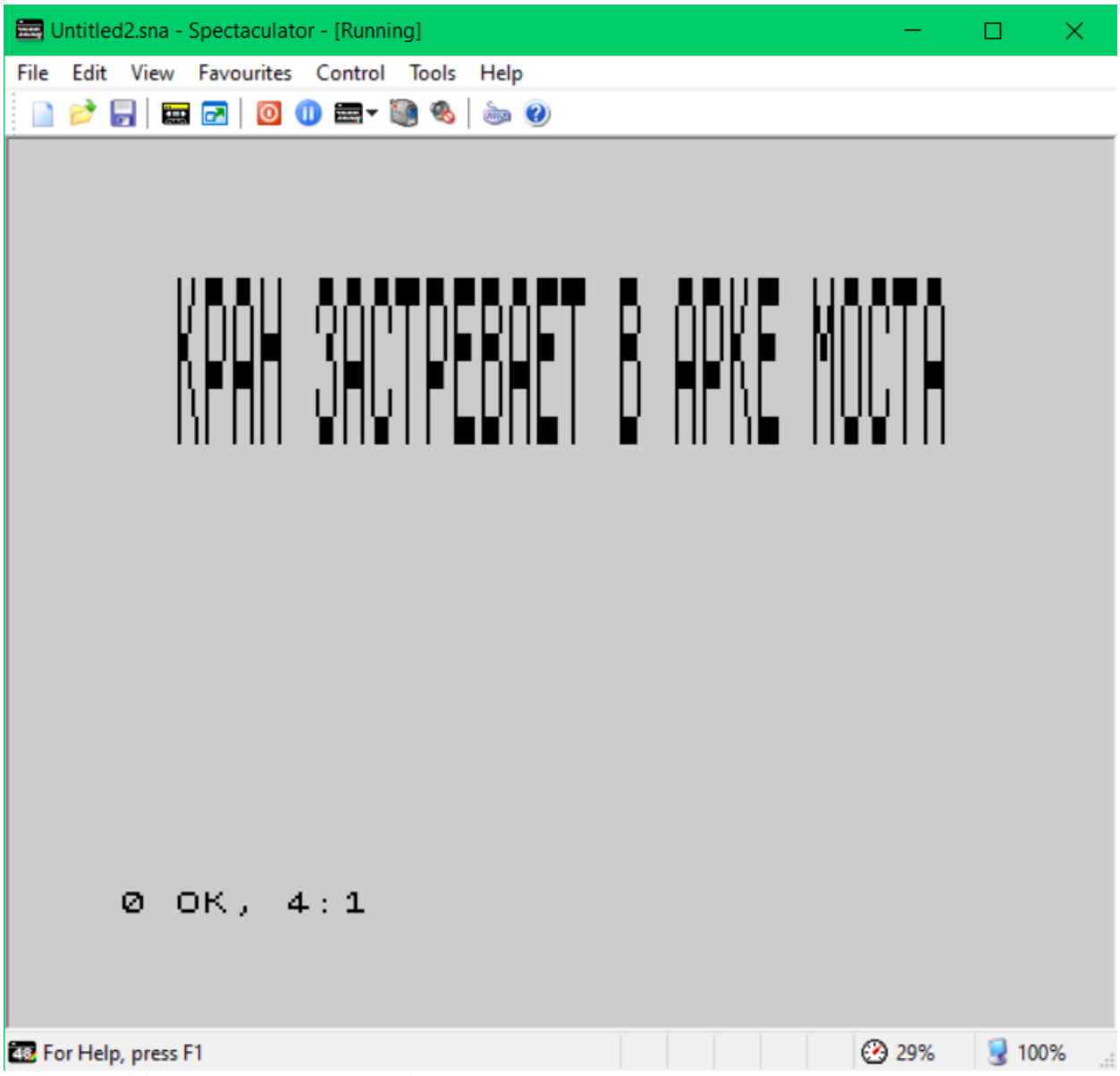
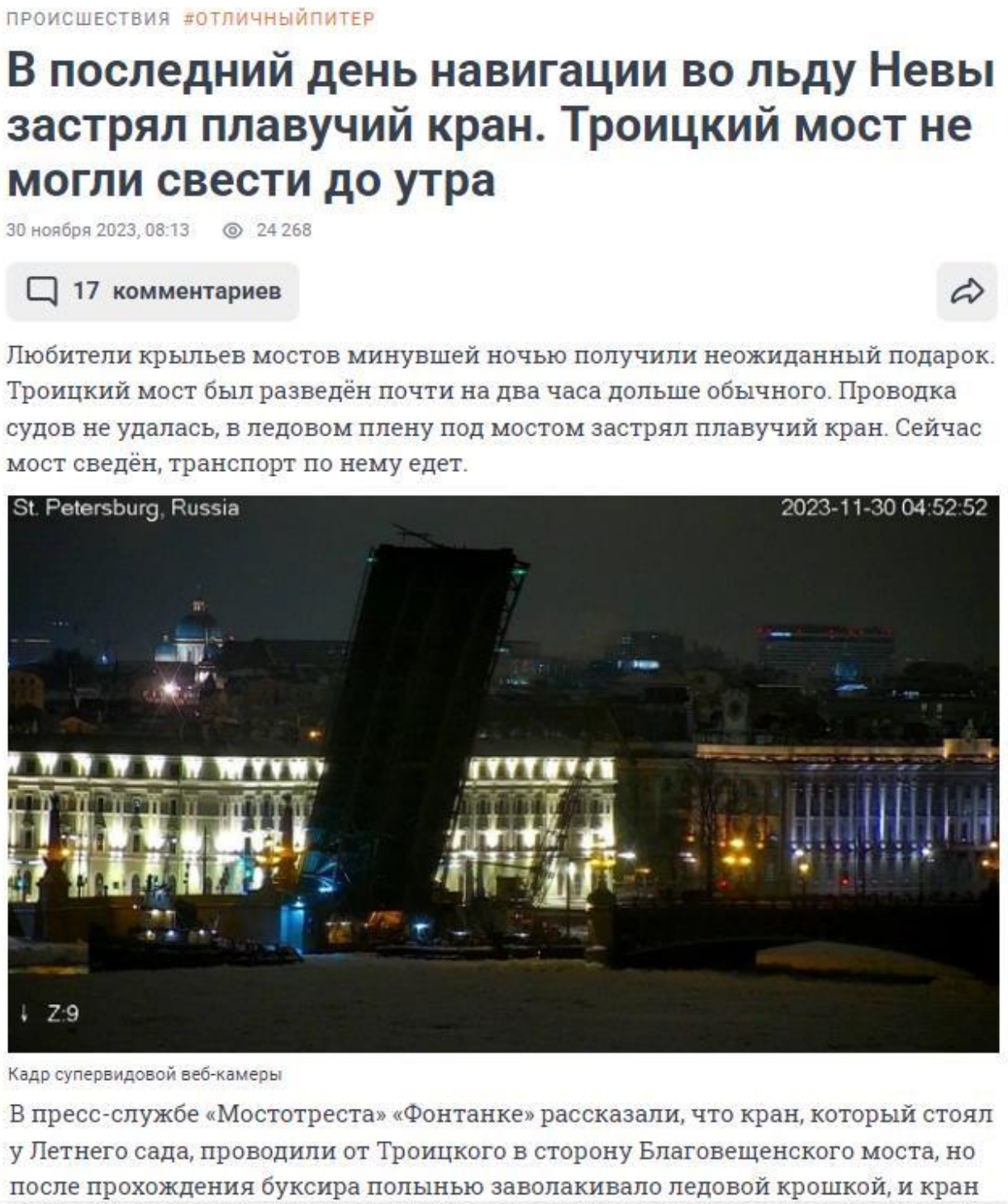


Рис. 519. Эффект декоративного шрифта с растяжением по вертикали.

Красотища и без особых усилий! Интересно, что до этого додумались гораздо позже, чем написали унылую главу в самоучителе по BASIC.

А тот самый кран, который застрял в арке моста, на следующее утро вытащили! «Фонтанка» не даст соврать.



ПРОИСШЕСТВИЯ #ОТЛИЧНЫЙПИТЕР

В последний день навигации во льду Невы застрял плавучий кран. Троицкий мост не могли свести до утра

30 ноября 2023, 08:13 24 268

17 комментариев

Любители крыльев мостов минувшей ночью получили неожиданный подарок. Троицкий мост был разведён почти на два часа дольше обычного. Проводка судов не удалась, в ледовом плену под мостом застрял плавучий кран. Сейчас мост сведён, транспорт по нему едет.

Кадр супервидовой веб-камеры

В пресс-службе «Мостотреста» «Фонтанке» рассказали, что кран, который стоял у Летнего сада, проводили от Троицкого в сторону Благовещенского моста, но после прохождения буксира полынью заволакивало ледовой крошкой, и кран

Рис. 520. Фотография с новостного сайта «Фонтанка».

Поскольку команды `LLIST` / `LIST #3`, в своём алгоритме также используют `RST 10`, чуда не ждите. По команде `POKE 23681,64: LLIST`, как на примерах выше, в выбранный адрес скопируется лишь первая строчка. По тому же принципу программой команды `PRINT (6268 RST 10)` она напечатается в заданный адрес, взятый из пары ячеек `PR_CC (23680)`, а потом идет на программу принтера, восстанавливая переменную до базового значения. Все последующие строчки продолжат сканироваться в привычную область 23296.

Для демонстрации вышесказанного, наберите и запустите натуральным способом следующую BASIC программу:

```
1 REM На погосте у креста
2 REM грустит Костик
3 FOR a=64 TO 79
4 POKE 23681,a: LLIST 2
5 NEXT a
```

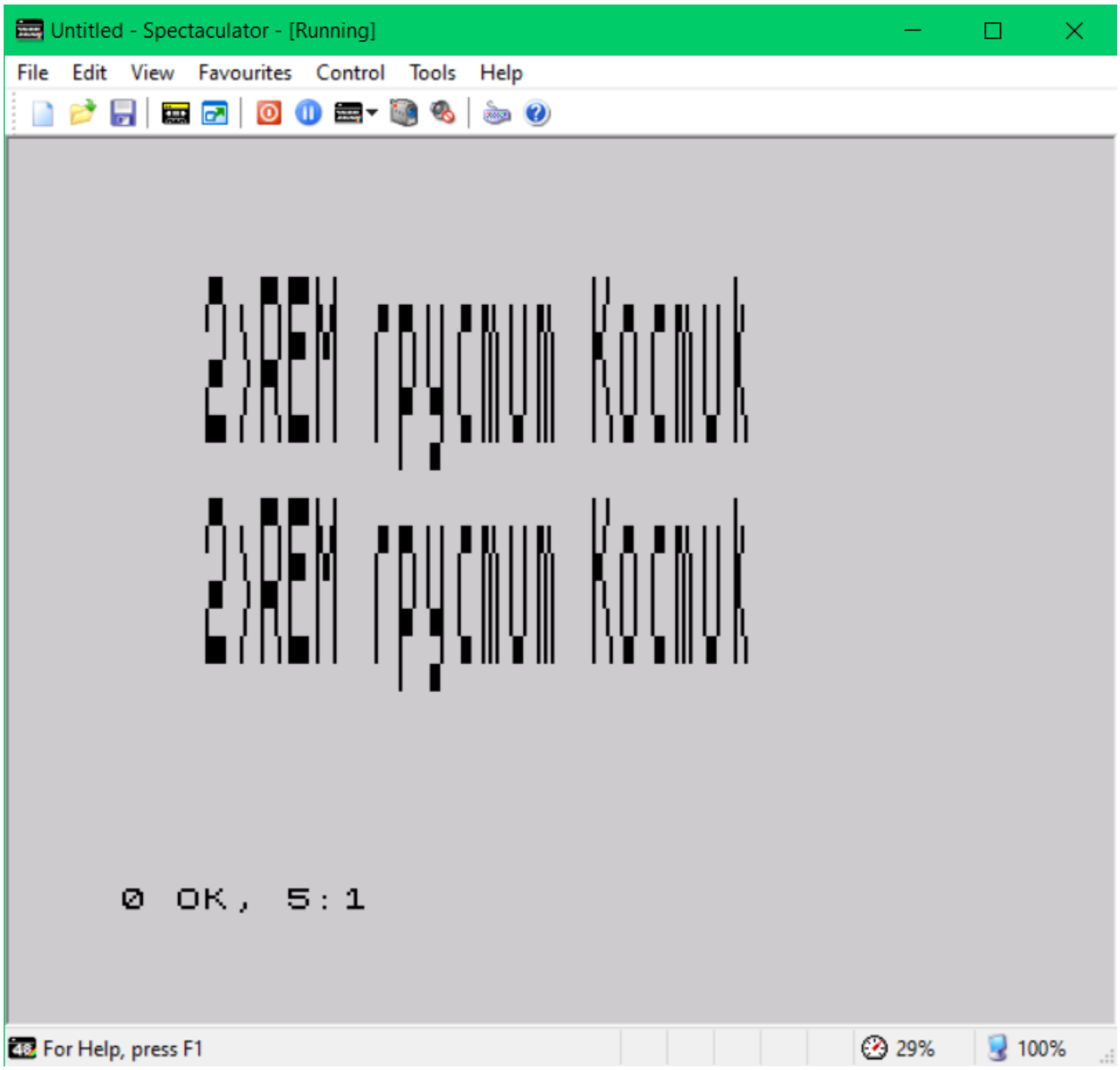


Рис. 521. Вывод первой заданной строчки по команде LLIST и игнорирование остальных.

Эффект аналогичный, только сейчас увеличенным шрифтом вывелась вторая строка, а дальше пошел повтор. Третья строка так и не появилась.

Из принтерного комплекта осталась неупомянутой команда `COPY`. Её программа с одноимённым названием, находится по адресу 3756:

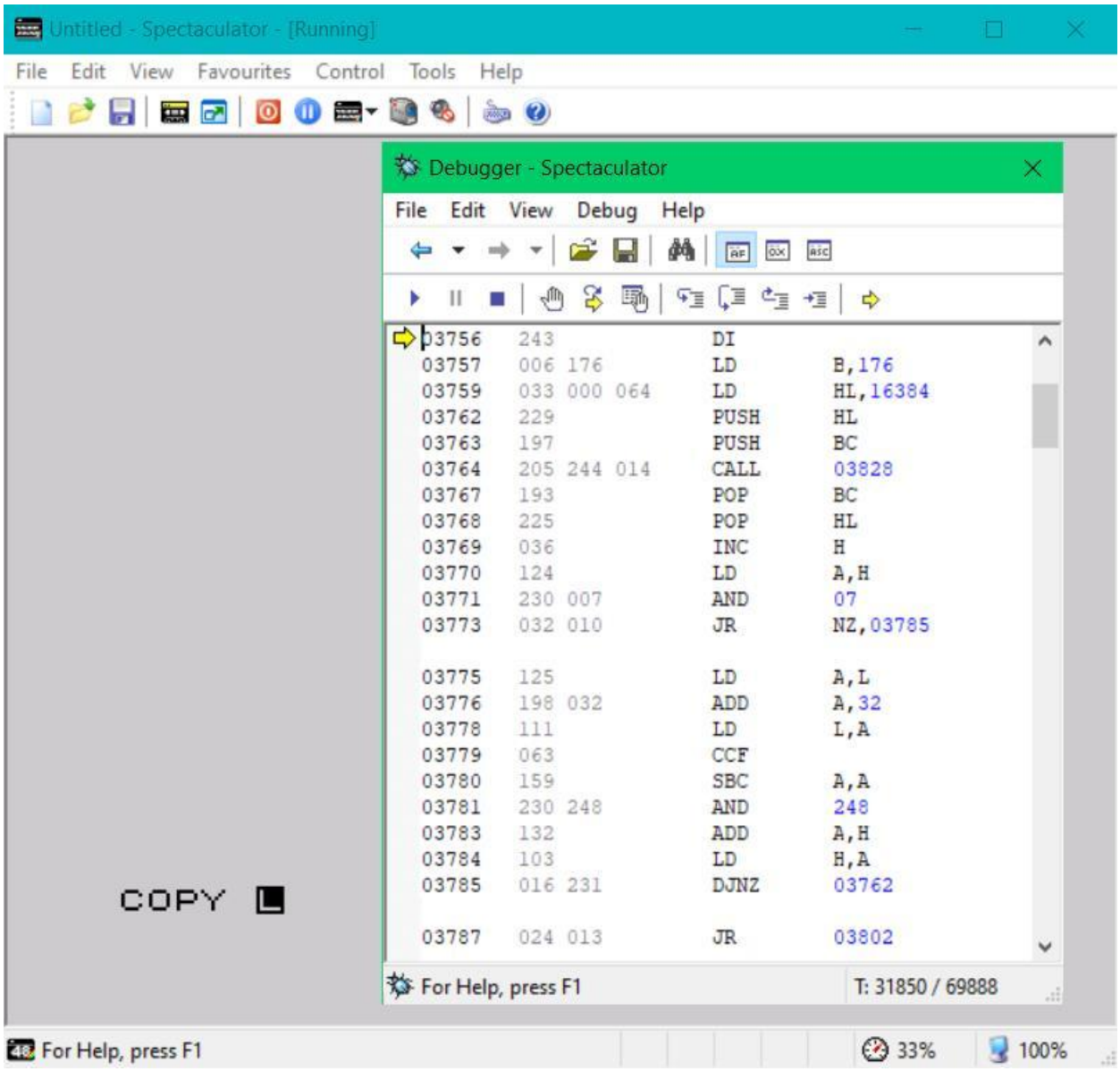


Рис. 522. Программа команды COPY.

Эта команда напрямую перебрасывает содержимое экрана на несуществующий принтер без обращения к ячейкам `PR_CC (23680/81)`. Мало того, если перед запуском `COPY` изменить содержимое этих ячеек, то после отработки команды и вывода `0 OK` там так и останется измененное значение. Таким образом, команда `COPY`, в рамках данной книги полностью бесполезна.

Глава 65
Другое несуществующее периферийное оборудование

Краткое содержание: CAT, ERASE, FORMAT, MOVE, разочарование

Введите следующую вычурную BASIC программу:

```
1 CAT
2 ERASE "ВСЕгеш"
3 FORMAT "Несум"
4 MOVE "пасму","гош"
```

Командой `RUN` можно позапускать программу с любой строки. Ответом на такие фантастические просьбы станет закономерное сообщение об ошибке:

Рис. 523. Ошибка при попытке выполнить CAT, ERASE, FORMAT или MOVE.

- Я вчера с такой тёлочкой познакомился, прикинь
- Круто, как её зовут?
- Лена, да ты её знаешь
- Какая Лена?
- С Эропроекта, ну ЛЕНАЭРОПРОЕКТ которая!
- НИТУ МИСИС ты выбрал, ох не ту…

Так вот, для чего и куда ВСЕГЕИ несут пасту ГОИ история также умалчивает.

Ну а теперь серьёзно. На самом деле, при запуске любой из этих команд, Стрелочка попадает на один и тот же адрес 6035. А там вместо таинственных алгоритмов, самый обычный переход `JR 5925`.

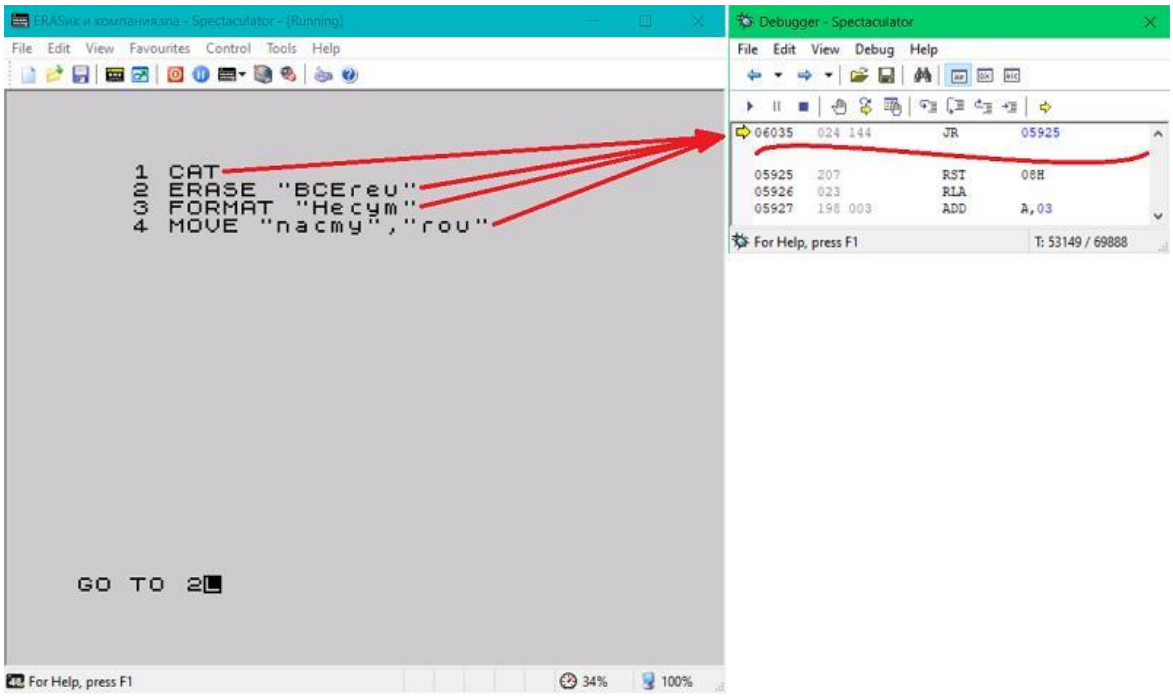


Рис. 524. Формальная программа действий при запуске команд CAT, ERASE, FORMAT, MOVE.

В официальной литературе эта точка так и называется `CAT-ETC`. Вот так открытым текстом, без пара и воза:

```
КОМАНДНАЯ ПРОЦЕДУРА 'CAT, ERASE, FORMAT & MOVE' ('Каталог', 'Уничтожить',
'Форматировать' и 'Переместить')
В стандартной системе SPECTRUM использование этих команд приводит к сообщению O -
неправильный поток.

1793 CAT-ETC.     JR    1725,REPORT-O       Выдача этого сообщения.
```

Рис. 525. Вырезка из книги «Полное описание ПЗУ ZX-Spectrum».

Так что не ERASE'ньки интересного тут нету. По адресу 5925 стоит банальная заглушка RST 8 с сообщением об ошибке №23. Да это команды-пустышки без обслуживающих программ, но в таблице параметров, с адреса 6918, для них на перспективу прописан синтаксис,

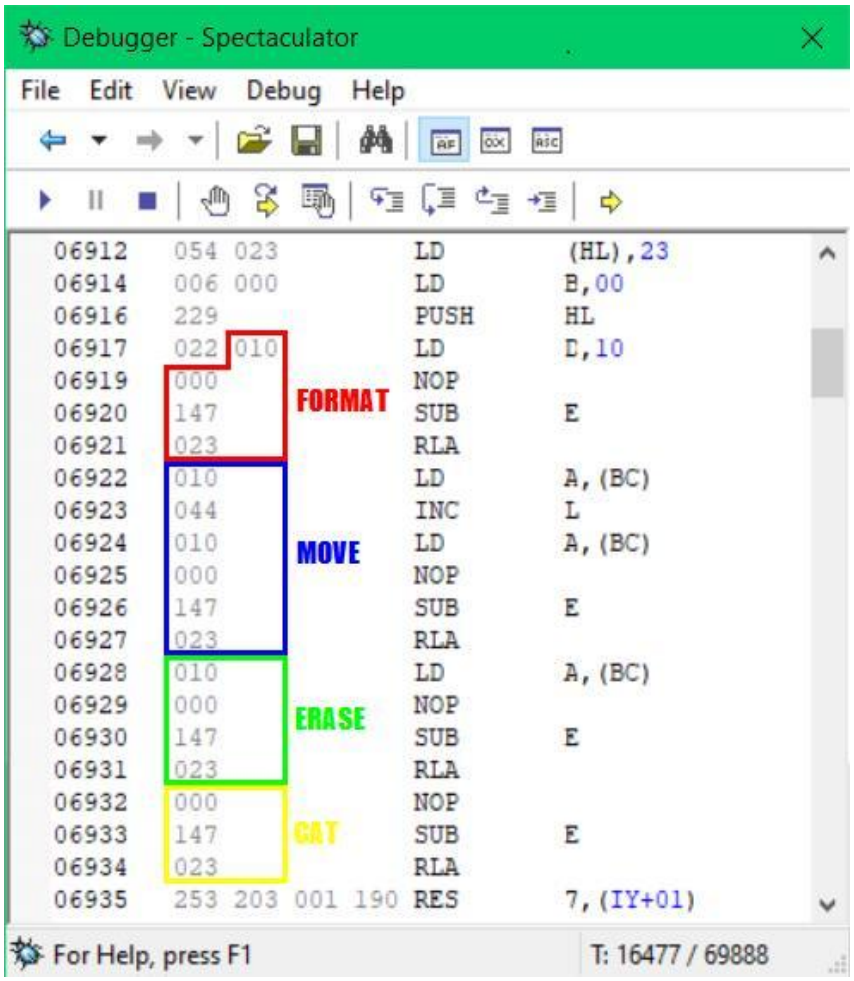


Рис. 526. Синтаксис команд FORMAT, MOVE, ERASE и CAT.

Наглядно видно, что для команд FORMAT и ERASE запрограммирован лишь набор текста в кавычках, подобно PRINT (10). Для MOVE через запятую надо вводить две строчки текста в кавычках (10, 44, 10). Ну а CAT и вовсе запускается без каких-либо параметров.

В заключение главы традиционная ретро-перспектива с вырезкой из старой литературы.

Глава 22

Другое периферийное оборудование

Имеется ряд других устройств, которые могут быть подключены к компьютеру ZX SPECTRUM. ZX MICRODRIVE-высокоскоростное устройство памяти, может быть использовано вместо кассетного магнитофона. Однако оно не может управляться командами SaVE, VARYFY, LOAD, MERGE, а лишь командами PRINT, LIST, INPUT и INKEY$. При помощи этого устройства можно организовать сеть из нескольких компьютеров ZX SPECTRUM. стандартным интерфейсом для ZX SPECTRUM является RS-232, посредством которого подключаются: клавиатура, принтер и любые другие устройства, отвечающие стандартам этого интерфейса. При работе с такими устройствами могут использоваться имеющиеся на клавиатуре дополнительные ключевые слова: OPEN$, CLOSE$, MOVE, ERASE, CAT, FORMAT.

Рис. 527. Глава 22 из самоучителя по BASIC.

Вот так. А мужики то ие знают, что OPEN # и CLOSE # без Microdrive не работают.

Глава 66
Точный NMI-выстрел в призовом тире

Краткое содержание: NMI-выстрел, NMIADD (23728), ошибка ПЗУ

Эта тема так же, как «Каналы с потоками», обходилась стороной на протяжении пары десятков лет. Вернее, не было стимула вникать, потому что непонятно как всё это использовать на практике. Пришло время закрывать последние долги юности по ZX-Spectrum. Во второй половине августа 2025 года, внезапно нашлась подходящая аналогия из реальной жизни, после чего тема была детально изучена.

Вначале историческая справка.

Немаскируемые прерывания

Немаскируемое прерывание вызывается при поступлении сигнала на вход NMI процессора. При этом выполнение основной программы приостанавливается, и управление передается на ячейку с адресом 102 (#66) (выполняется команда CALL 102).

В ZX Spectrum немаскируемое прерывание не задействовано, но оно может вызываться некоторыми внешними устройствами. При этом вместо стандартной «прошивки» подключается ПЗУ вызвавшего прерывание устройства, в котором записана программа обработки прерывания. Этот принцип используется, например, в дисковой системе Beta Disk фирмы Technology Research и интерфейсе Multiface One фирмы Romantic Robot.

При входе в немаскируемое прерывание сбрасывается триггер IFF, и, следовательно, запрещается вызов маскируемых прерываний.

Возврат из немаскируемого прерывания и восстановление состояния регистра IFF происходит по команде процессора RETN.

Рис. 528. Фрагмент книги «ZX-Spectrum для пользователей и программистов».

Подаются некие сигналы на микросхему, триггеры, прерывания, внешние устройства... Все по существу, но с первого взгляда это уже даже не высокоуровневое программирование, а вообще тема для радиолюбителей. Ведь вы не можете с клавиатуры подать спецсигналы на микросхему. Читая эту главу в конце 2000-х, мне представлялся конец 1980-х годов, когда люди сидели с паяльниками и вечерами в свободное время собирали себе компьютеры.

Теперь закономерные вопросы: что это такое простыми словами, и каким боком оно нужно в данной книге?

Представьте себе, призовой тир, где за меткий выстрел дают подарки. Так вот, у вас в руках «NMI-ружье». Крепко держа, чтобы не отнесло отдачей, вы хорошо прицеливаетесь и нажимаете на курок, чтобы смачно жахнуть одиночным в самую цель. Но есть нюанс. Оружие общественное, подраздолбанное с кривым дулом. Чтобы попасть в цель, нужно постараться.

0066-006F – RESET.

Процедура обработки немаскированного прерывания. Сразу скажем, что в фирменной ПЗУ здесь содержится ошибка.

При получении прерывания по линии INT, процессор обращается к адресу 0066 (это заложено в устройстве процессора), и здесь он должен найти адрес процедуры, которая будет это прерывание обрабатывать.

```
PUSH AF          ; Запомнили на стеке
PUSH HL          ; содержимое этих пар.
LD HL, (5CB0)    ; В регистр HL загружается содержимое системной переменной, находящейся по
      указанному адресу - 5C80.
```

Не ищите эту системную переменную в литературе. Поскольку данная процедура не работает из-за содержащейся в ней ошибки, то и про эту системную переменную нигде не пишут, а говорят, что адрес 5CB0 (23728) якобы не используется. Должна же эта переменная называться NMIADD и содержать адрес, по которому находится программа, обрабатывающая немаскируемое прерывание.

```
LD A,H           ; Проверка содержимого
OR L             ; HL на 0.
JR NZ, 0070      ; ОШИБКА! Здесь должно быть JR Z,0070. Должно быть так, что если в NMIADD
      стоит 0, то переход в 0070 и оттуда возврат, ничего не сделав. А если не 0, то переход
      по адресу из NMIADD.
JP (HL)          ;На самом же деле все происходит наоборот: если 0, то происходит переход по
      адресу 0, т.е. системный сброс компьютера, а если не 0, то возврат, ничего не сделав.
      0070-0073 - NORESET.
POP HL           ; Восстановление содержимого
POP AF           ; этих пар.
RETN             ; Возврат после обработки немаскированного прерывания.
```

Рис. 529. Описание программы NMI. Фрагмент журнала «ZX-Ревю» №1 от 1991 года.

В общем, в программе допущена ошибка и её нужно будет исправлять на ходу во время запуска. Теперь по второму пункту. Время реальных компьютеров ушло в историю. Нетрудно догадаться, что у Spectaculator.exe есть ручное управления всеми этими странными штуками.

Предлагаю перейти к нехитрому примеру. Суть в следующем. В произвольный адрес записывается программа вывода текста. Чтобы не сохранять все значения в конце будет привычная срезка SP с выходом в `MAIN-4`. В общем, эта часть стандартная. Далее нужно зарядить ячейки `NMIADD (23728/29)` адресом своей программы, установить малиновую ● точку по адресу 109 с ошибочной командой и выйти из отладчика. Все это тоже пройденный материал.

Из нового останется сделать выстрел из «NMI-ружья», чтобы Стрелочка прибежала на адрес 102 и попалась в ловушку на малиновой точке, Выставив Z ☑ галочку Z, просто продолжите выполнение программы.

В God Mode введите следующую программу:

Debugger

```
Dec
Go To 109
Add Breakpoint 109
Go To 23728
23728 ← 30000
30000 ← 253 54 2 32 17 77 117 1 33 0 205 60 32
30013 ← 49 84 255 237 115 61 92 241 253 54 0 255
30025 ← 251 195 3 19 19 1 16 3 22 11 3
30036 ← прохор пумаем мир и сормир
Trace
```

А теперь на основном окне программы нажмите на меню «*Control*» и найдите пункт «*Generate NMI (F5)*»:

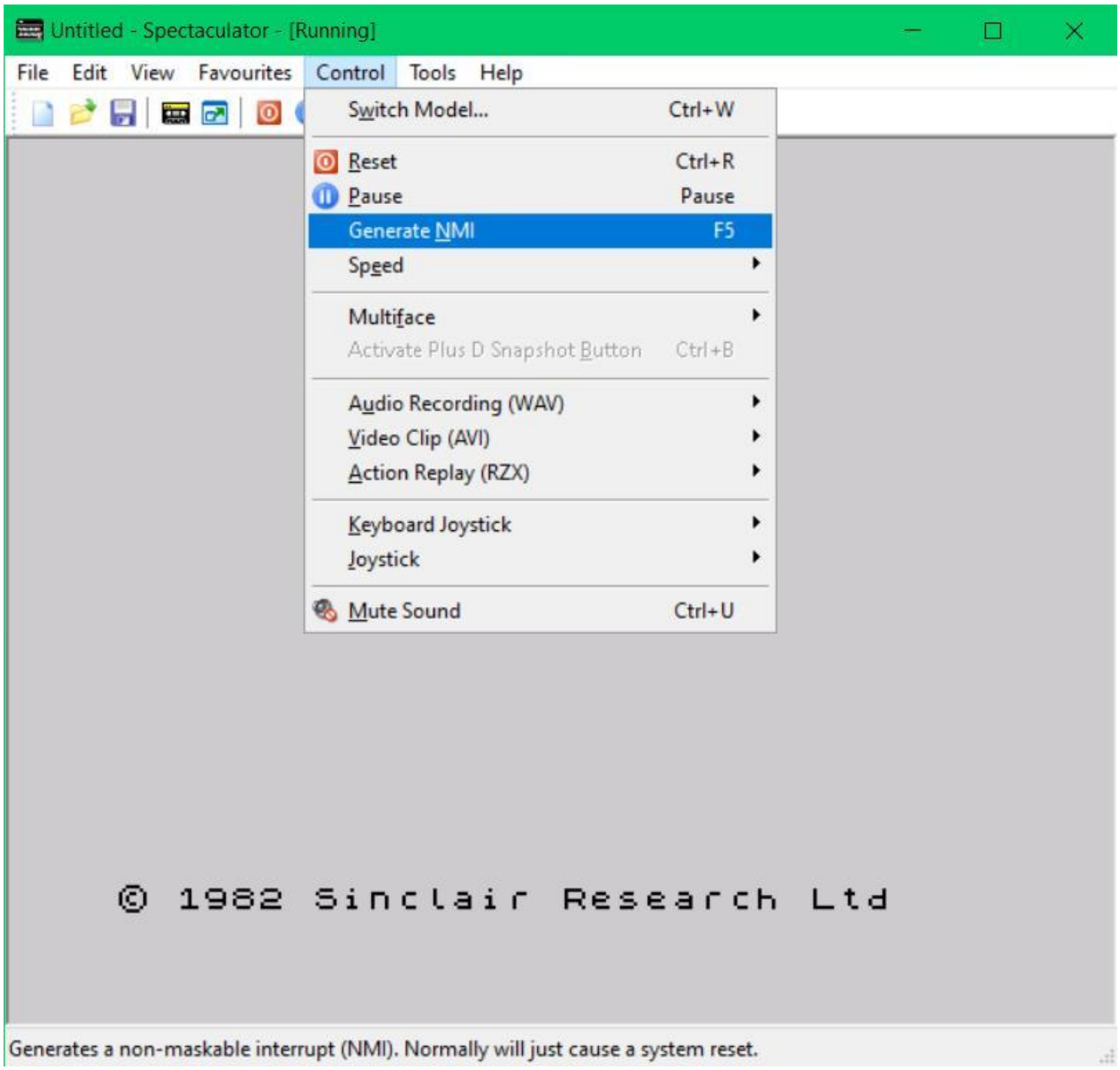


Рис. 530. Подготовка к выстрелу. Выбор пункта «Generate NMI».

После нажатия на подсвеченный пункт меню закроется, а на экран выдастся окошко. Программа поинтересуется, вы точно отдаёте себе отчёт в плане правомерности применения «NMI-оружия»:

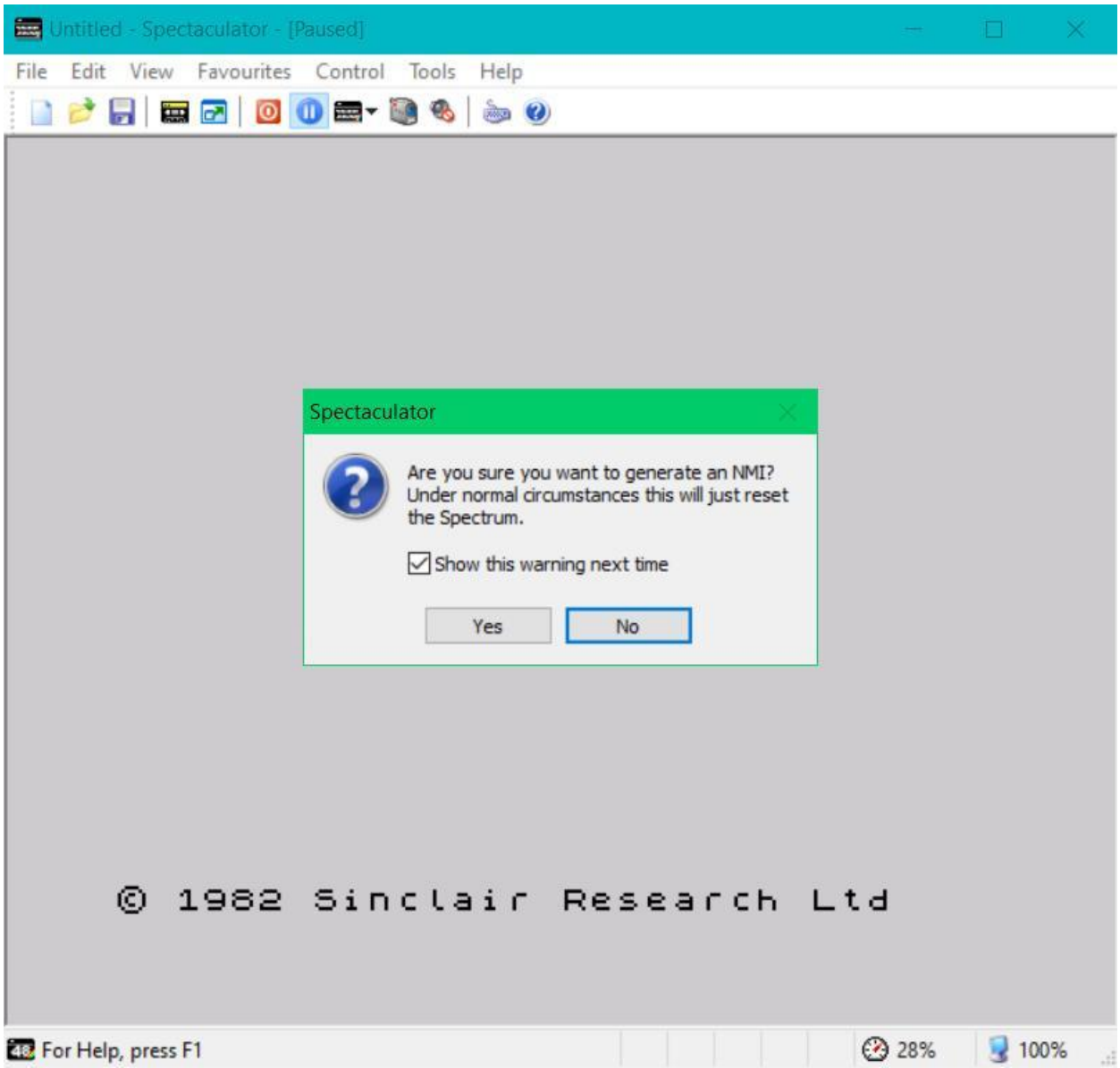


Рис. 531. Последнее предупреждение перед генерацией NMI-выстрела.

Если вы хотите избавиться от этого предупреждения в дальнейшем, то просто снимите галочку слева от фразы «*Show the warning next time*». Нажмите кнопку «*Yes*» и пуля полетела в цель. Стрелочка очутилась на заброшенной подпрограмме по адресу 102 и остановилась на малиновой точке в 109. Поставьте Z галочку Z или к «AF» добавьте 64, чтобы не дать Стрелочке совершить роковую ошибку.

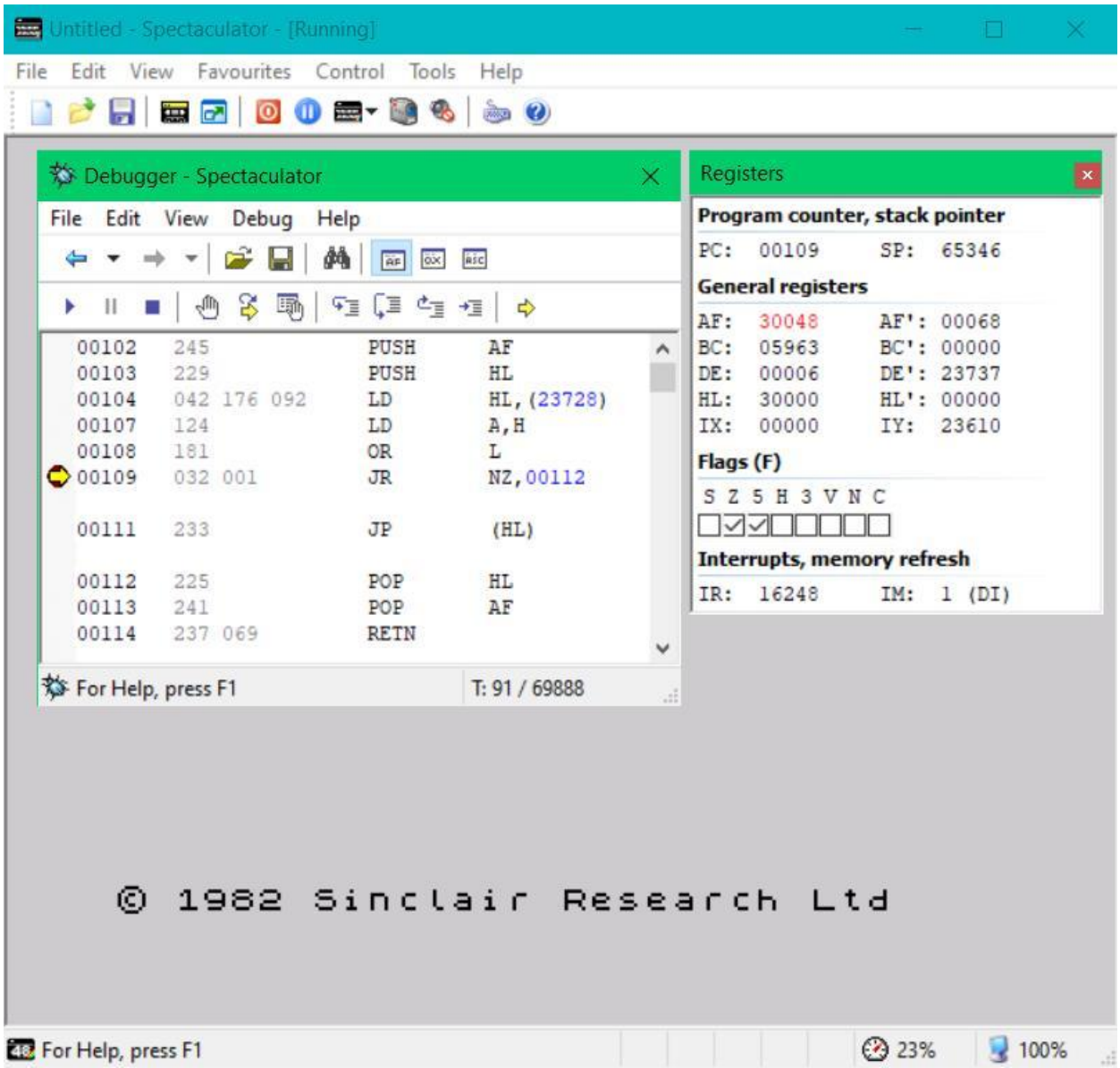


Рис. 532. Исправление маршрута Стрелочки из-за ошибки в программе.

Готово! А теперь снимите малиновую точку и нажимайте «*Trace*»:

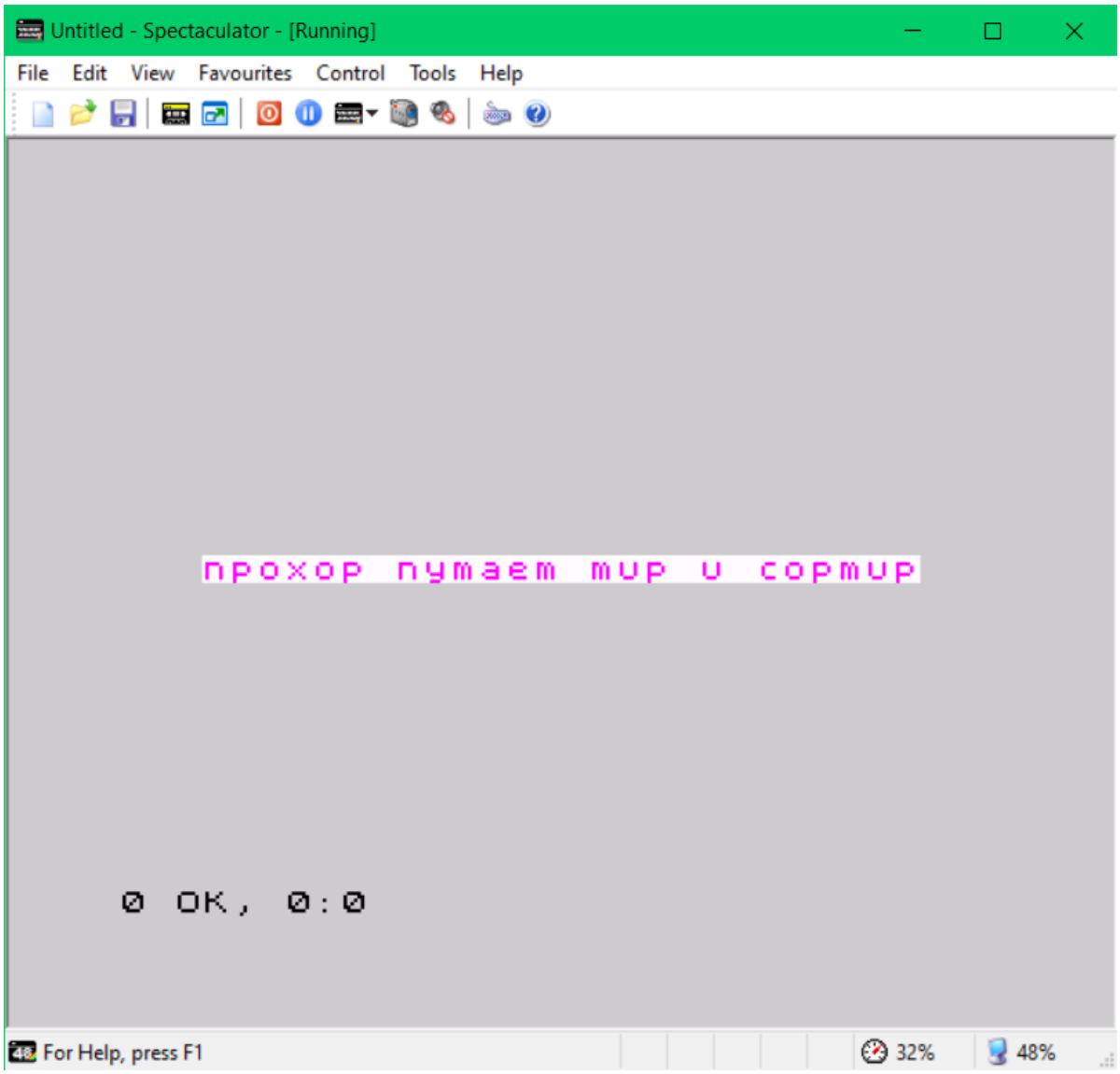


Рис. 533. Вывод собственной программы после попадания NMI-выстрела.

«NMI-пуля» попала точно в цель. В качестве выпавшего приза выполнилась ваша программа. В этой ситуации жалко чувака, который по синьке путает два совершенно разных заведения.

Предлагаю новую команду так и назвать «*Generate NMI*». Тогда полный алгоритм программы вместе с выстрелом, будет выглядеть так:

```
Debugger
Dec
Go To 109
Add Breakpoint 109
Go To 23728
23728 ← 30000
30000 ← 253 54 2 32 17 77 117 1 33 0 205 60 32
30013 ← 49 84 255 237 115 61 92 241 253 54 0 255
30025 ← 251 195 3 19 19 1 16 3 22 11 3
30036 ← прохор пумаем мир и сормир
Trace
Generate NMI
AF ← AF+64                ; установить галочку Z исправив ошибку разработчиков
Remove Breakpoint 109
Trace
```

По идее, эту ошибку после выявления, можно было смело исправлять в следующих сериях компьютера, без потери совместимости.

Глава 67
Чистая память с «EI-капканом» и глюками T-State

Краткое содержание: файлы .rom и .bin, процессорное время, обнуление памяти команды DI, EI, IM 0, IM 1

До этого момента все действия рассматривались с позиции того, что на компьютере в ПЗУ существует набор вспомогательных программ, включая BASIC. Возврат Стрелочки «домой», означал переход в цикл программ `MAIN EXECUTION`. А представьте, если внезапно обнулить все 65536 ячеек. Как поведёт себя Стрелочка, очутившись в пустыне, среди сплошных нулей?

Сегодня в рубрике «Ретро» я жёстко покритикую собственную книгу 2013 года, а именно Часть 3 «*Работы по модификации ПЗУ эмулятора ZX-Spectrum*». Особенно затрону глава 5 «*Создание и запуск простейшей программы в ПЗУ*».

Нажмите клавишу 9, и запустится наша подпрограмма, которая очистит экран, и он станет черным. Это очистится все ОЗУ от мусора. Понажимайте клавиши от 1 до 7 и вы увидите, как от нажатых клавиш меняется цвет рамки. Наша программа полностью работоспособна. Нажав клавишу 8, программа выйдет на адрес 00079, в котором ничего нет, и пойдет гулять вниз по свободным адресам. Команда RET не имеет никакого смысла, так как возвращаться то некуда. Гуляя по памяти, компьютер снова вернется к выполнению программы с адреса 0. В этом можно убедиться, нажимая клавиши с цифрами. Снова от нажатия клавиш переключаются рамки:

На самом деле, чтобы создать нормальную прошивку ПЗУ, следует также учитывать систему прерываний, и некоторые другие моменты, под которые изначально «затачивалась» стандартная программа ПЗУ.

Рис. 534. Фрагменты из книги «ZX-Spectrum-48K в эпоху windows и эмуляторов» за 2013 год.

«*…систему прерываний и некоторые другие моменты*». Шедеврально, правда? На самом деле сказано это такими обтекаемыми фразами, потому что я знать не знал, как работает система прерываний. Я видел, что в пустом ПЗУ Стрелочка иногда как то странно ходила по памяти и делала незапланированные крюки. Рискнув, я сделал наугад ту самую программу с рамками и запустил. К счастью она как-то заработала. Ну а странное поведение было списано на мифические «другие моменты», которые неподвластны начинающим любителям машинных кодов. Конечно, все мелкие несостыковки были тихонько замяты. А сколько лишних телодвижений для создания файлов в *HexEdit* и подключения ПЗУ…

Да, смешно читать себя самого и свои познания 2013 года из 2025-го.

А как всё на самом деле? Предлагаю разобраться, что это за «*система прерываний*» и «*другие моменты*».

Как ни прискорбно, но придется создать два пустых файла внешними средствами. Один должен быть размером 16384 байта и иметь расширение *.rom*, другой 49152 байта и оканчиваться на *.bin*. Я лично буду делать это в редакторе «*HxD*».

Откройте редактор и нажмите «*CTRL+N*». Появится заготовка для файла, размером 0 байт, а в углу приветственно замигает курсор.

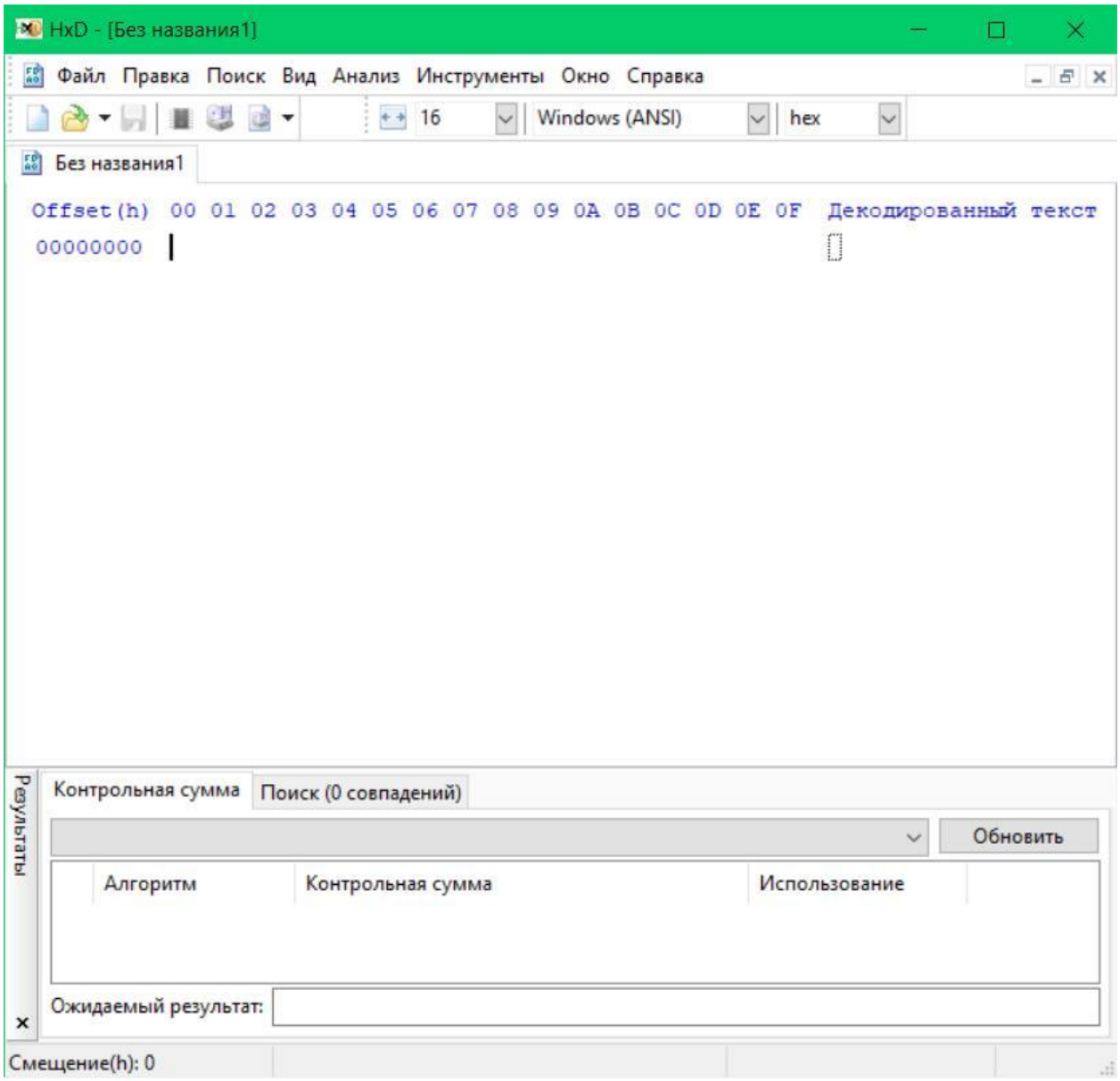


Рис. 535. Редактор HxD. Заготовка для создаваемого файла.

Начало положено. Далее в меню «*Правка*» наведите мышку на пункт «*Вставить байты*»

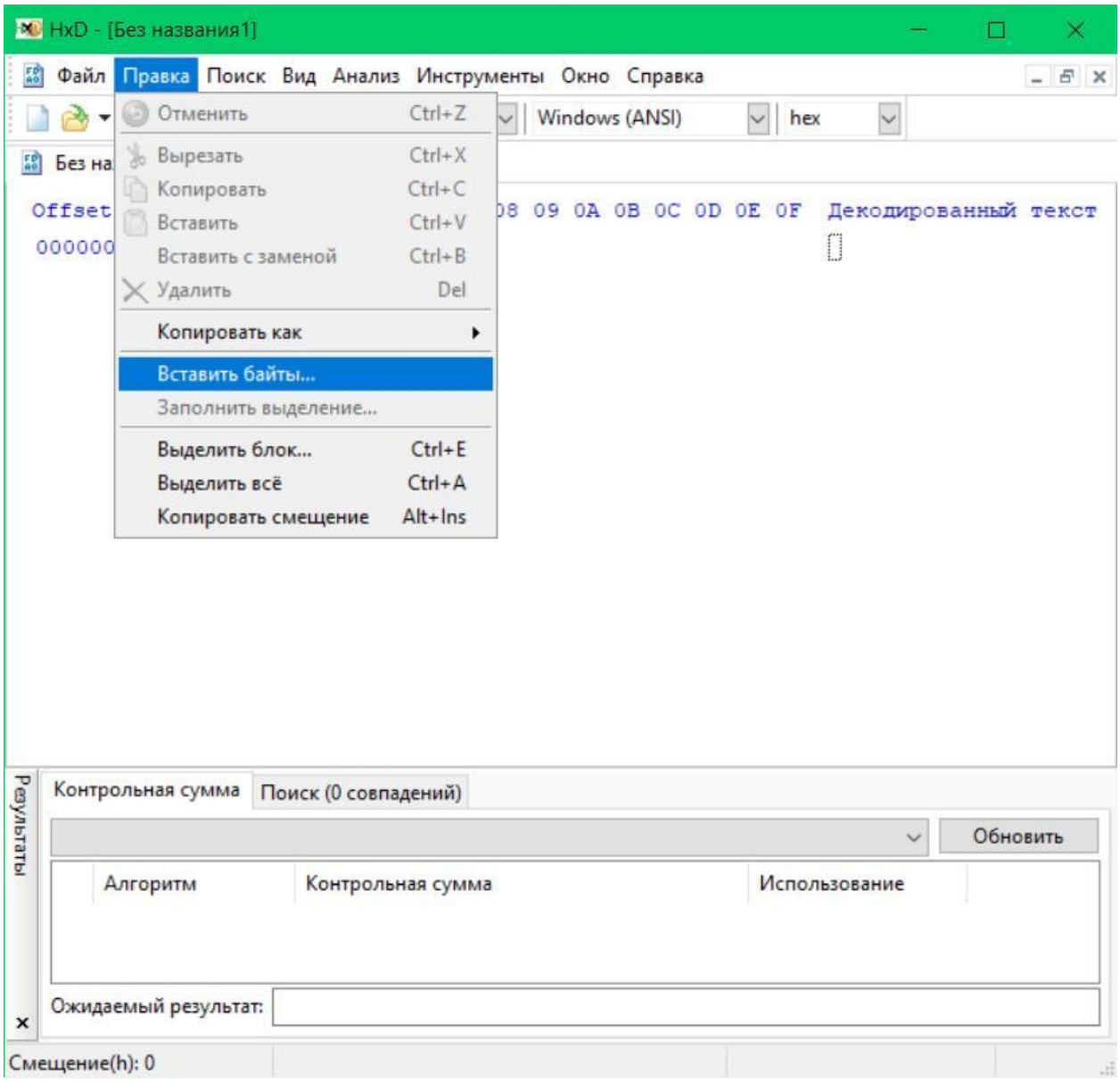


Рис. 536. Редактор HxD. Меню «Правка». Подготовка к вставке байтов.

Нажмите на него, и откроется само окошко «*Вставить байты*». В разделе «*Образец заполнения*» переключите кнопочку на «*dec*», а в полоску «*Число байт*» впишите 16384.

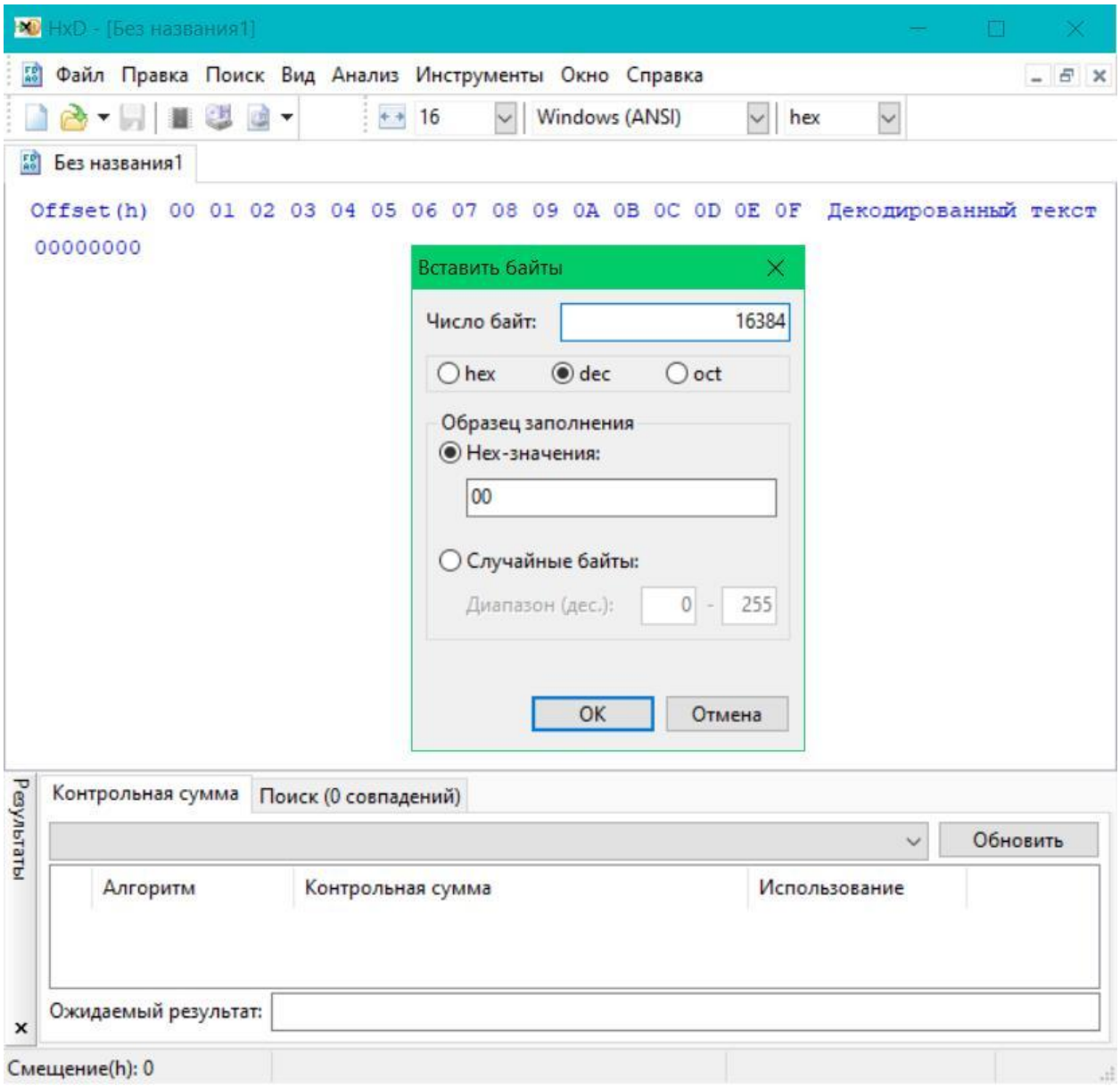


Рис. 537. Редактор HxD. Окошко «Вставить байты». Настройка шаблона массива.

Проверьте, чтобы в окошке «*Hex-значения:*» стояло 00. Это то значение, из которого будет сформирован массив, чтобы вам не набирать вручную дохренища чисел.

Нажимайте кнопку «*OK*». Окно закроется, и созданная заготовка файла заполнится нулями.

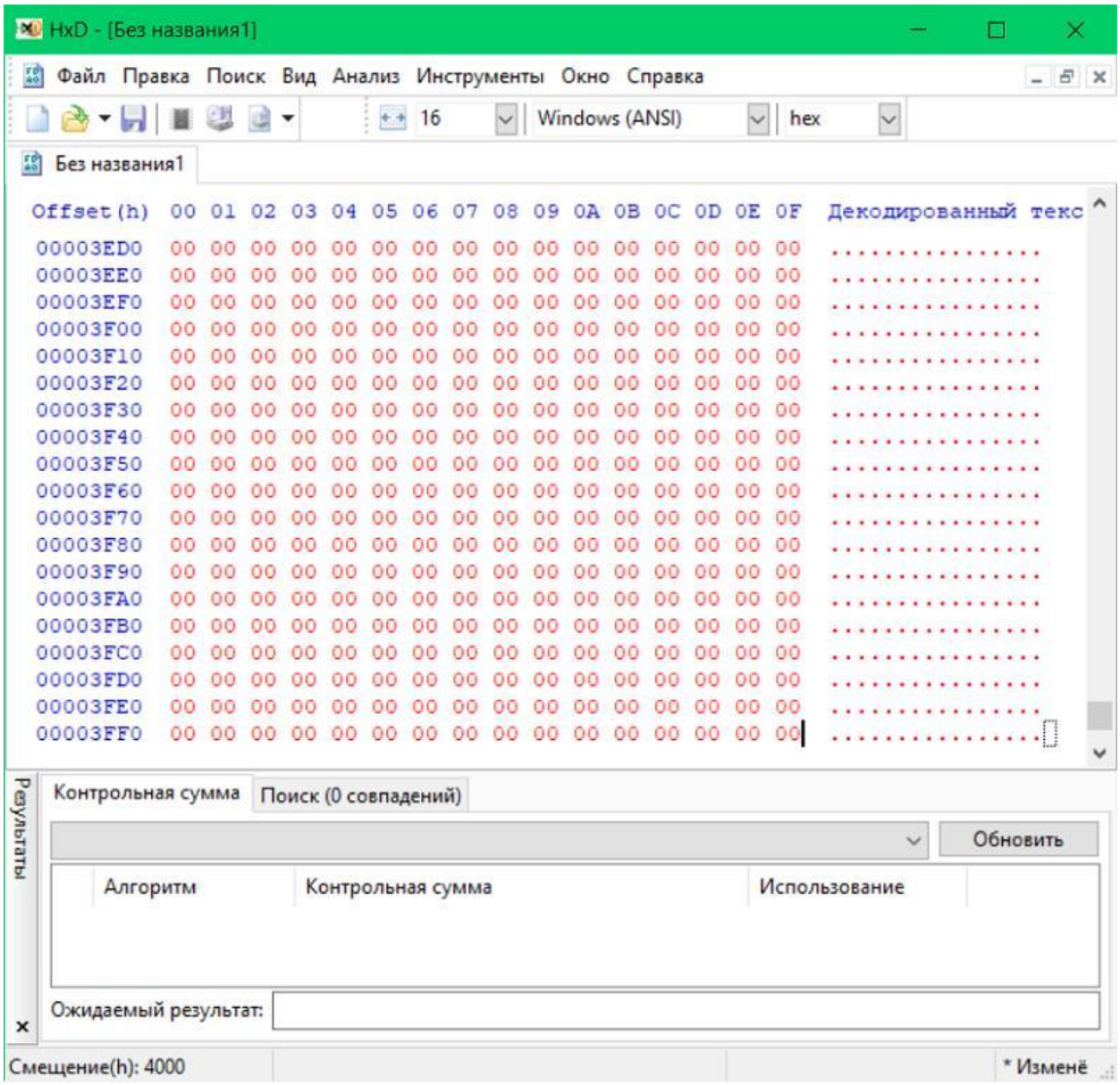


Рис. 538. Редактор HxD. Сформированный массив из 16384-х нулей.

Все готово. Осталось его сохранить. Нажмите «*CTRL+S*». Выскочит окно «*Сохранение*». Выберите папку, в которую нужно сохранить и задайте имя файлу «*Чистое ПЗУ.rom*»

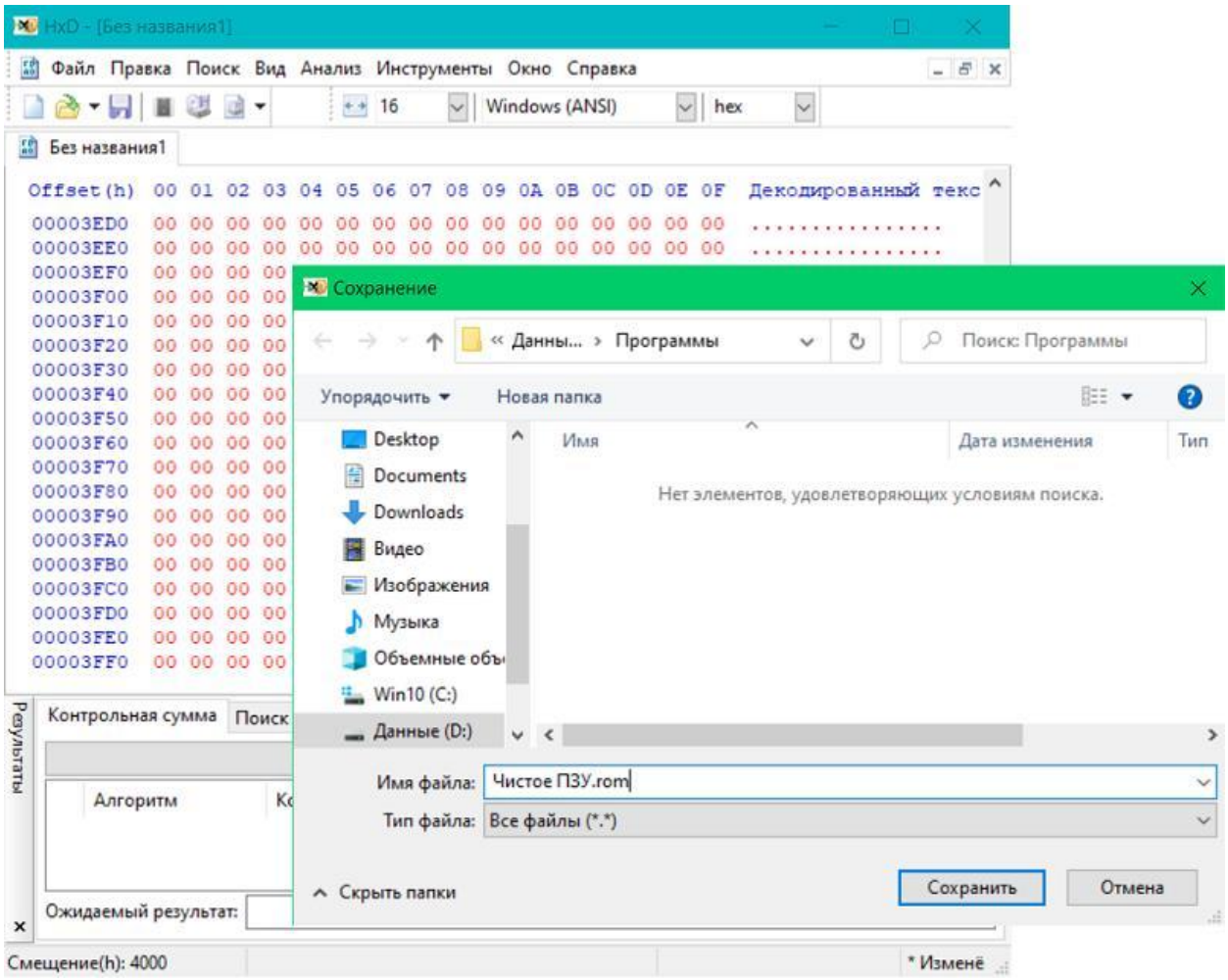


Рис. 539. Редактор HxD. Сохранение созданного файла «Чистое ПЗУ.rom».

Нажимайте «*Сохранить*» и шаблонный файл для затирания ПЗУ готов!

Теперь закройте файл и аналогичным образом создайте еще один (*CTRL+N* → «*Правка*» → «*Вставить байты*»). На этот раз задайте размер 49152.

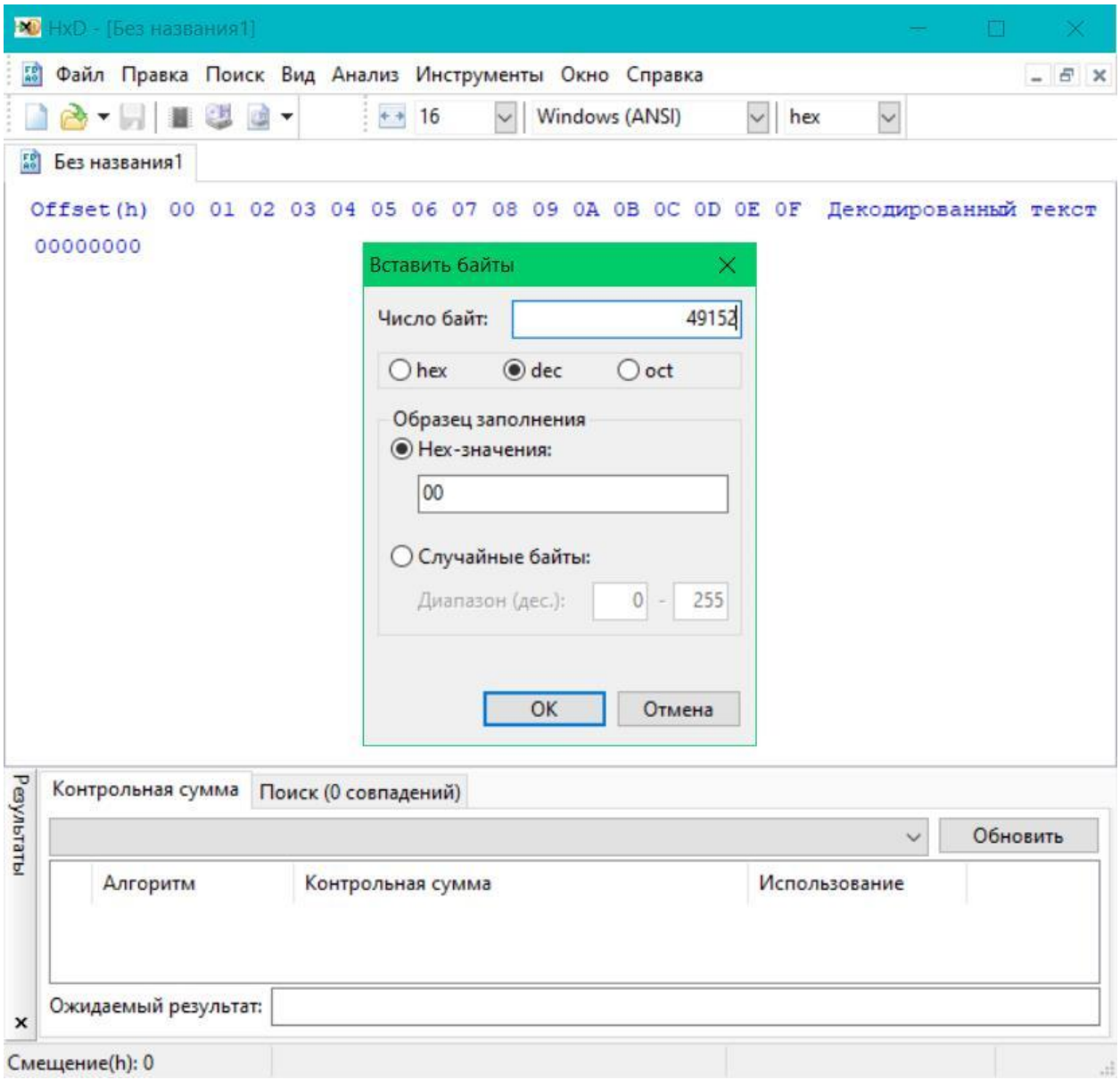


Рис. 540. Редактор HxD. Настройка шаблона массива для другого файла.

Это и есть максимальный размер ОЗУ (65536-16384=49152), который зовётся «48K». Нажмите «*OK*» и появится файл, размером с ОЗУ.

Снова нажмите «*CTRL+S*». В выскочившем окне дайте сохраненному файлу имя «*Чистое ОЗУ.bin*»

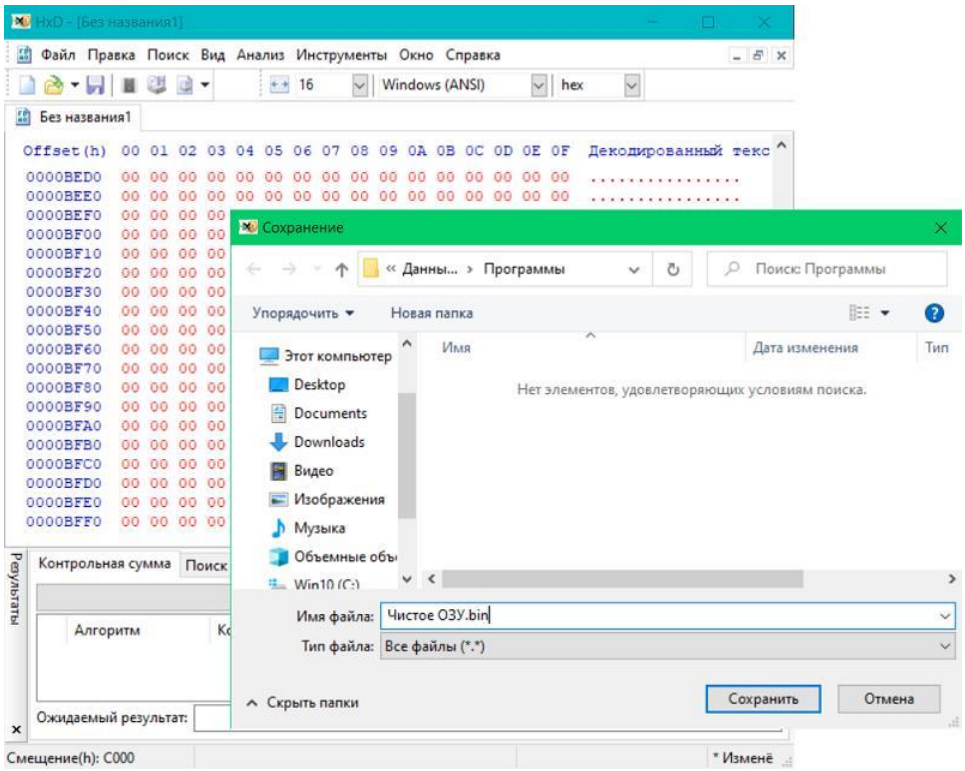


Рис. 541. Редактор HxD. Сохранение созданного файла «Чистое ОЗУ.bin».

Нажмите сохранить и закрывайте редактор. Файлы «обнуляторы» готовы.

Теперь открывайте *Spectaculator*. Полностью дав загрузится BASIC системе, поставьте малиновую точку на адрес 0, после чего нажмите «*Trace (F5)*».

А теперь на самом окне *Spectaculator'e* найдите кнопку «*PAUSE (Pause)*».

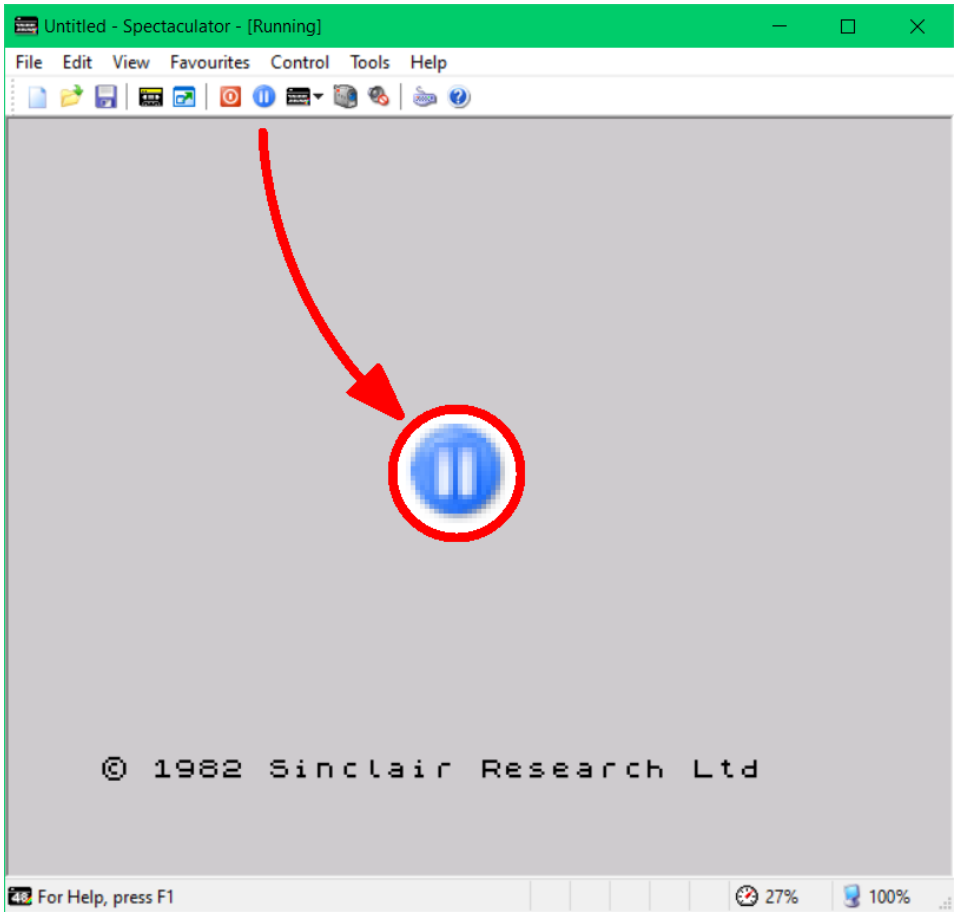


Рис. 542. Spectaculator. Кнопка «Pause» на панели.

Нажмите её или одноимённую кнопку «*Pause*» на клавиатуре. Кнопочка вдавится, и программа застынет в ожидании дальнейших действий. Перетащите в окно файл «*Чистое ОЗУ.bin*». В открывшемся окошке «*Import Machine Code*» наберите 16384 (естественно без галочки «*Execute after import*»)

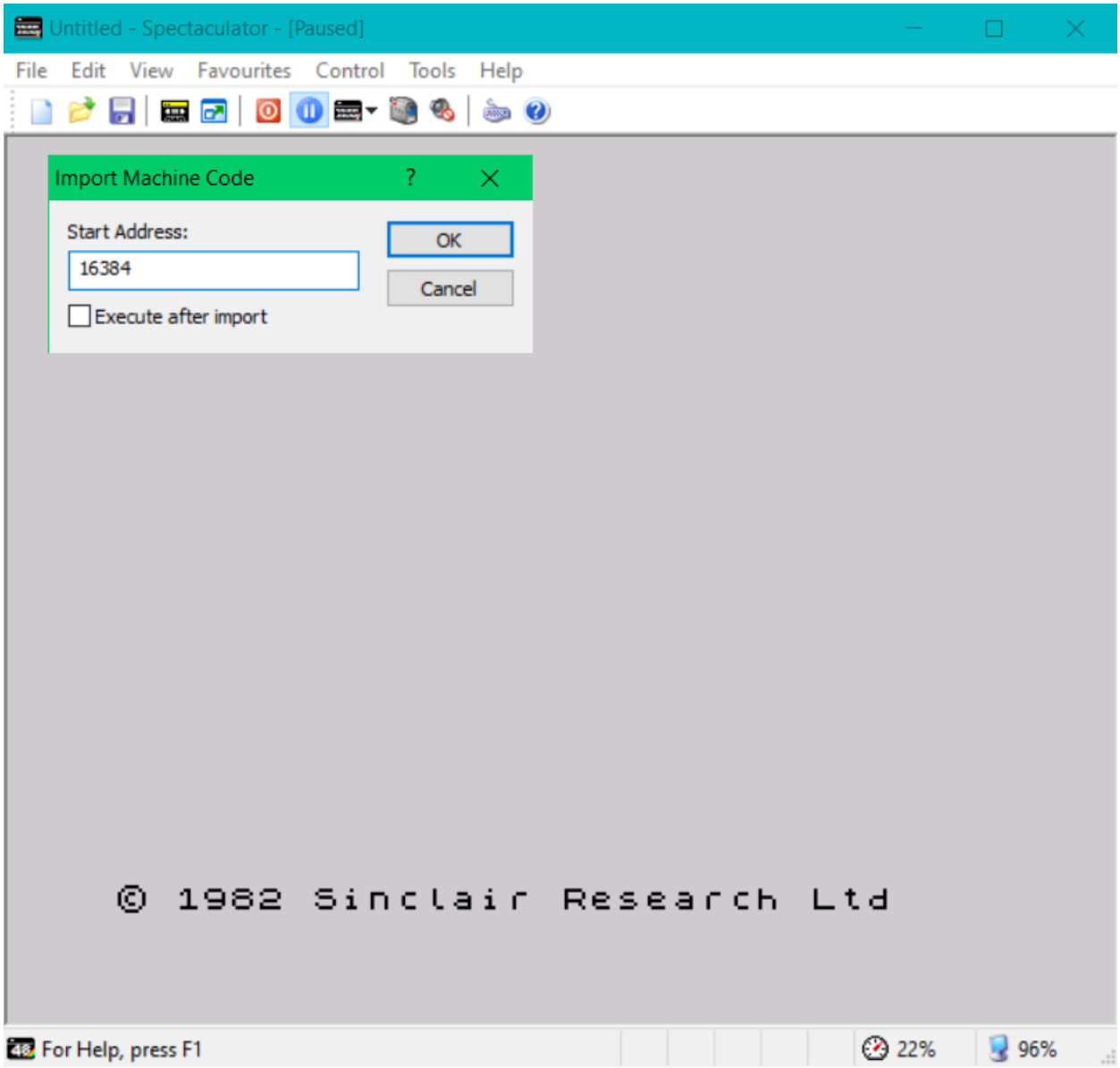


Рис. 543. Spectaculator. Окошко «Import Machine Code». Задание адреса для записи в память.

После ввода экран почернеет. Это значит, что ОЗУ уже полностью стёрлось.

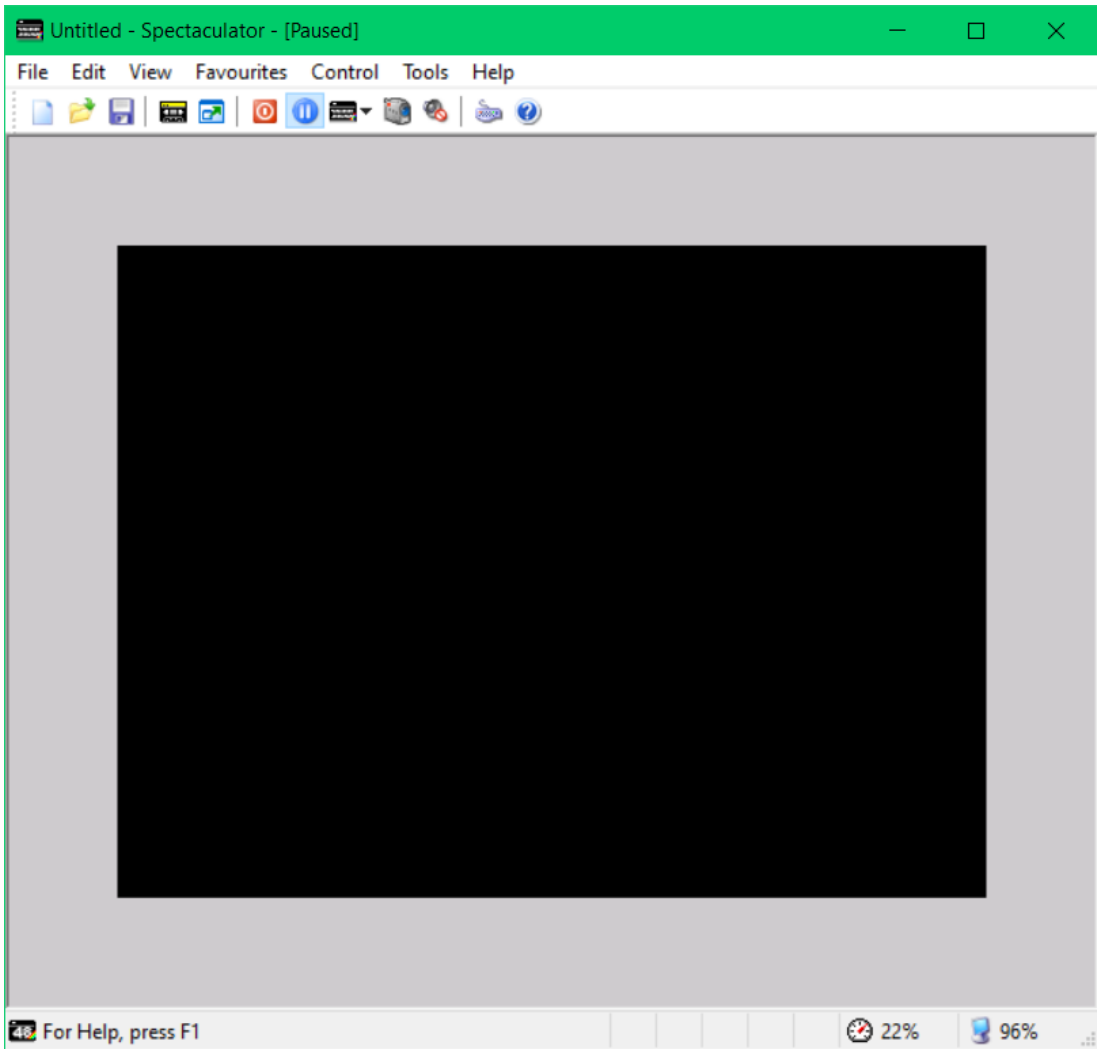


Рис. 544. Spectaculator. Очищенная память ОЗУ после перетаскивания файла с нулями.

А теперь точно также перетащите внутрь «*Чистое ПЗУ.rom*» и отожмите голубую кнопочку «*Pause*».

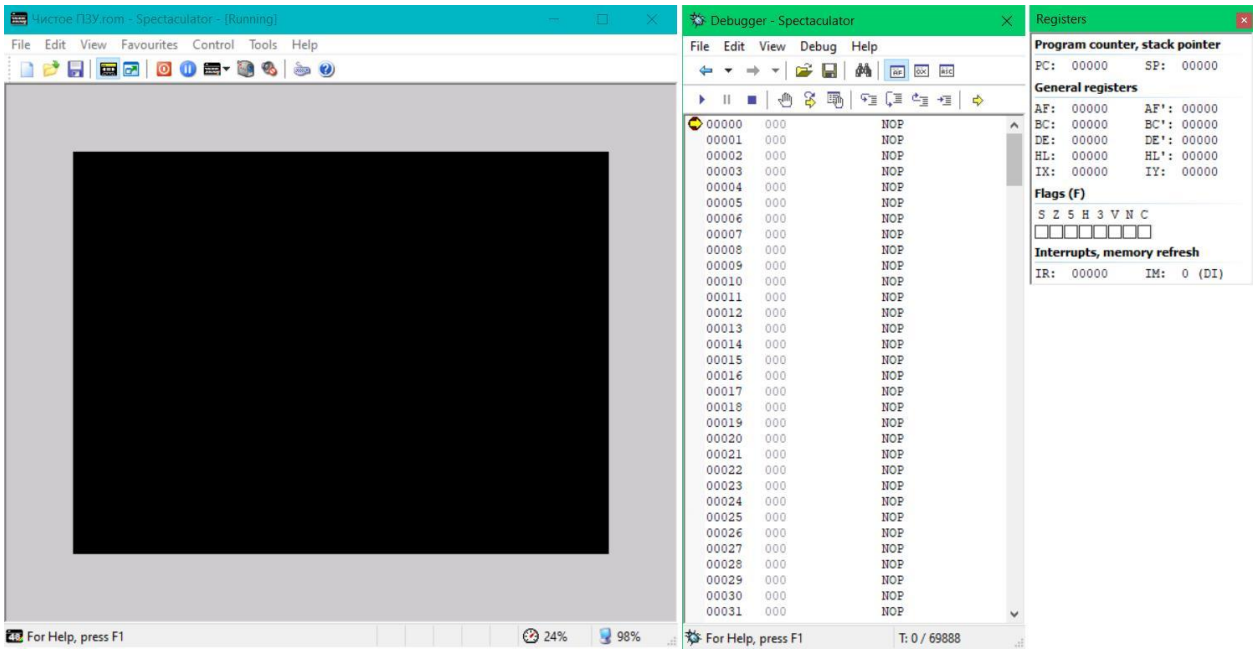


Рис. 545. Запуск Spectaculator с абсолютно пустой памятью.

Открылся отладчик. Как в главе 2, Стрелочка стоит на адресе 0 и ожидает отправления в долгий путь. Окошечко «*Registers*» также чистое, а внизу отладчика виднеется «T:0 / 69888». Вроде все как в ознакомительной главе, но… память безжизненная как пустыня. Куда ни глянь, сплошная пустота от 0 и до 65535. Программы сброса нет, да вообще ничего нет. Вот оно кристально чистое состояние, на котором можно узнать, как и каким образом будет вести себя Стрелочка с `DI`, `EI`, `IM 0` и `IM 1`.

Теперь, если вы отпустите Стрелочку, то пробежав по всем адресам, до 65535, она снова перескочит на 0, совершив круг почёта. Во время путешествия поменяются только три параметра: «R», «PC» и «T».

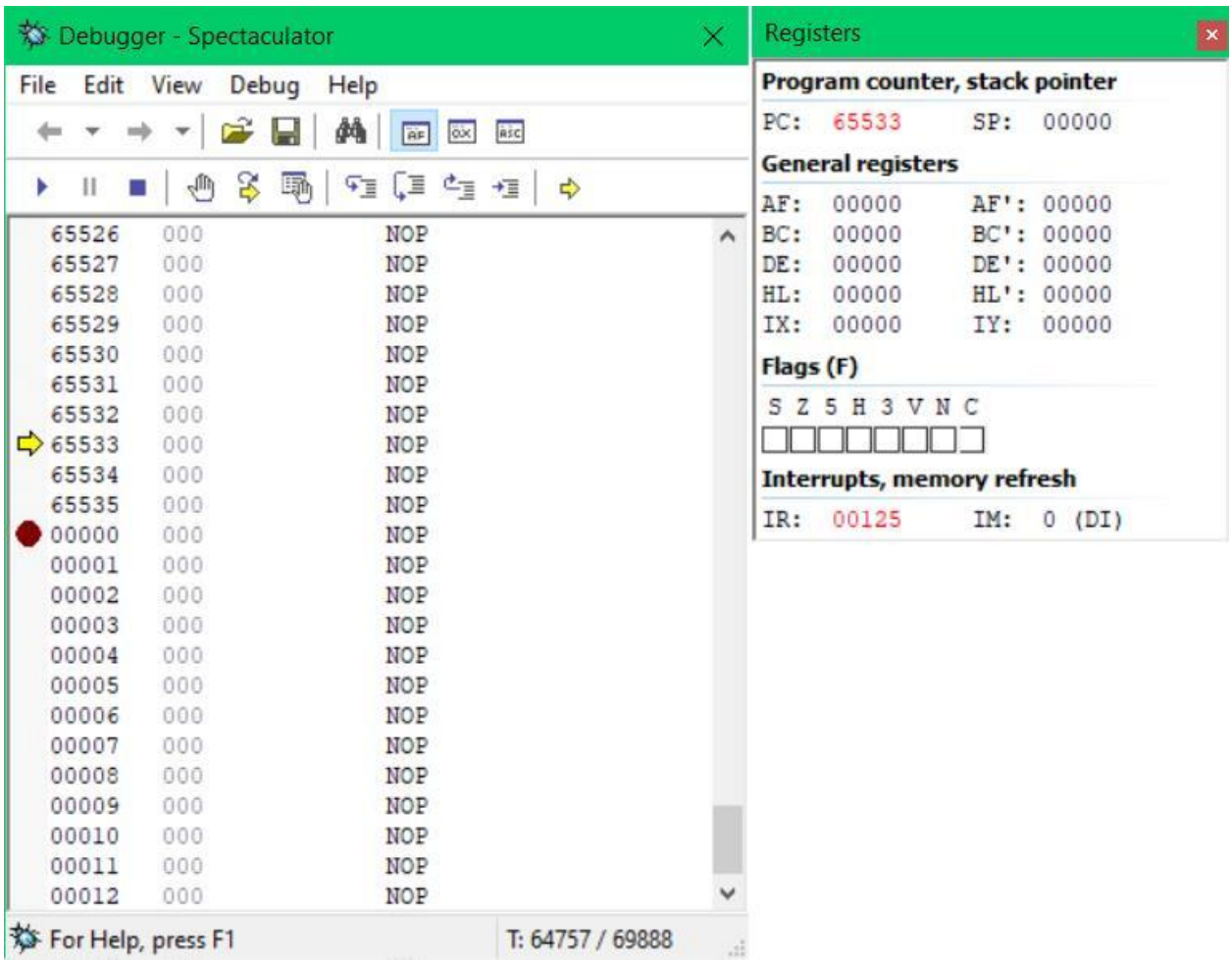


Рис. 546. Debugger. Возвращение Стрелочки в «0» после прохода по всей памяти.

Таким образом, в режиме `DI` Стрелочка действительно бегает по кругу.

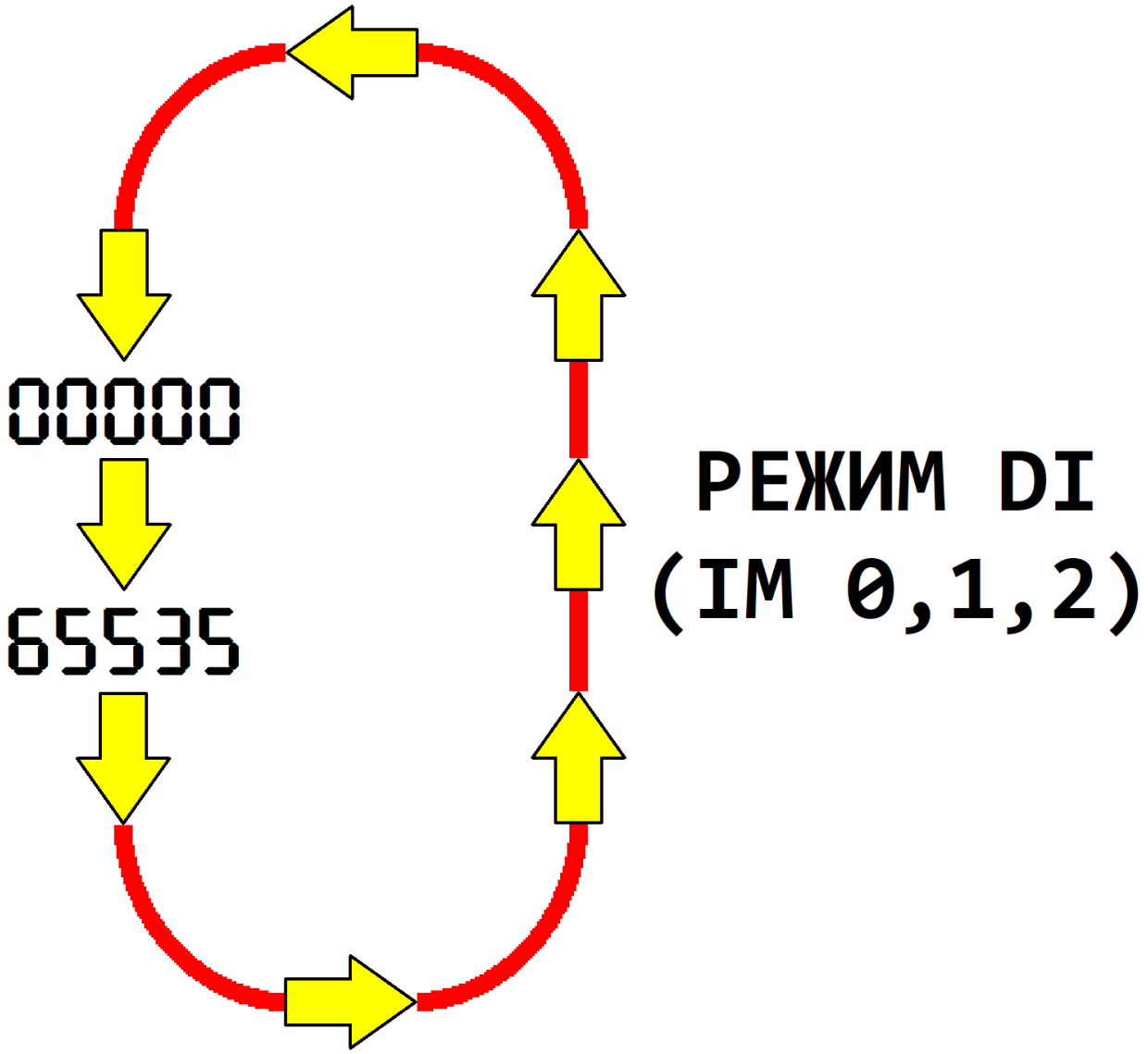


Рис. 547. Схема движения стрелочки по памяти в режиме DI.

Вернуть обратно данные в ПЗУ очень просто. Удалите все точки прерывания, закройте отладчик и нажмите «*Reset (CTRL+R)*». На экране промелькнёт знакомый сброс, а следом приветствие BASIC. Следов вандализма как ни бывало.

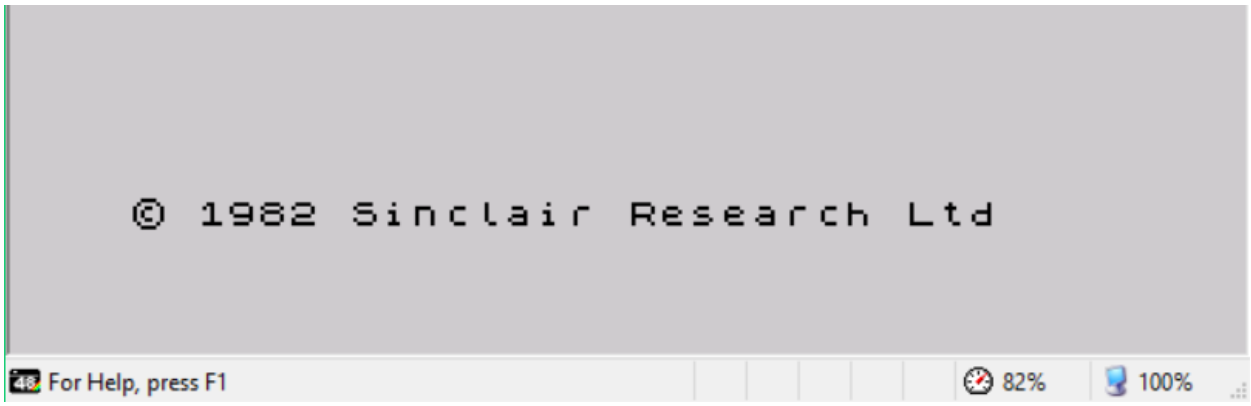


Рис. 548. Spectaculator. Возврат ПЗУ на место после сброса.

В этом преимущество перетаскивания файлов, перед полным подключением альтернативного ПЗУ, как описывалось в старой книге. После экспериментов любой сложности достаточно одного сброса.

Для этой программы обнуления на 1 сеанс предлагаю следующий алгоритм:

```
Debugger
Dec
Add Breapoint 0
Trace
Pause [ON]
Open File [Чистое ОЗУ.bin]
Import Machine Code 16384
Open File [Чистое ПЗУ.rom]
Pause [OFF]
Remove Breapoint 0
Reset
```

Тут новыми командами станут:
«*Pause [ON]*» - поставить на паузу окно *Spectaculator*.
«*Pause [OFF]*» - снять с паузы окно *Spectaculator*.

А теперь предлагаю немного усложнить задачу Стрелочке, и после полного обнуления памяти единоразово выполнить команду `EI` в том же режиме `IM 0` и посмотреть, как она отреагирует на практике.

Для подготовки выполните следующий фрагмент алгоритма. Это чуть усовершенствованный вариант прошлого:

```
Debugger
Dec
Add Breapoints 0, 16385, 17471
Trace
Pause [ON]
Open File [Чистое ОЗУ.bin]
Import Machine Code 16384
Open File [Чистое ПЗУ.rom]
Pause [OFF]
16384 ← 251
Trace
```

Выполнив этот алгоритм, Стрелочка остановилась на отметке 16385, выполнив предыдущую команду `EI`.

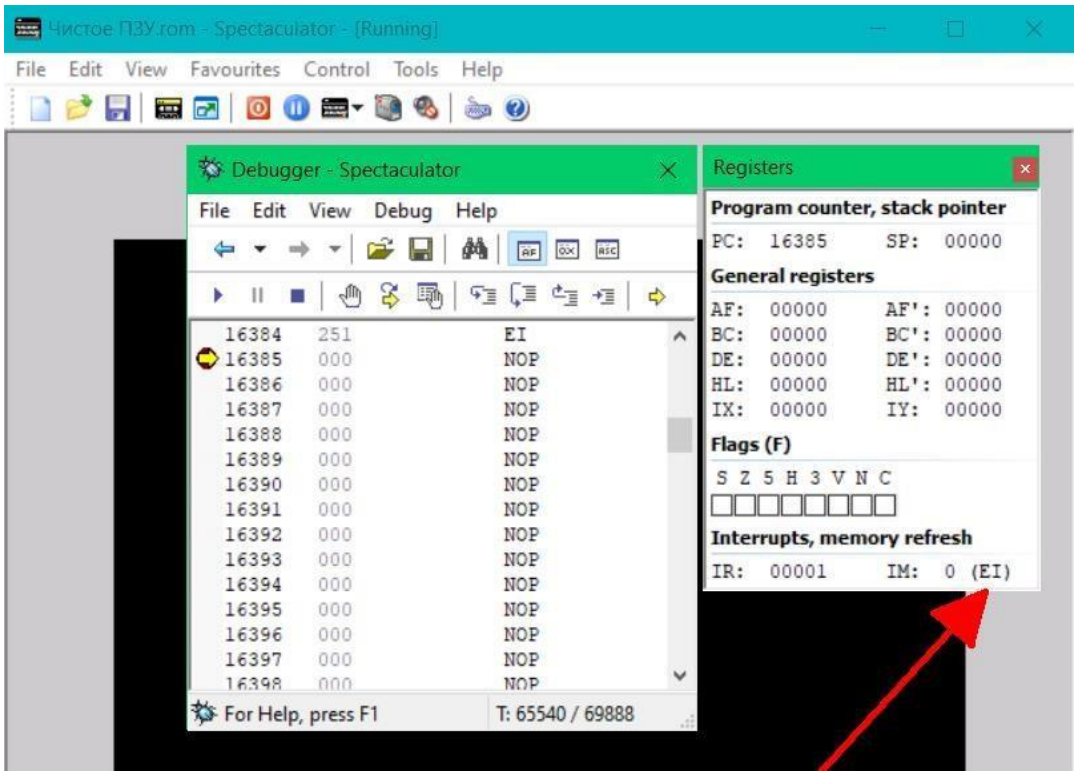


Рис. 549. Spectaculator. Активация режима EI.

В окошечке «*Registers*» успешно переключился режим. Теперь удалите отработавшую `EI` из адреса 16384, заменив на 0 (16384 ← 0). Обратите внимание: все эти танцы с бубнами совершаются потому, что в отладчике не предусмотрено прямого редактирования значений в этом окошечке. Также вы не сможете на лету поменять параметр `IM`. Для их изменения придётся также временно выставлять нужную команду, выполнять, а потом удалять. На мой взгляд, это достаточно серьёзная недоработка программы *Spectaculator*.

После удаления команды снова нажмите *Trace*:

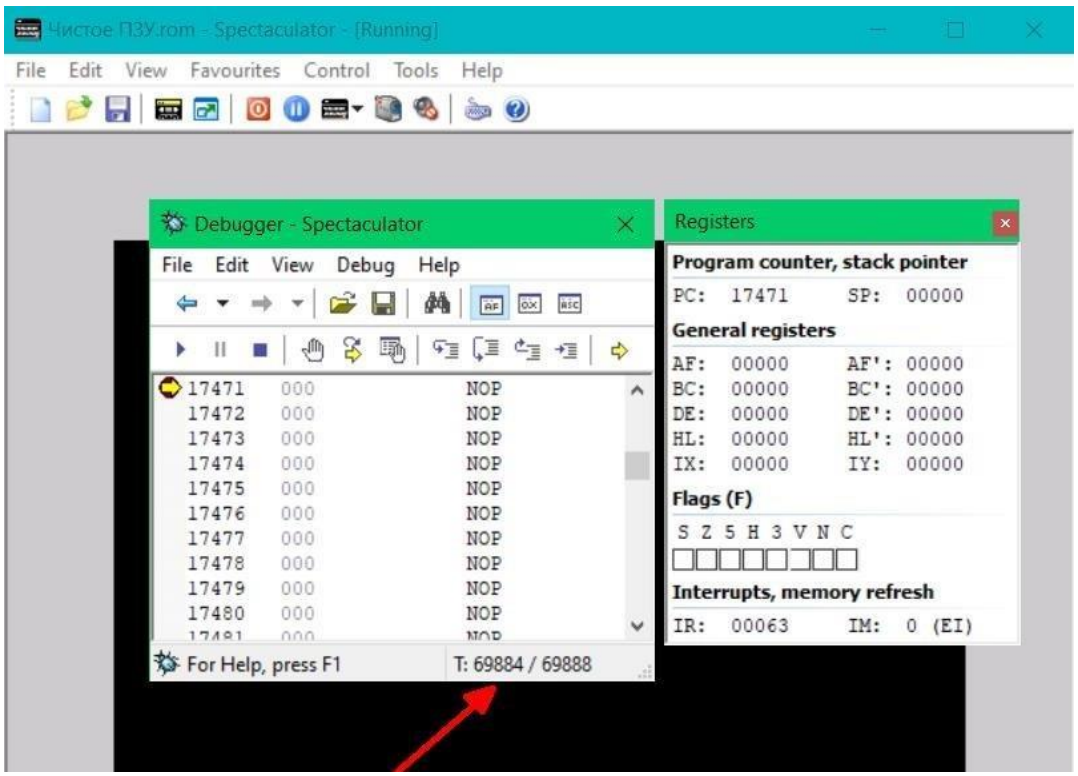


Рис. 550. Spectaculator. Один шаг до «Нового Года» и перехода в адрес 00056.

Стрелочка остановилась на адресе 17471. Обратите внимание на значение «T» внизу окна. Оно равно 69884, а это значит, что в волшебном мире без четырех секунд полный процессорный год.

Нажмите «*Step Into (F11)*» и Стрелочка внезапно....

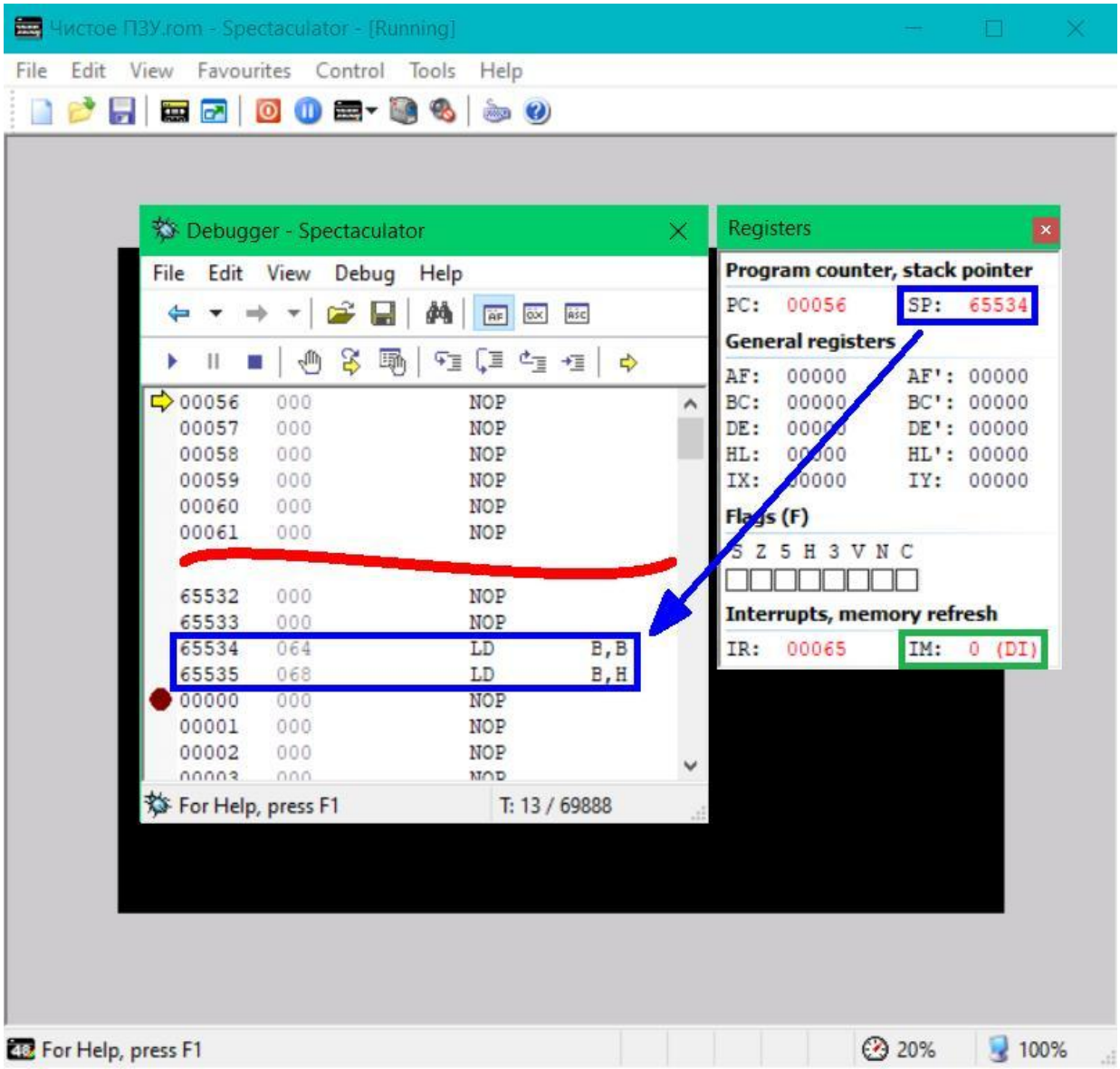


Рис. 551. Выполнение скрытого алгоритма в режиме EI после полного цикла процессорного года.

…очутилась на адресе 00056. Обратите внимание. Если следующий выполняемый шаг по прогнозам даёт перехлёст времени через 69888, то происходит преждевременный переход на адрес прерывания. В данном случае полный цикл так и не наступил, а переход произошёл после 69884.

Более того, несмотря на кристально чистую память, внутренними алгоритмами программы выполнилась небольшая последовательность в машинных кодах. Произошел принудительный переход по адресу 00056, «SP» поднялся на 2 ячейки выше и стал 65534. В ячейки, указанные «SP» записался адрес, до которого так и не дошла Стрелочка (64+68*256=17472), и выполнилась команда DI. Время «T» сбросилось и стало 13. Потому что накинулось время на выполнение «внутреннего машинного кода».

Таким образом, во внутреннем мире выполнилось:

```
DI
CALL 56
```

или

```
DI
RST 56
```

Ну а дальше, в режиме `DI` Стрелочка снова пойдёт гулять полными кругами по всей памяти от 0 до 65535.

Таким образом, маршрут Стрелочки с единоразовой установкой `EI` будет следующий:

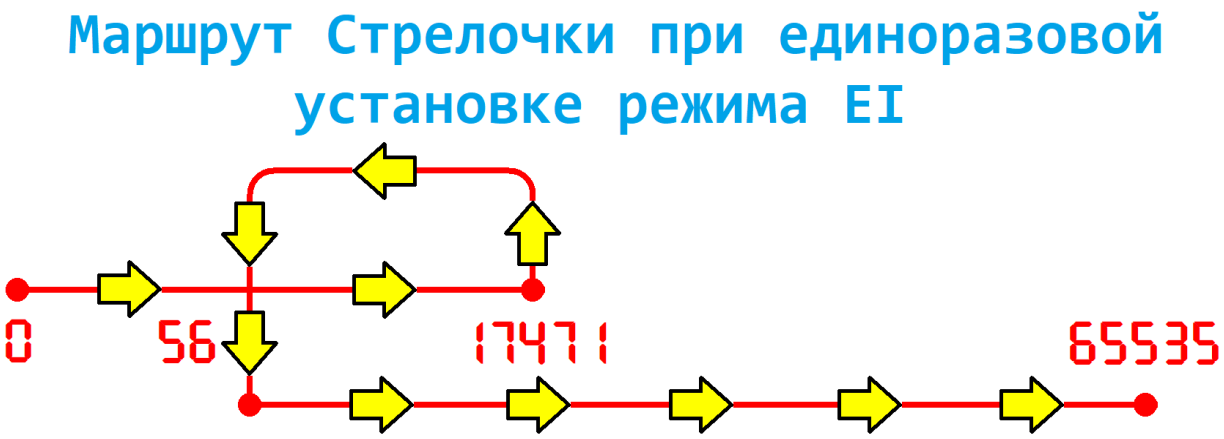


Рис. 552. Схема движения Стрелочки при одноразовом попадании в «EI-капкан».

В общем, процесс не отличается от рассматриваемого в прошлой главе «NMI-выстрела». Поскольку в данном случае Стрелочка наступает на подложенную команду, явление получило название «EI-капкан».

Нажмите «*Reset*» для возврата *Spectaculator'a* в работоспособное положение.

Предлагаю заключительный опыт на эту тему. На этот раз задача упрощается.

Теперь, предлагаю там же установить команду `EI` на постоянной основе. После срабатывания капкана её не убирать. Тогда каждый раз, достигая адреса 16384, Стрелочка автоматически будет включать режим `EI`, тем самым заставляя себя снова и снова возвращаться в точку 56. Пробегая вперёд, она снова попадёт в капкан…

Подготовительная программа в God Mode будет выглядеть следующим образом:

```
Debugger
Dec
Add Breapoints 0, 17471, 17524
Trace
Pause [ON]
Open File [Чистое ОЗУ.bin]
Import Machine Code 16384
Open File [Чистое ПЗУ.rom]
Pause [OFF]
16384 ← 251
Trace
```

Запустив данную программу, как и в прошлом примере, произойдет активация `EI`. Дойдя до адреса 17471, Стрелочка ожидаемо перескочит в 56.

Поскольку `EI` в 16384 не убрано, то по прохождению «капкана» снова включится режим `IM 0`. Поскольку повторный отсчет начался с 00056, то Стрелочка пройдет по памяти чуть дальше обычного. На этот раз, крайней точкой станет адрес 17524.

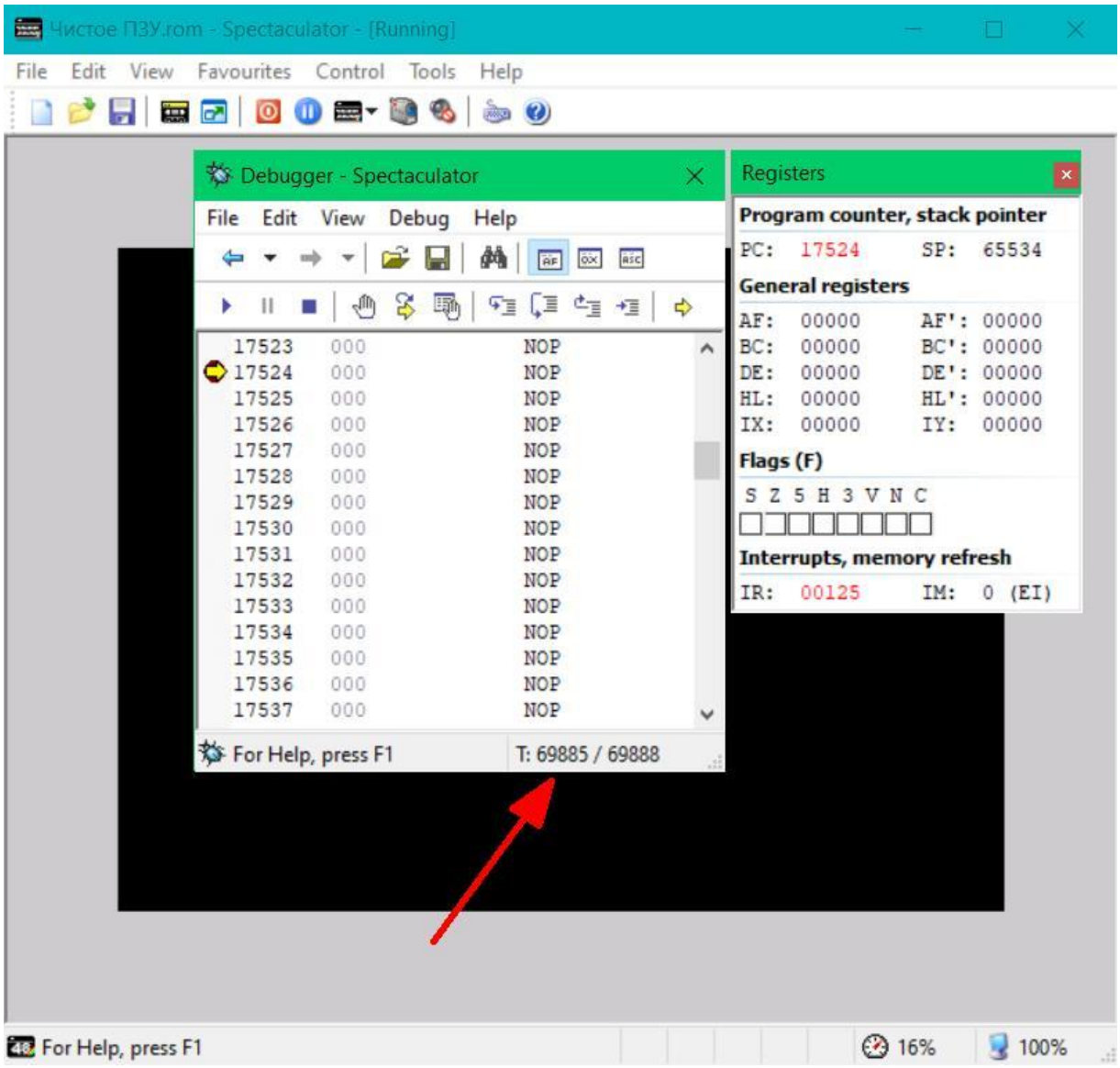


Рис. 553. Крайняя точка перед возвращением в адрес 56 после 2-го круга. T-State=69885.

И теперь внимание на «T», которое, в отличие от прошлого примера, равно 69885. Следующий шаг по расчетам должен выполниться за 4 «НеСекунды». Понятно, что полностью уложится в текущий цикл нереально. Произойдёт перехлёст и на окончание выполнения «прошлогодней» команды и одна «НеСекунда» добавится к будущему году.

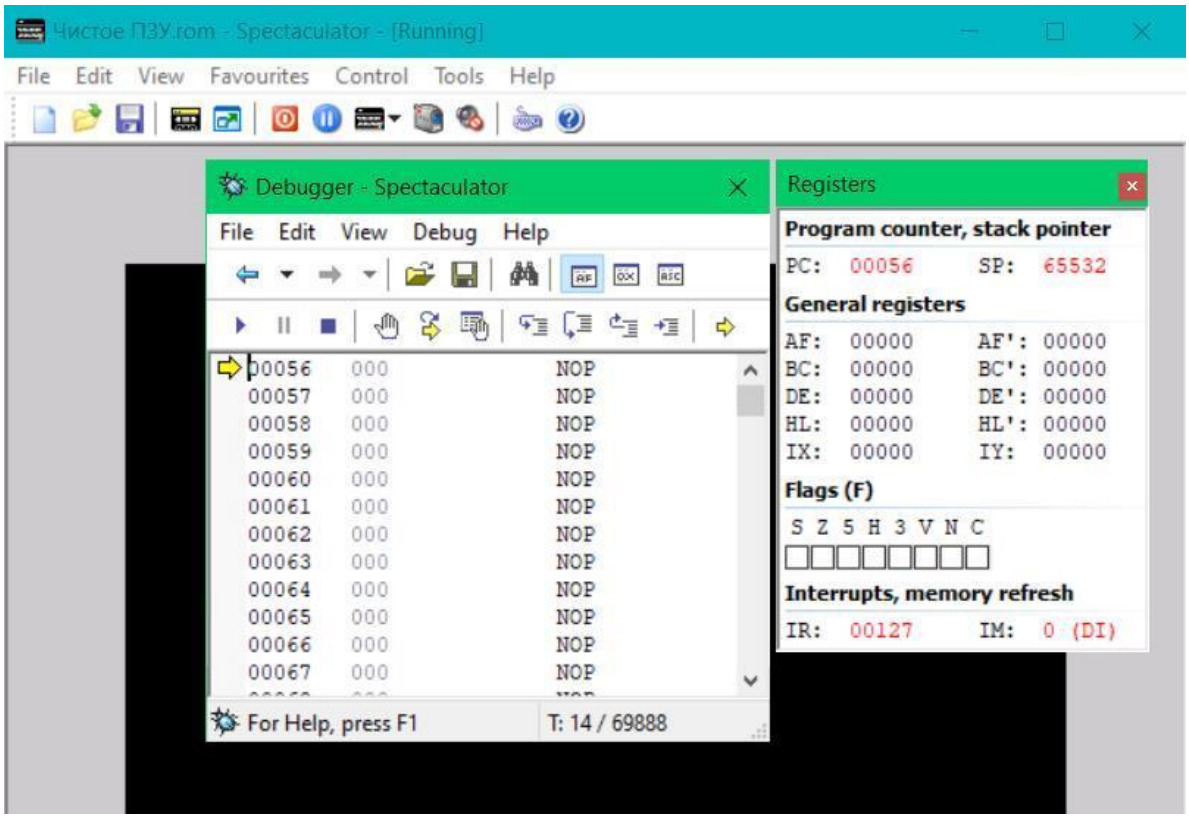


Рис. 554. Накидывание прошлогодней секунды на следующий год.

Как и в реальной жизни, идеальный баланс года не получается. После перехода, заёмная секунда прибавилась ко времени выполнения внутренних алгоритмов (13+1=14). Таким образом, отсчёт «T» с начала года стал 14. На следующем круге, после возврата в 56, «T» будет равным 15. А вот на 4-м круге смещение станет 4 «НеСекунды», поэтому переход произойдёт по адресу 17523/24.

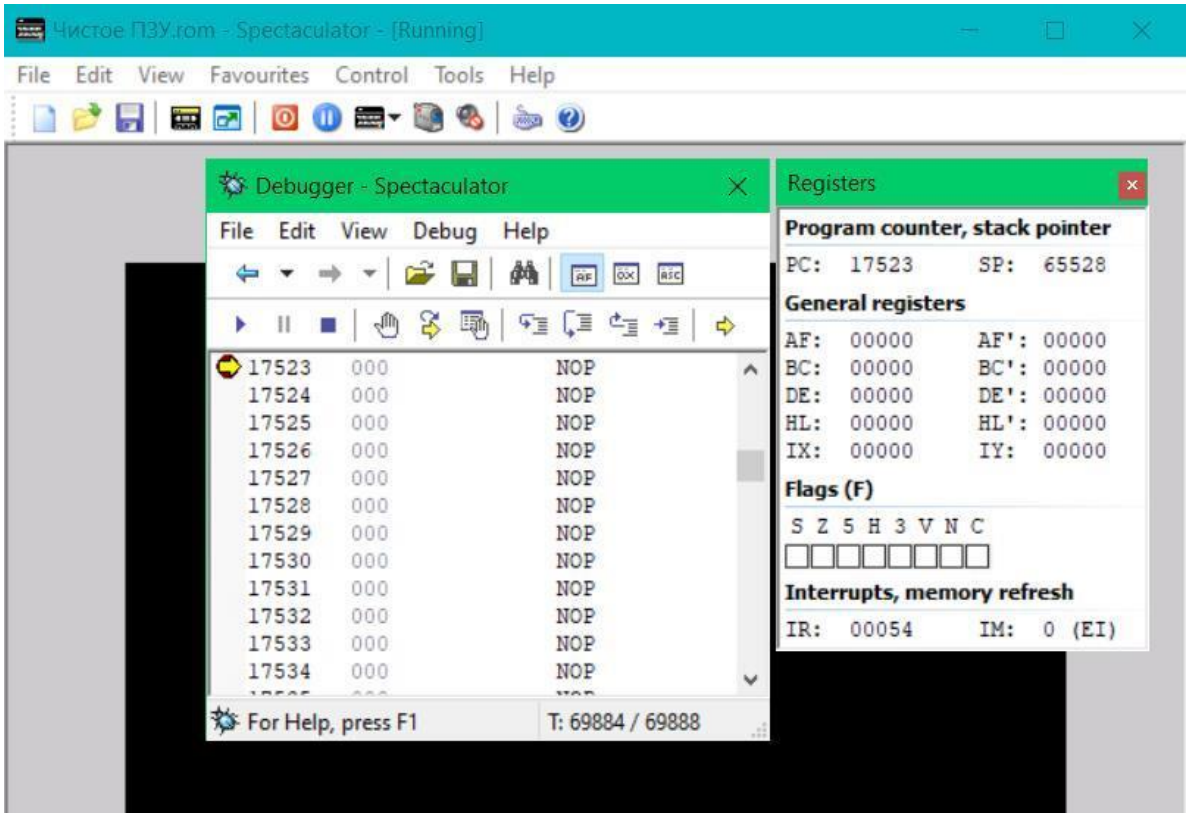


Рис. 555. Стрелочка за шаг до перехода на адрес 56.

После раннего перехода, цикл снова придёт к первоначальной точке. Таким образом, раз в четыре круга Стрелочка будет встречать «антивисокосный год», переходить по адресу 17523 и компенсировать получающийся сдвиг. После него следующие три «процессорных года» снова будет переход по адресу 17524.

Ну а постоянный годовой прирост столбика «SP» с невостребованными адресами возврата никто не отменял. Ведь никакой разновидности RET из адреса 56 не предусмотрено. Снизу вверх начнется постепенное заполнение памяти циклическими значениями.

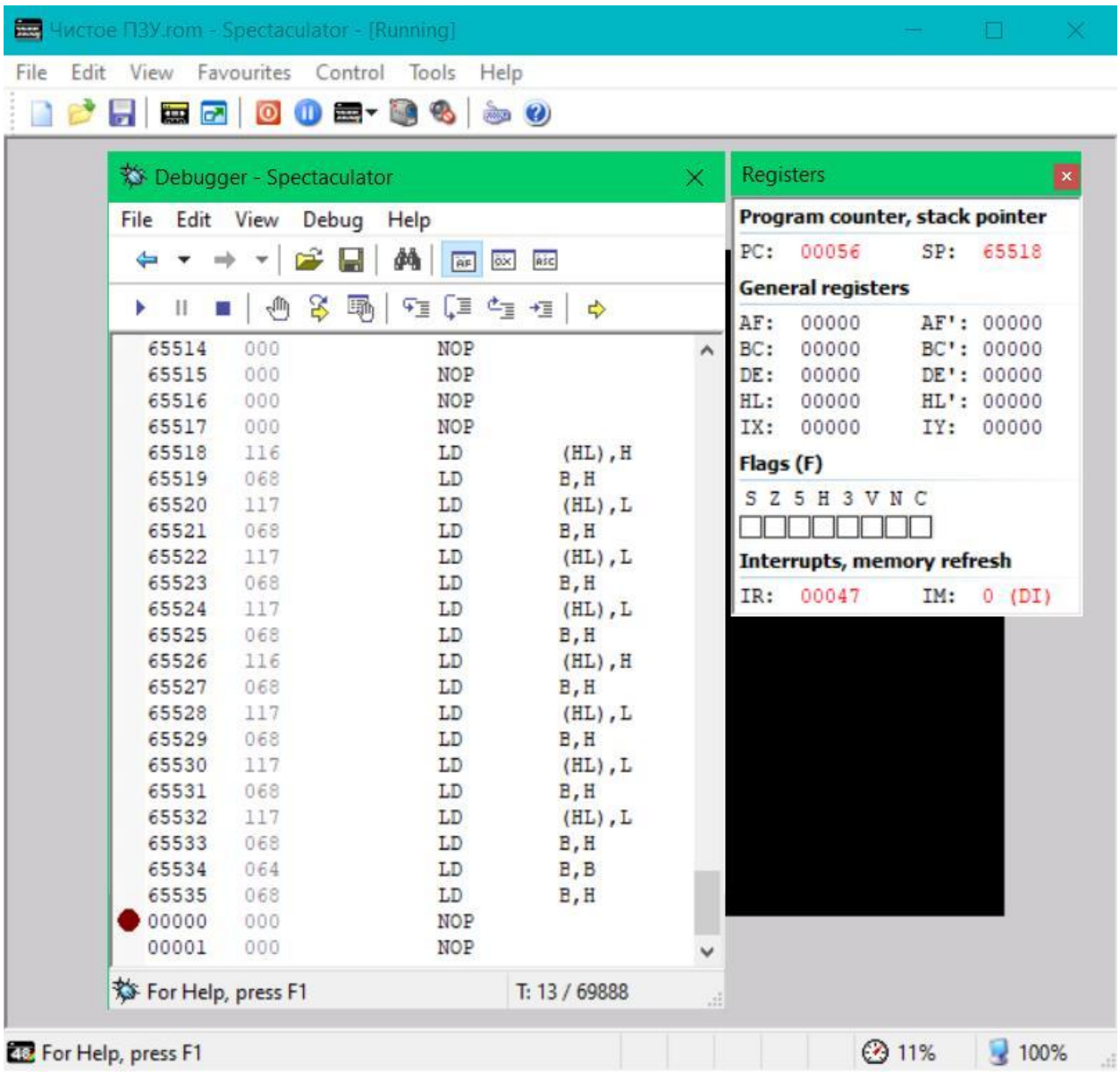


Рис. 556. Начало заполнение памяти невостребованными адресами перехода.

Такой цикл будет достаточно долгим, около 5 минут. В конечном итоге, «SP» дойдёт до области экрана, и начнутся визуальные эффекты

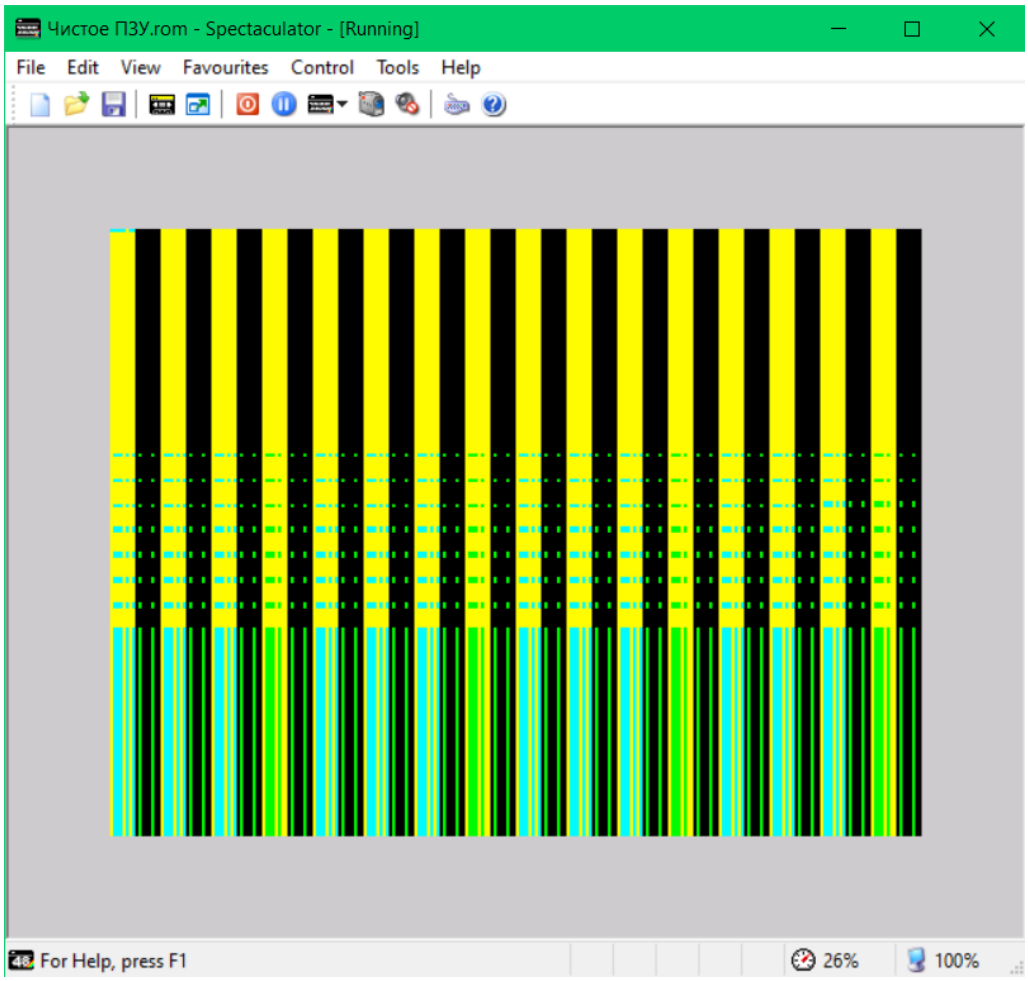


Рис. 557. Spectaculator. Рост столбика SP в экранной области.

Вскоре «SP» достигнет адреса 17524 и начнёт ставить преграды на пути у Стрелочки. Её круги немного уменьшатся. Записываемое в память содержимое тоже слегка изменится. Поскольку команды из адресов получились достаточно безобидные, Стрелочка продолжит путь.

Так будет продолжаться, пока «SP» не доберется до 16384. Уничтожив команду `EI`, прирост значения «SP» прекратится.

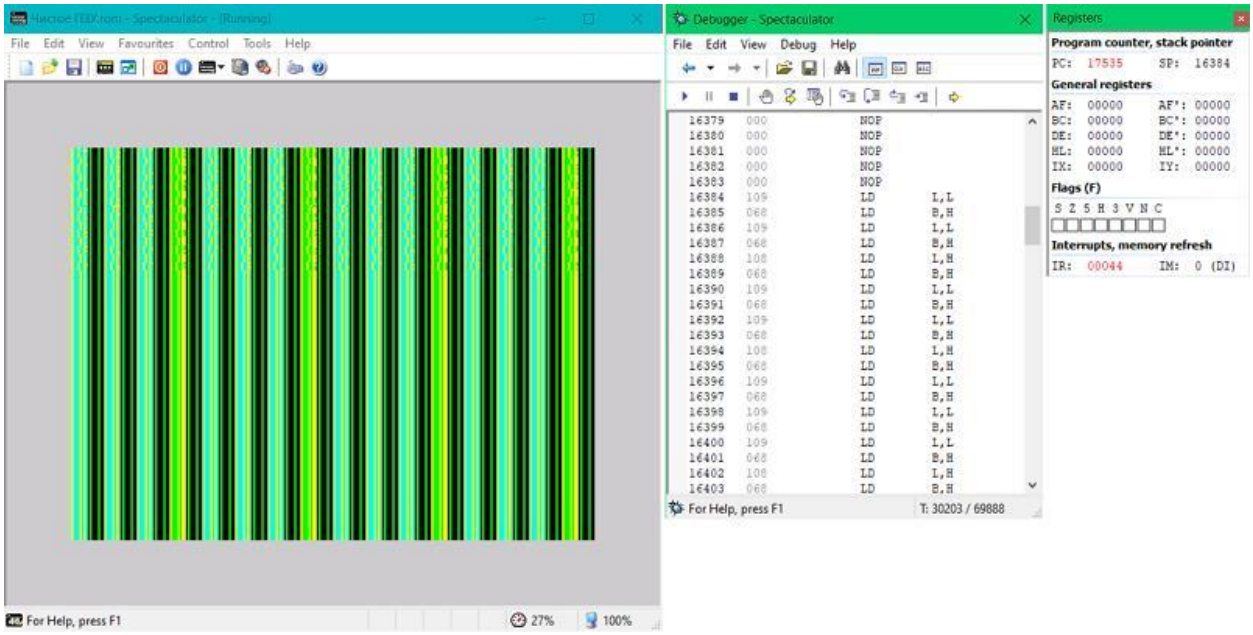


Рис. 558. Выход из режима прерываний после самопоедания команды EI.

Оставшись в режиме `DI`, Стрелочка снова начнёт гулять по памяти большими кругами. Остаётся только нажать  «*Reset*», чтобы очередной раз привести программу в чувство.

Но даже такую ситуацию можно назвать «идеальной» и складывается она не всегда. Это, скорее, исключение. Если `EI` поставить в произвольном месте памяти, то при переходе по адресу 56 в «`SP`» сформируются другие команды. Например, от команды `JP` или `JR`, всё могло закончиться гораздо раньше и банальнее. Стрелочка бы просто застряла на случайном цикле.

Таким образом, без `RET`, и изоляции адреса 56, ничего хорошего не выйдет. Цикл будет работать, в зависимости от сложившихся команд. Режим `EI IM 1` отдельно рассматривать не имеет смысла. В нём точно также переход будет по адресу 56, следовательно, сценарий получится аналогичный.

Это и есть та самая «система прерываний». Создавая программу в книге 2013 года, я даже не подумал зарезервировать критический адрес 56 и бездумно перекрыл его программой. Именно поэтому я не развил опыт с режимами `IM 0/1` в книге «*ZX-Spectrum-48K в эпоху windows и эмуляторов*». Включив прерывания на примере с рамками, я заметил, что программа начинала странно себя вести и сбиваться. Толком не разобравшись, я быстренько свернул все запланированные эксперименты с ПЗУ. Решив не продолжать эту тему, я закончил раздел, ограничившись примитивным вариантом с режимом `IM 0`.

Ну а теперь о глюках самой программы, а точнее, времени выполнения команд в зависимости от пройденного пути и на разных точках памяти. Во всех версиях *Spectaculator'a* имеется достаточно неприятная ошибка с отсчётом процессорного времени «*T-State*». Она хорошо заметна на примерах с полностью обнулённой памятью. Поскольку опыты, описанные ранее, производились в верхней части памяти, до адреса 20000, то эти эффекты никак не проявлялись.

Предлагаю теоретически посмотреть на суть проблемы. Допустим, вы обнулили память, как описывалось в начале главы, и пустили Стрелочку на прогулку. Каждый шаг от ячейке к ячейке она проходит за 4 «НеСекунды». Следовательно, полный круг по пустой памяти должен быть пройден за 262144 единицы этого хитрого времени.

До адреса 17472 она добирается ровно «Волшебный год», что вполне соответствует ожиданиям. Еще через год, согласно теоретическим подсчётам, Стрелочка должна добраться до адреса 34944, где «*T-State*» повторно обнулится. Однако на практике, всё идёт совсем не по планируемому сценарию, и цикл заканчивается гораздо раньше. Мало того, на конце появляются нечётные числа, хотя в памяти абсолютно никаких других команд нет.

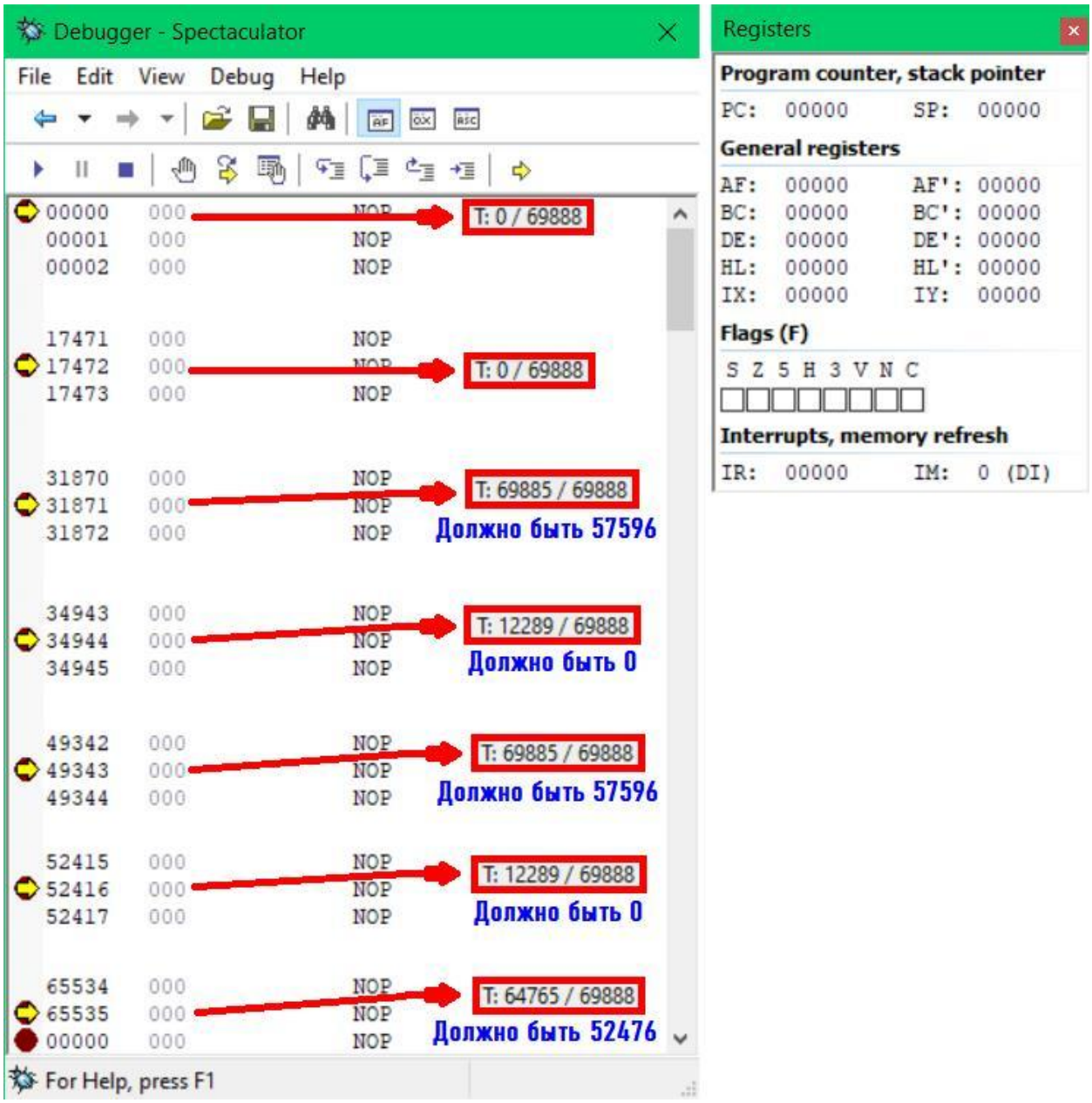


Рис. 559. Сбой времени при выполнении команд в режиме 3,5 МГц.

Как показали исследования дело не только в участках памяти, и в количестве ячеек, пройденных Стрелочкой подряд, но даже в режиме виртуального процессора. При частоте 3,5 MHz на прямом пути сбой начинается на ячейке 21057. Там вместо положенных 4 «НеСекунд» накидывается сразу девять. Следующие 14 шагов проход Стрелочки по пустым ячейкам будет совершаться за 8 «НеСекунд». Потом на некоторое время глюк пропадает, и шаг по ячейке памяти снова становится равным 4 «НеСекунды».

Далее, по пути к адресу 28712 снова периодически будут попадаться участки, с большим временем прохождения. В итоге за цикл прохода по памяти с 21057 по 28712 набегает балласт из 12289 «НеСекунд», что достаточно серьёзно. Как видно из рисунка с замерами, далее, до конца памяти снова всё идёт ровно и без накладок.

В итоге, вместо ожидаемых 262144, Стрелочка прошла память за 274433 «НеСекунды». А это всё те же 12289 единиц отставания, которые возникают в указанных проблемных местах. Как я уже писал, проблема не только в самих адресах. Если после обнуления ПЗУ, Стрелочку пустить не с 0, а например, с адреса 10, то область с некорректным выполнением команд пустоты также сдвинется вниз на 10 ячеек. Но всё равно, большая часть проблем будет наблюдаться в куске до 32767.

В режиме 7 МГц «Волшебный год» удваивается и становится 139776. На удивление, первый круг по памяти Стрелочка пройдёт по теоретическим расчётам. Ошибочные надбавки времени начнутся со 2-го круга, причём почти сразу. Так что, с большой вероятностью, это недоработка программы *Spectaculator*.

Глава 68
«Ввод и вывод» через IM 2 без поручика Ржевского

Краткое содержание: параллельные процессы, переход между режимами IM 1, IM 2, надпись-перевертыш, туалет

Есть в BASIC самоучителе такая глава, которая называется «Ввод и вывод». А еще я не рассмотрел свежим взглядом режим `IM 2`. Предлагаю совместить полезное с полезным, а также чуть подробнее разобраться в этом вопросе.

Режим `IM 2` это продвинутый вариант режима `IM 0` или `IM 1`. В нём можно самому выбрать адрес, на который по истечении «Волшебного года» должна перенестись Стрелочка. Точно также при переходе на заданный адрес произойдёт и принудительное переключение с `EI` на `DI`.

По закону подлости, в установке этого режима присутствуют элементы подвоха. Дело в том, что настроить адрес, на который должен будет осуществиться переход, можно с помощью другого адреса, а точнее пары ячеек. Вспомогательных адресов в памяти, которые можно приспособить для этих целей, всего 256 штук. Расположены они с интервалом в 256 байт по всей длине.

Почему так? Да потому что пару ячеек-посредников придумали, задавать старшим значением параметра «`IR`», а точнее регистром «`I`». Естественно, от этого и будет такой интервал. Кроме того, адрес складывается не из чистого значения «`I`», а с добавкой к нему некоторого числа. И этим числом является не «`R`», а значение со стороны.

По задумке, оно должно меняться в зависимости от подключенных устройств к реальному компьютеру. В реальных же компьютерах, этот добавочный параметр мог гулять и сам по себе, в зависимости от элементной базы и модификации. В древние времена это порождало множество споров, и дискуссий по поводу совместимости игр.

Именно поэтому в большинстве фирменных игр, которые использовали режим `IM 2`, нижние адреса драгоценной памяти приходилось забивать бессмысленным блоком из 256 одинаковых чисел для гарантированного перехода на нужный адрес. Как правило, массивы состояли из чисел 255, 254 или 253.

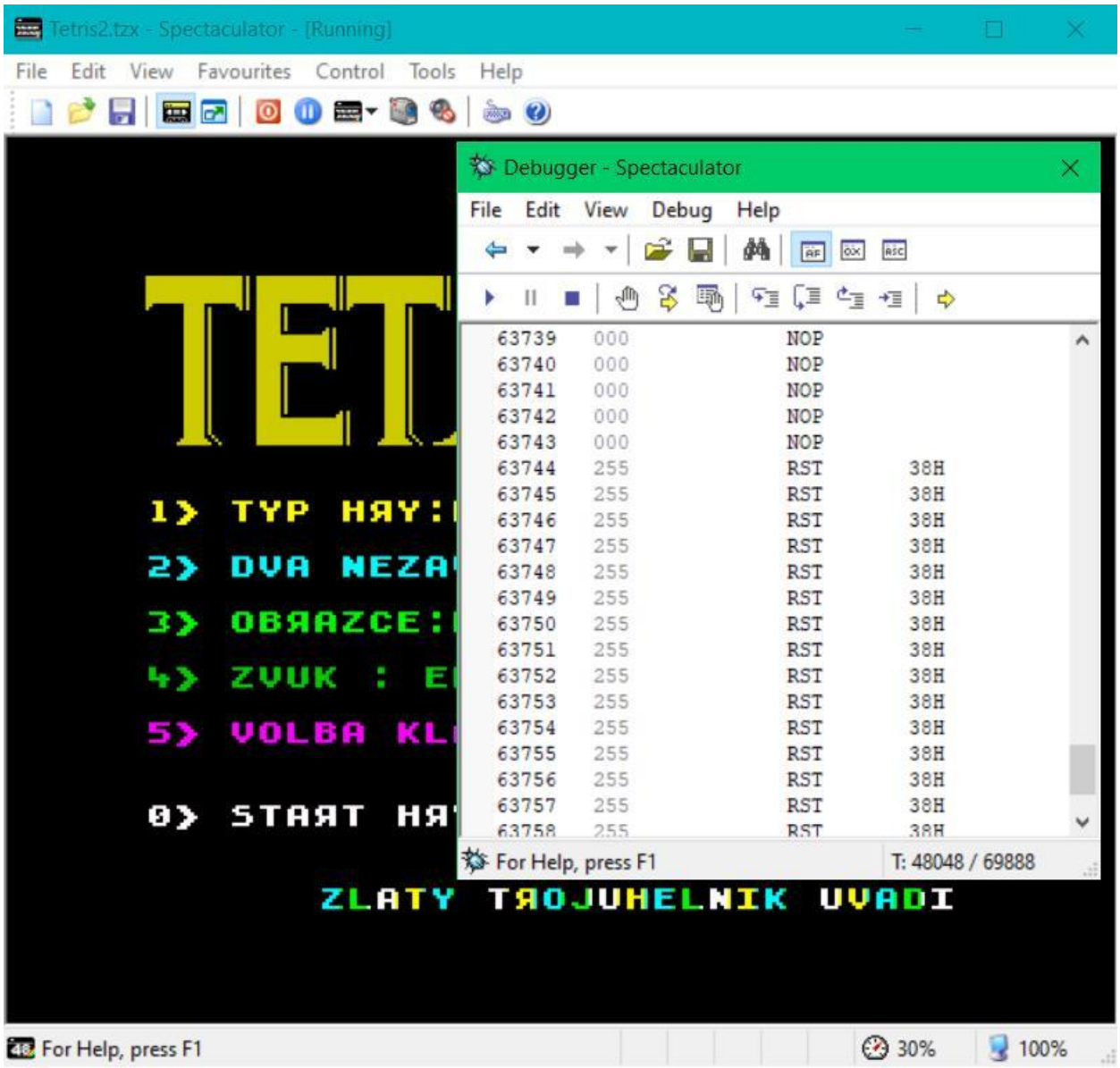


Рис. 560. ТЕТЯIS-2. Начало массива чисел «255» для указания адреса программы в паре адресов 65535-0.

У *Spectaculator'а* в обсуждаемой конфигурации добавочное значение стабильно и равно «255», так что в текущем случае, такой едрический числовой поезд не понадобится. Достаточно всего одного указательного адреса.

Естественно 256 комбинаций посреднических ячеек справедливо для полной памяти от 0 до 65535. Если отбросить ПЗУ, область экрана и карту системных переменных, первую треть можно смело забыть. С большой натяжкой первым пригодным адресом можно назвать 23807/08 и это при отсутствии BASIC программы, пара строк которой быстро уничтожит всё это хозяйство. Таким образом, на компьютере с полноценной BASIC системой, задавать «I» менее 93, не имеет практического смысла, разве что только считывать из ПЗУ случайно получившийся адрес. Такой подход применяется в игре *Panama Joe*, и описывается в книге «*ZX-Spectrum для пользователей и программистов*».

Например: I=93 (IR ← 23808), тогда ячейки-посредники для задания адреса будут 93*256+255=24063. Таким образом, в этих ячейках, нужно записать адрес начала изолированной программы. Пусть она разместится на три байта ниже, сразу за этой ячейкой. (24063 → 24066). В 24066 уже можно разместить саму программу, на которую будет периодически заходить Стрелочка, в конца своего «Волшебного года». Ну и самым необходимым условием в параллельной программе, конечно, будут команды `EI`, `RET` и

опционально PUSH / POP. Основную программу и вовсе можно разместить в любом свободном адресе без привязки к изолированной.

В книге 2013 года я приводил пример на работу в IM 2, который работал параллельно с BASIC, окрашивая экран квадратиками. Предлагаю вспомнить, о чём речь:

Глава 3.
ZX-DESKTOP, или ностальгия по windows.

Краткое содержание: параллельная работа BASIC и машинных кодов, режим прерываний IM2, заставка «рабочего стола».

Давайте создадим фоновый рисунок рабочего стола для спектрума, с возможностью подгрузки собственных картинок. При этом узоры на экране должны сохраняться во время параллельной работы программ на BASIC, включая нижние строки. При этом постараемся максимально автоматизировать процесс загрузки, и запуска, чтобы нашей задачей было набрать LOAD "" и больше не беспокоится. Потом останется только подгружать собственные картинки для рабочего стола.

Вспомним методы автозагрузки, рассматриваемые во 2-й части книги. Так как программа не длинная, ее можно запихать в нижнюю часть памяти, чтобы была возможность загружать в верхние адреса сразу несколько картинок. Дополнить программу можно автозагрузкой через машинный стек, с последующим его восстановлением, и работоспособностью интерпретатора BASIC. Это также освободит от лишних манипуляций с запуском «активатора» прерываний режима IM 2.

Откройте эмулятор EmulzWin, и первым делом наберите CLEAR 29999, потому что предстоит опасная работа со стеком. Далее откроем Ассемблер и введем (или скопируем через буфер обмена) следующую программу. После символа точка с запятой

Рис. 561. Фрагмент главы из книги «ZX-Spectrum-48K в эпоху windows и эмуляторов» за 2013 год.

«*Опасная работа со стеком*». Слова то какие! Конечно, опасно, когда не знаешь, как это работает и получаешь внезапный сброс. Скажу честно, я тогда не понимал, как параллельно заработал BASIC послу ухода в адрес 56. Сейчас посмотрел и чуть прибалдел. Ну, отпускать Стрелочку в ПЗУ с IM 2 не лучшее решение.

Предлагаю более чистый и прозрачный пример на эту тему. В основной программе будет опрос клавиатуры, то есть ввод. В изолированном куске будет лишь вывод изображения на экран. Также для разнообразия из режима IM 2 будет выход в BASIC с переключением в IM 1. Всё это чистый ассемблер, завёрнутый в алгоритм God Mode:

```
24063 ← 2 94 0 245 213 253 54 2 32 1 20 0
24075 ← 205 60 32 209 241 251 201 22 10 8 16 1
24087 ← НОС ВМЕСТО ХЕРА
24102 ← 22 10 8 16 2
24107 ← ХЕР ВМЕСТО НОСА
24130 ← 62 93 237 71 17 18 94 237 94 62 254
24141 ← 219 254 203 87 204 119 94 62 191 219 254
24152 ← 203 103 204 123 94 62 127 219 254
24161 ← 203 103 32 230 237 86 49 84 255
24170 ← 237 115 61 92 241 253 54 0 255 251
24180 ← 195 3 19 17 38 94 201 17 18 94 201
```

Запустите натуральным образом, набрав RANDOMIZE USR 24130. После запуска понажимайте клавиши «Н» (*от слова «Нос»*) и «Х» (*от противоположного слова*). На экране поочерёдно будет высвечиваться первая и последняя часть фразы-перевёртыша. Как говорится, что имеем, то и введём. Как в анекдоте про поручика Ржевского.

Нажав клавишу «B» произойдёт корректный *Выход* в BASIC с восстановлением режима `IM 1`.

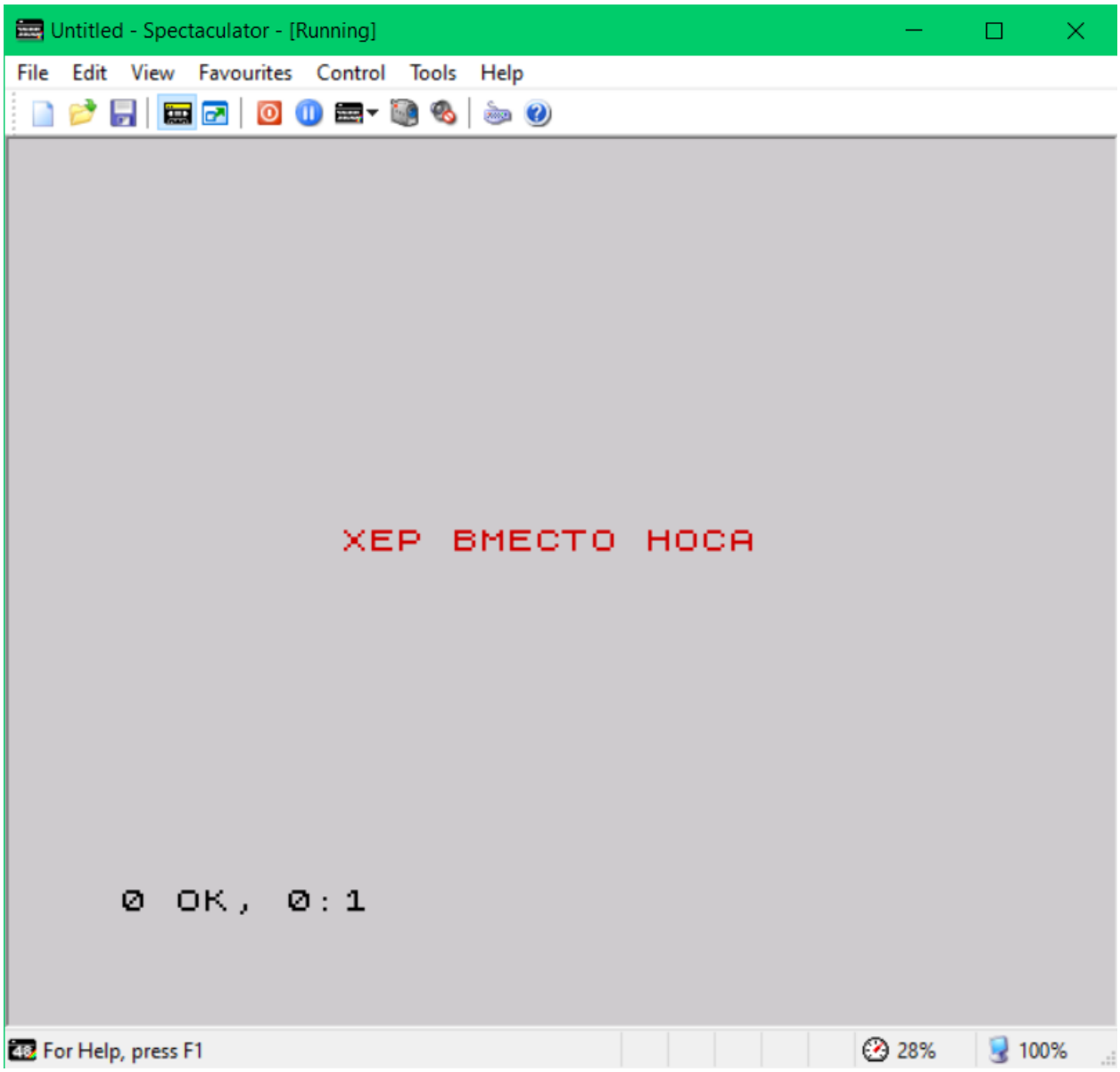


Рис. 562. Результат работы изолированной программы и выход в BASIC с IM 2.

Середина 2000-х годов. Золотая эпоха развития, модернизации и эффективного менеджмента. Для большей пропускной способности автомобилей, с улиц Петербурга убирают трамвайные пути, считая их бесполезными железками. Закрывают заводы и фабрики в историческом центре города. Их сносят под элитное жильё или перестраивают в бизнес-центры.

Не обошла участь и фабрику «Рабочий» на проспекте Обуховской Обороны. Весной 2007 года часть зданий активно готовят к реконструкции под офисы. Старый советский туалет доживает свои последние недели жизни. Его стены хранят целый культурный пласт «сортирного юмора» последних десятилетий…

Рис. 563. Надпись в туалете фабрики «Рабочий». Проспект Обуховской Обороны. 12 апреля 2007 года.

Как говорится, «*Каждому поэту по туалету*». Можно возразить, что поэтов в разы больше, чем туалетов. Я отвечу так: «*Это уже проблемы туалетов*».

Глава 69
Использование машинных кодов для циркового представления

Краткое содержание: ID BC 99 RET, SPEKTRUM, Асемблер, ГДР, СССР, таблицы, юмор, качество изданий, эпилог

Несмотря на то, что школьный учебный год, длится дольше существования Вселенной, он тоже имеет свойство заканчиваться. Всё в жизни когда-то заканчивается. Вот и эта книга подходит к своему логическому завершению. Впереди глава с подведением итогов, и зачётом для осмысления пройденного материала. Ведь в любом учебном заведении, окончанию учебного процесса предшествуют итоговые контрольные и экзамены.

Поначалу предэкзаменационная подготовка. Внимание на доску:

Краткое содержание: USR с числовым аргументом.
Эта глава описывает применение машинных команд микропроцессора Z80 [U880 (ГДР), 1810ВМ80 (СССР)].
Программы в машинных кодах пишут обычно на ассемблере с последующей трансляцией. (перечень мнемокодов команд микропроцессора Z80 приведен в приложении а). транслятор с ассемблера встроен в компьютер ZX SPECTRUM.

Эта глава описывает применение машинных команд микропроцессора Z80 (U880 (ГДР), 1810ВМ80 (СССР)).
Программы в машинных кодах пишут обычно на асемблере с последующей трансляцией, (перечень мнемокодов команд микропроцессора Z80 приведен в приложении А). Транслятор с асемблера встроен в компьютер ZX SPEKTRUM.

Рис. 564. Вырезки главы 26 «Использование машинных кодов» из разных книг.

...словно Берлинской стены километр триста, оставленной предками для туристов...

О как! Вот это я понимаю! Оказывается, транслятор с асемблера встроен в ZX-SPEKTRUM. Эх, а на моём ZX-Spectrum'е никакого транслятора не было или плохо искал. Хотя нет, телек, к которому всё это подключалось транслировал канал «Останкино». А вот

если бы у меня был «SPEKTRUM», да ещё с «Асемблером»! Ух как зажил бы бохато и щясливо!

Кстати, в «Дубновской» книжке писали, что нет никакого транслятора, так что мнения разделились. Кто был прав, узнаете чуть позже, а пока сохраняю интригу. Понятно одно: в мозгу у человека точно нет никакого встроенного транслятора ни с «Асеблера», ни с «Асемблера».

Ладно, в сторону лирику. Предлагаю посмотреть, как обстоят дела на самом деле:

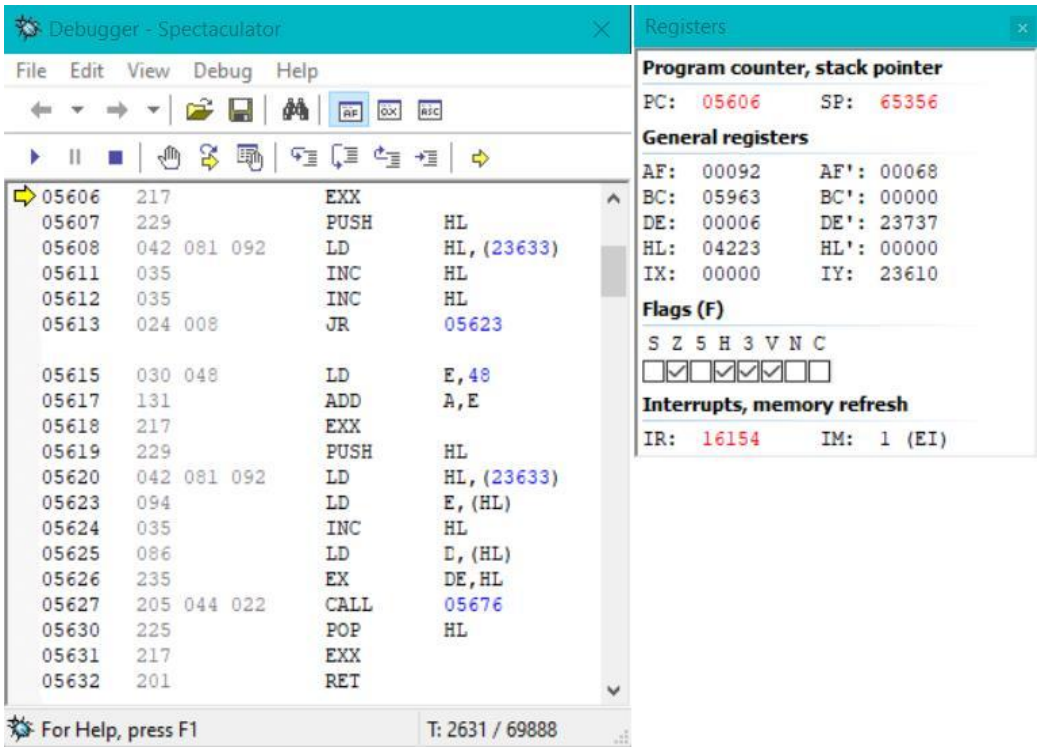


Рис. 565. Debugger с ассемблером и машинными кодами в произвольном месте.

Хотя… Ну как бы это и есть визуальный транслятор. Какие-то циферки вон виднеются и даже команды. Ладно, уговорили, пусть пока будет встроенный транслятор.

Попёрли дальше. Теперь задание. Ну-ка глянем, чего там прислали из РОНО в качестве контрольных заданий:

ID BC,99
RET

которая загружает в "BC" регистр число 99, эта программа будет транслироваться в 4-х байтный машинный код:
байты 1,99,0 для ID BC,99 и 201 для RET.

Рис. 566. Легендарная 4-х байтная программа.

Блин, а где такой ID-шник взять. Свой что ли давать? Короче, если не поняли, ща будет наводящая подсказка! Это фирменная команда компьютера «ZX-SPEKTRUM»!

Следующим шагом является загрузка программы в компьютер, для этого используется дополнительная память, получаемая между бейсик- областью и областью определяемых пользователем символов. Допустим, вы имели следующее распределение последней части ОЗУ:

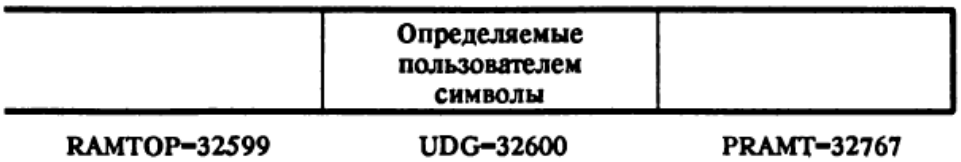


Рис. 567. Распределение памяти компьютера модели «16К» бездумно скопированное в книги.

Для загрузки рекомендуется использовать обе руки и желательно, чтобы ладошки без мозолей… от МЫШКИ, а не то о чём можно подумать. Допустим, а почему такое странное распределение памяти? UDG в 32600. Куда зажали ещё 32768 памяти? А всё понятно, сейчас дальше всё объяснят, куда она подевалась.

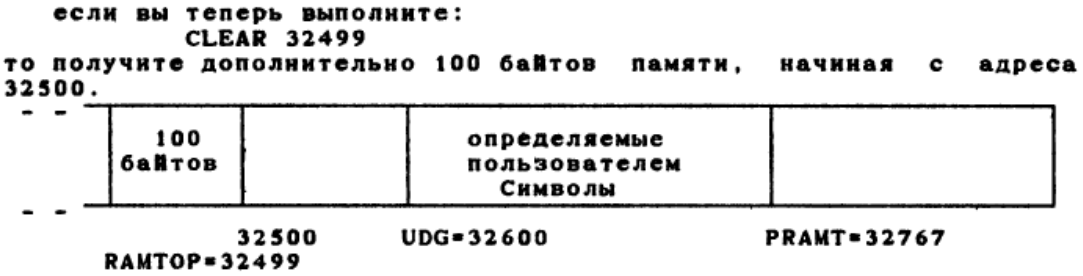


Рис. 568. Получение дополнительных 100 байтов в районной жилконторе.

Э-э-э! Я же набрал `CLEAR 32499`, почему мне не выделили дополнительно 100 байт гуманитарной помощи в виде микросхем? А ещё… А ещё… немецкие консервированные сосиски зажали. Вот! И вообще, почему нижний кусок памяти так и не вернулся? Где адрес 65535, мать его ети???

```
Для загрузки программы в машнных кодах вы можете выполнить
следующую бейсик-программу:

          10 LET A=32500
          20 READ N: POKE A, N
          30 LET A=A+1: GOTO 20
          40 DATA 1,99,0,201
(программа может завершиться с сообщением 'E OUT OF DATA',
если переполняются отведенные вами 4 байта).
```

Рис. 569. BASIC программа для загрузки легендарного Ассемблера.

А, ёп…! Так вот же тот самый встроенный транслятор с «Асемблера» в «BASIKA». И вообще, есть такое предчувствие, что программа гарантированно завершится этим сообщением и даже без слова «может».

Для выполнения загруженных машинных кодов используется функция USR, но с числовым аргументом, определяющим начальный адрес.
Если Вы выполните:

```
PRINT USR 32500
```

то получите ответ: 99.

Рис. 570. Наглухо загруженные и обсаженные машинные коды дают ответ «99».

Да как же их выполнить, если двумя заданиями ранее их загрузили по полной, не выслушав ни единого числового аргумента! Кажется, уже поздно пить все минеральные воды Боржоми. В нынешнем состоянии, даже числовой аргумент гарантированно не станет контраргументом. Коды уже не только выполняться, а до начального адреса кровати на карачках доползти не могут.

А ведь от этих кодов нужно еще получить ответ. Не «99», а нормальный ответ текстом. Звёзд с неба не требуется. Вполне достаточно формальной отписки формата РЭУ/жилконторы «*…на ваше обращение, разъясняем…*». Короче, проспятся машинные коды после загрузки и утром обязательно дадут какой-нибудь бюрократический ответ.

```
Возврат в бейсик-программу осуществляется обычным образом по
комаиде микропроцессора RET. В машинной программе вы не должны
использовать регистры IY и I.
```

Рис. 571. Обычный повседневный и непримечательный возврат в BASIC.

Знаете, на автопилоте по команде `RET`, любой человек обычно возвращается домой с самых первых дней своей жизни. Это настолько обыденно и повседневно, что даже

упоминать не стоило. Зря только краску типографскую и бумагу тратили. Вон, новорожденного можно спросить, он и то такие банальные вещи знает.

Оказывается IY и I нельзя использовать. Во как! Трудно представить, если бы тот, кто изначально писал эту главу, увидел примеры с изменённым SP.

Вы можете записать вашу программу на ленту:
SAVE "NAME" CODE 32500,4
Можно записать эту программу и так, что она будет автоматиче- ски выполняется после загрузки:
10 LOAD " " CODE 32500,4
20 PRINT USR 32500
Для чего надо сделать:
SAVE NAME LINE, а затем:
SAVE " XXXX" CODE 32500,4
LOAD "NAME"

Рис. 572. Схема автостарта «Секрет Полишинеля». Скан из BASIC самоучителя.

Итак, перед вами новый эксклюзивный автостарт, который имеет официальное название «*Секрет Полишинеля*». Данная сложная конструкция требует пояснений.

«*NAME LINE*» – это специальная BASIC команда, программа которой располагается в неизведанных уголках ПЗУ по отрицательному адресу. Она сканирует человеческий мозг, еще до момента набора команды `LOAD ""` позволяя сохранить и запустить данные на компьютере до первых прикосновений пальцев к клавиатуре.

А вот что такое «XXXX» пока так и осталось тайной. Уже даже об «Порте FF» и игре ELITE с её гребучей тайной планетой давно позабыли, а об этом «XXXX» дискуссии не утихают по сей день.

Дело в том, что X это единица измерения экипировки человека, а именно:

X – тело в костюме химзащиты или космическом скафандре.
XX – тело в простой повседневной одежде (треники, вытянутая майка, потники, и пр.)
XXX – тело с полным отсутствием одежды (обычно женское, но бывают исключения)
XXXX– вид изнутри человека на окружающий мир (из какого места не регламентировано).

Таким образом, у всемирной ассоциации программистов нет единого мнения, что может обозначать «XXXX» с командой «`SAVE`». Пока самой правдивой гипотезой остаётся теория о том, что этот 4-х байтный массив был создан в секретных лабораториях для отправки инопланетянам.

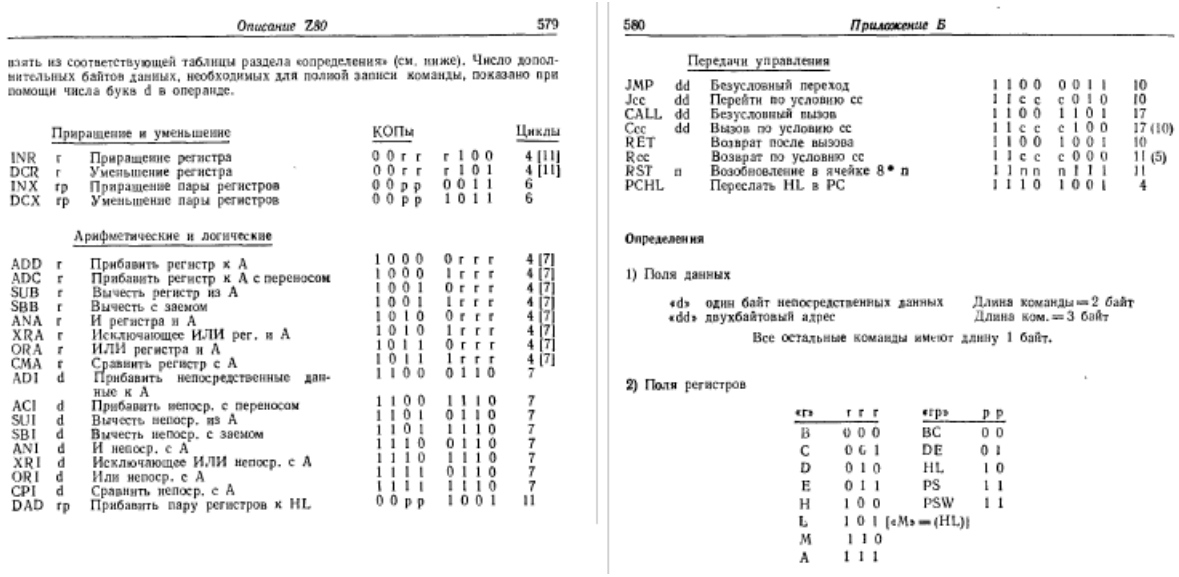
А теперь о том, какие могут произойти, если данная шифровка всё таки долетит до адресата.

```
   Это приведет к тому, что вначале будет загружена и автомати-
чески выполнена бейсик-программа, которая, в свою очередь, загру-
зит и выполнит программу в машинных кодах. Книга "Искусство схе-
мотехники" П. Хоровиц, У. Хилл Мир,1986, том 2, стр.579-58 0.
   Далее приводятся 78 команд микропроцессора 8085, совместимых с
микропроцессором Z80 (158 команд). /U880-ГДР, К1810ВМ-СССР/
```

Рис. 573. Последствия отправки легендарной программы инопланетянам.

Ну вот и ответ! Загрузка и выполнение тайного шифра «1,99,0,201» приведёт к тому, что инопланетяне организуют нападение на Землю с целью выкрасть сверхсекретную книгу «Искусство Схемотехники» том 2 от издательства «Мир» 1986 года, чтобы посмотреть, что написано на страницах 579-580, стих 1, абзац 99, слова с 0 по 201-е.

Пока инопланетяне не похитили последние экземпляры книги, вам представляется уникальная возможность, глянуть в это элитное сверхсекретное издание.



Описание Z80 579

взять из соответствующей таблицы раздела «определения» (см. ниже). Число дополнительных байтов данных, необходимых для полной записи команды, показано при помощи числа букв d в операнде.

		Приращение и уменьшение	КОПы	Циклы
INR	r	Приращение регистра	00rr r100	4 [11]
DCR	r	Уменьшение регистра	00rr r101	4 [11]
INX	rp	Приращение пары регистров	00pp 0011	6
DCX	rp	Уменьшение пары регистров	00pp 1011	6

Арифметические и логические

ADD	r	Прибавить регистр к A	1000 0rrr	4 [7]
ADC	r	Прибавить регистр к A с переносом	1000 1rrr	4 [7]
SUB	r	Вычесть регистр из A	1001 0rrr	4 [7]
SBB	r	Вычесть с заемом	1001 1rrr	4 [7]
ANA	r	И регистра и A	1010 0rrr	4 [7]
XRA	r	Исключающее ИЛИ рег. и A	1010 1rrr	4 [7]
ORA	r	ИЛИ регистра и A	1011 0rrr	4 [7]
CMA	r	Сравнить регистр с A	1011 1rrr	4 [7]
ADI	d	Прибавить непосредственные данные к A	1100 0110	7
ACI	d	Прибавить непоср. с переносом	1100 1110	7
SUI	d	Вычесть непоср. из A	1101 0110	7
SBI	d	Вычесть непоср. с заемом	1101 1110	7
ANI	d	И непоср. с A	1110 0110	7
XRI	d	Исключающее ИЛИ непоср. с A	1110 1110	7
ORI	d	Или непоср. с A	1111 0110	7
CPI	d	Сравнить непоср. с A	1111 1110	7
DAD	rp	Прибавить пару регистров к HL	00pp 1001	11

580 Приложение Б

Передачи управления

JMP	dd	Безусловный переход	1100 0011	10
Jcc	dd	Перейти по условию cc	11cc c010	10
CALL	dd	Безусловный вызов	1100 1101	17
Ccc	dd	Вызов по условию cc	11cc c100	17 (10)
RET		Возврат после вызова	1100 1001	10
Rcc		Возврат по условию cc	11cc c000	11 (5)
RST	n	Возобновление в ячейке 8* n	11nn n111	11
PCHL		Переслать HL в PC	1110 1001	4

Определения

1) Поля данных

«d» один байт непосредственных данных — Длина команды = 2 байт
«dd» двухбайтовый адрес — Длина ком. = 3 байт

Все остальные команды имеют длину 1 байт.

2) Поля регистров

«r»	r r r	«rp»	p p
B	0 0 0	BC	0 0
C	0 0 1	DE	0 1
D	0 1 0	HL	1 0
E	0 1 1	PS	1 1
H	1 0 0	PSW	1 1
L	1 0 1	[«M» = (HL)]	
M	1 1 0		
A	1 1 1		

Рис. 574. Фрагменты 579-580 страниц книги-легенды «Искусство Схемотехники» 1986 года.

А там… ничего полезного и этой самой мифической главы, которая раскроет тайну программирования в машинных кодах. Сплошное разочарование, как в сказке «*Про олдовых людей*». Одни схемы, и групповые списки команд, которые никаким боком не относится к стандартной модели *ZX-Spectrum*.

Но предлагаю вернуться в самоучители по BASIC. Следующим абзацем за упоминанием книги «*Искусство схемотехники*», обычно идёт следующее:

Далее приводятся 78 команд микропроцессора 8085, совместимых с микропроцессором Z80 (158 команд). /U880-ГДР, К1810ВМ-СССР/

Мнемоника	Действие	КОП	Циклы
1	2	3	4
пересылка, загрузка, запись			
MOV R,R	ПЕРЕСЛАТЬ РЕГИСТР В РЕГИСТР	01RR RRRR	4[7]
MVI R,D	ПЕРЕСЛАТЬ НЕПОСР.В РЕГИСТР	00RR R110	7[10]
LXI RP,DD	ЗАГРУЗИТЬ НЕПОСР.В ДВА РЕГ.	00PP 0001	10
STAX B	ЗАПОМНИТЬ A КОСВЕННО ПО BC	0000 0010	7
STAX D	ЗАПОМНИТЬ A КОСВЕННО ПО DE	0001 0010	7
LDAX B	ЗАГРУЗИТЬ A КОСВЕННО ПО BC	0000 1010	7
LDAX D	ЗАГРУЗИТЬ A КОСВЕННО ПО DE	0001 1010	7
STA DD	ЗАПОМНИТЬ A ПО АДРЕСУ DD	0011 0010	13
LDA DD	ЗАГРУЗИТЬ A ПО АДРЕСУ DD	0011 1010	13
SHLD DD	ЗАПОМНИТЬ H,L ПО АДРЕСУ DD	0010 0010	16
LHLD DD	ЗАГРУЗИТЬ H,L ПО АДРЕСУ DD	0010 1010	16
XCHG	ОБМЕНЯТЬ DE И HL	1110 1011	4
ПРИРАЩЕНИЕ И УМЕНЬШЕНИЕ			
INR R	ПРИРАЩЕНИЕ РЕГИСТРА	00RR R100	4[11]
DCR R	УМЕНЬШЕНИЕ РЕГИСТРА	00RR R101	4[11]
INX RP	ПРИРАЩЕНИЕ ПАРЫ РЕГИСТРОВ	00PP 0011	6

Рис. 575. Таблица команд процессора 8005. Скан из BASIC самоучителя.

Вот так, назло всем завистникам! Внезапно возьмём и приведём команды для микропроцессора 8085, которые совместимы с *ZX-Spectrum*. Конечно, после прочтения первых 25-ти глав самоучителя BASIC, любой читатель в совершенстве владеет данными командами и виртуозно применяет их в повседневной жизни. Никто в этом не сомневается. Но, кажется, суть этого абзаца еще раз напомнить марки процессоров и названия стран, которые их выпускают.

Теперь следующий вопрос: нахуа эти команды тут, если далее в приложении «*Полный Набор Символов*» приводятся нормальные команды для Z80? Эта таблица просто

вырвана из контекста книги по микроэлектронике. Что-то мне подсказывает, что креативщик, который дорабатывал эту главу и сам не знает, зачем оно тут. Просто во время формирования русскоязычной версии, автор пыхнул пару раз за работой и случайно открыл не ту книгу. Увидев море мудрой информации, он решил малость разнообразить содержимое оригинала 1983 года.

Ну а если копнуть дальше, то в общих чертах, эта глава действительно присутствовала в оригинале самоучителя Steven'a Vickers'a «*ZX-Spectrum BASIC Programming*». Она располагалась на страницах 131-132 и называлась «*Chapter 26 Using Machine Code*». Вот её первая страница на языке оригинала.

CHAPTER 26
Using machine code

Summary
USR with numeric argument

This chapter is written for those who understand *Z80 machine code*, the set of instructions that the Z80 processor chip uses. If you do not, but would like to, there are plenty of books about it. You want to get one called something along the lines of 'Z80 Machine code (or assembly language) for the absolute beginner', and if it mentions the Spectrum, so much the better.

These programs are normally written in *assembly language*, which, although cryptic, are not too difficult to understand with practice. (You can see the assembly language instructions in Appendix A.) However, to run them on the computer you need to code the program into a sequence of bytes – in this form it is called *machine code*. This translation is usually done by the computer itself, using a program called an *assembler*. There is no assembler built in to the Spectrum, but you may well be able to buy one on cassette. Failing that, you will have to do the translation yourself, provided that the program is not too long.

Let's take as an example the program

```
ld bc, 99
ret
```

which loads the bc register pair with 99. This translates into the four machine code bytes 1, 99, 0 (for ld bc, 99) and 201 (for ret). (If you look up 1 and 201 in Appendix A, you will find ld bc, NN – where NN stands for any two-byte number – and ret.)

When you have got your machine code program, the next step is to get it into the computer. (An assembler would probably do this automatically.) You need to decide whereabouts in memory to put it, and the best thing is to make extra space for it between the BASIC area and the user-defined graphics.

Suppose, for instance, that you have a 16K Spectrum. To start off with, the top end of RAM has

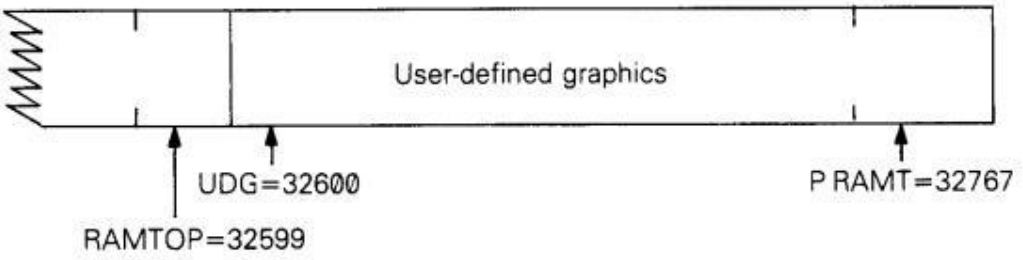


Рис. 576. Фрагмент главы 26 «Использование машинных кодов» из первоисточника.

Предэкзаменационная подготовка закончена, а теперь начинается экзамен. Пожалуйста, уберите все шпаргалки и внимание на доску.

1) И всё-таки, существует встроенный транслятор с Ассемблера?
2) Сколько упоминаний о ГДР и СССР видно на оригинале 1983 года?
3) Почему во всех копиях перевода регистр стал ID?

Время пошло…

```
ПОЗДРАВЛЯЮВСКЛ ВЫ УСПЕШНО СДАЛИ ЭКЗАМЕН И ПРОШЛИ ОБУЧЕНИЕТЧК
```

Всё правильно. Юмористические дополнения в виде «ГэДээРаСэСэСэРа», чудодейственного транслятора с языка Ассемблер и таблиц добавились после выхода перевода на русский язык в 1987 году. Их никогда не было в оригинале.

Микро эпилог

В начале 1990-х от перевода этой 26-й главы меня бомбило больше всего. Конечно, тогда такого термина не существовало, но состояние было. Будучи пацаном 12-14 лет, я не понимал, почему показан такая короткая программа. Что мешало в качестве примера вывести на экран слово «Тест» или хоть цветную буковку «А». Да пусть даже цвет рамки или экрана. Но самое главное, как применить эту таблицу с такими странными командами? Я не мог понять, как знание загрузить число в ячейку даст мне просветление, после которого я сразу научусь выводить графику и создавать игры по команде `RANDOMIZE USR`. В общем, загрузка каких-то чисел в какой-то регистр казался бредом.

Как показало время, я не сильно ошибался. Качество переведённой литературы просто на высоте. Не удивительно, что имея под рукой лишь такие книги, в эпоху реального *ZX-Spectrum'a*, я так и не смог нормально освоить программирование на ассемблере. Ну а книжка «*Машинные Коды*», купленная в ноябре 1996-го, за пару месяцев до окончательного выхода из строя компьютера, стала прощальным символом с яркой, но не долгой эпохой Спектрума. Даже сейчас, несмотря на приколы, я люблю, и буду любить этот замечательный компьютер с его архитектурой. Иначе, не было бы этой книги.

ЧАСТЬ IV. ЖЕЛТАЯ ТЕХНИЧЕСКАЯ

Как понятно из названия, в эту короткую часть войдут небанальные вспомогательные материалы, а также пара моментов из истории создания данной книги.

ПРИЛОЖЕНИЕ К
Описание приспособленческой русификации

Если почитать совестскую-российскую компьютерную литературу ~1989-1997 годов, то в глаза сразу бросаются две больные темы, которые периодически всплывали на страницах изданий: «*Порт FF*» и «*Русификация компьютера ZX-Spectrum*». Размышления об игрушке ELITE не в счёт. Из-за того, что самодельные, и вообще реальные компьютеры, давно ушли в историю, первая тема совсем потеряла актуальность. Вторая тема, на страницах этой книги, получила второе дыхание.

На начальном этапе формирования этого самоучителя я не раз сталкивался с желанием делать примеры на русском языке. Не стоит искать никаких политических подтекстов, всё гораздо проще. Я посмотрел, как коряво выглядят примеры с транслитерацией в своей старой книге 2013 года и решил, что повествование с программами должны быть на одном языке. Захотелось подойти к вопросу серьёзно, но с креативом.

Метод графических пользовательских символов, замена шрифта и даже навороченные программы с прерыванием и переключением раскладки по IBM методике не вселяли оптимизма. Ну не то это всё!

Решение родилось спонтанно уже во время работы над первой частью книги в июле 2024 года. Накатила ностальгия. Я вспомнил лето 2001 и 2002 года на даче в Ленинградской области. Интернета там естественно не было. Мобильные телефоны были массивные с антеннами. С черно-белого пиксельного экрана можно было отправлять только SMS сообщения латинскими буквами. Других шрифтов не было. Мало того, чтобы поймать связь с далёкой вышки, нужно было дойти до самого высокого места в посёлке и при отправке сообщения некоторое время махать телефоном.

Одним из первых тестовых русифицированных сообщений стала фраза «A B COCHOBO MHE XEPOBO», которую брат написал большими латинскими буквами и успешно отправил с дачи в Петербург с одного мобильника на другой.

Связь тогда была дорогая. Сигнал за городом ловился не везде, поэтому пара телефонов в семье была общественная. Использовались они, наподобие рации для связи «Дача-квартира», отправляя сообщения раз в день с важными новостями (какие продукты с вещами довезти в выходной, узнать прогноз погоды и т д.).

Уплывая еще дальше, в юность и детство, внезапно вспомнилось давно забытое слово – «Эрудит» и всё окончательно встало на свои места. Кто не помнит, в конце восьмидесятых была такая настольная игра:

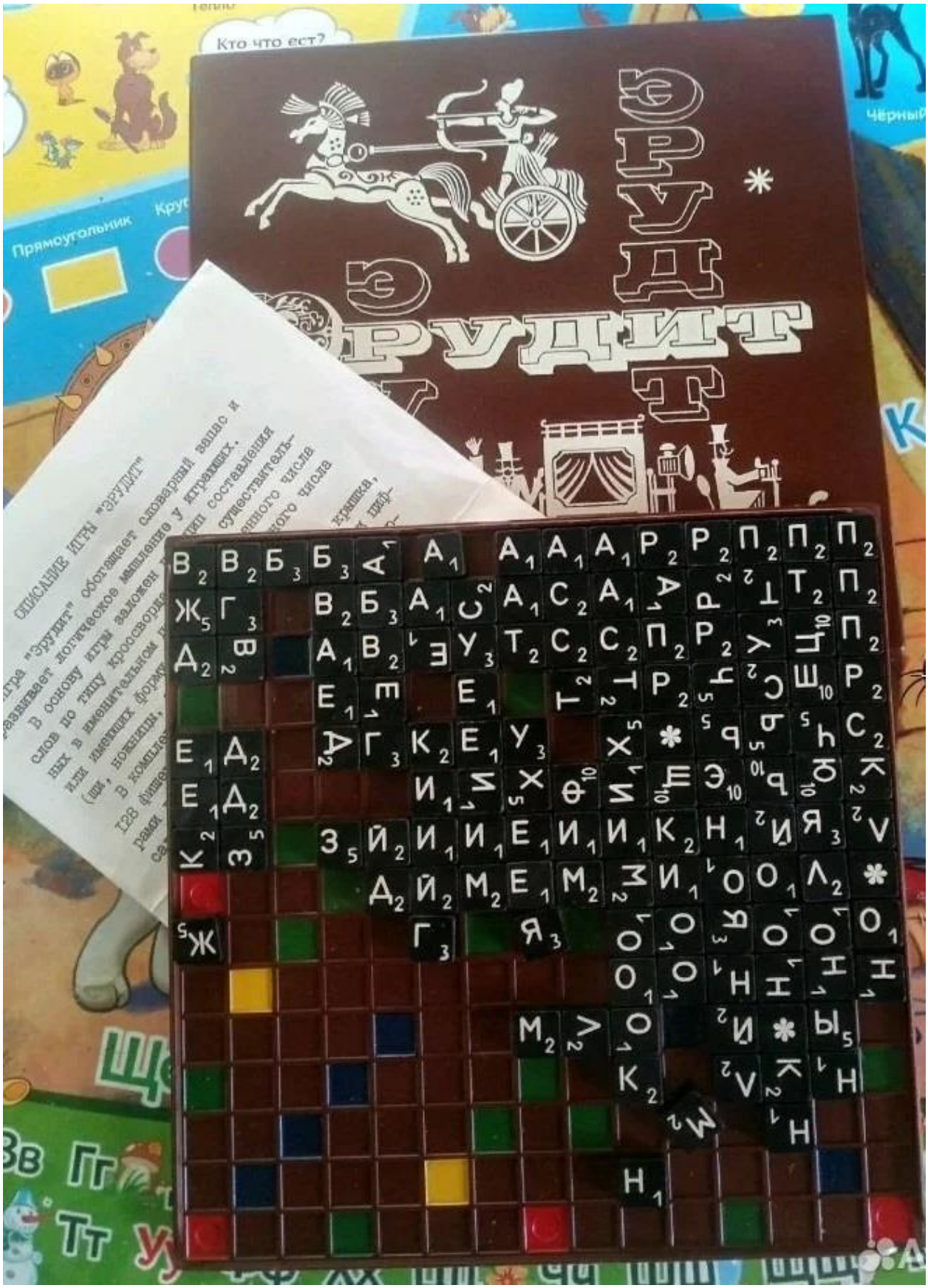


Рис. 577. Настольная игра «Эрудит». Общий вид. Фото из интернета.

В этой логической игре, из доставшихся букв нужно придумывать слова и выкладывать по методу кроссворда. Суть в том, что в некоторый момент времени у человека ограниченный запас ассортимента букв, и нужно хорошо напрячь мозги, чтобы выдать что-то оригинальное.

И тут я понял, что попытка использовать латинский алфавит для тематических примеров, да еще в качестве кириллицы, это тоже своего рода, «Эрудит». Может возникнуть вопрос: а зачем, если можно подгрузить набор и настроить указатель CHARS?

Да просто для прикола. Ведь когда запираешь себя в рамки ограниченного словарного запаса, приходится проявлять чудеса изобретательности, чтобы написать что-то читаемое, и самое главное, осмысленное. Порой такие фразы приходят на ум, что самого на ржач пробирает. В этом и основная фишка метода «Эрудита».

Кириллические слова, набранные латиницей, получили названия «Интернациональные». Самые популярные интернациональные слова были известны еще с незапамятных времён. Два из таких «СССР» и «МОСКВА» стали названиями музыкальных групп:

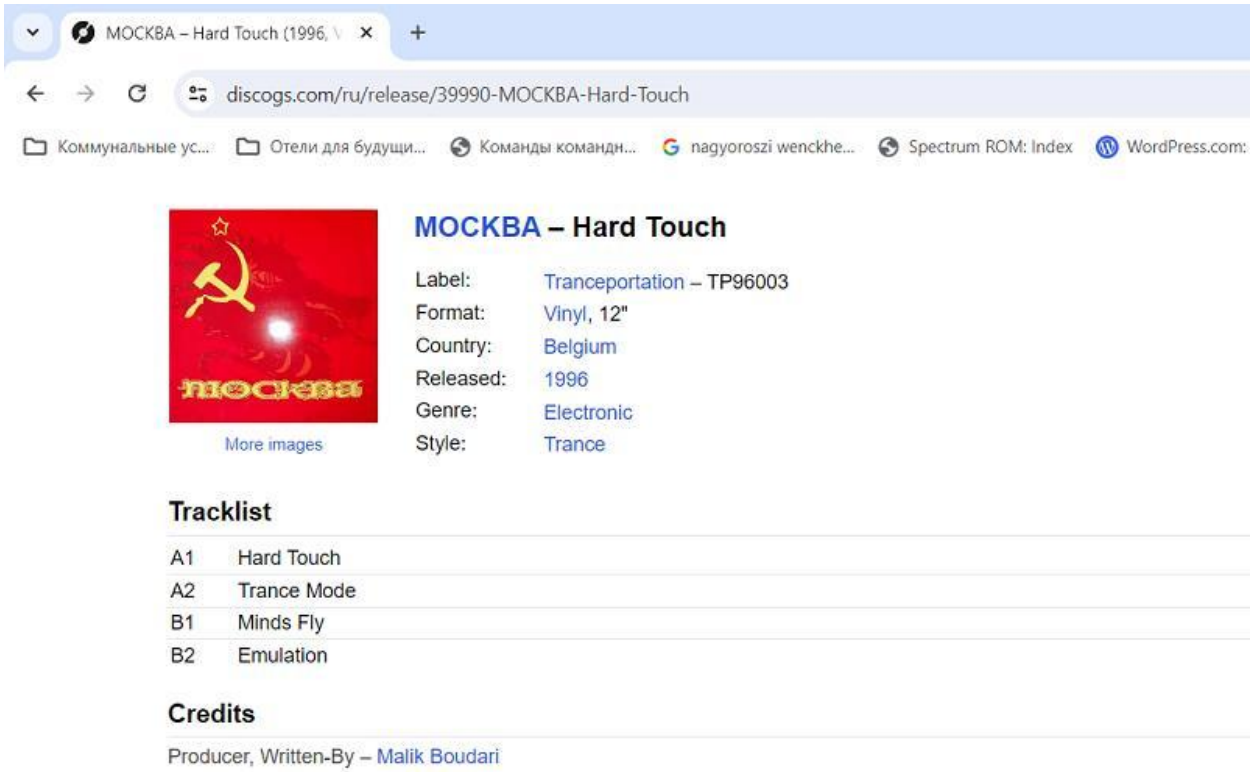


Рис. 578. Виниловая пластинка музыкальной группы «МОСКВА».

Параллельно с созданием книги велся сбор слов в базу, вырисовывалась терминология и структуризация. Появилось понятие «Коэффициент эстетичности». В конечном итоге, получилось следующее:

Сорт первый. Слова, состоящие из заглавных латинских букв, которые полностью идентичны кириллическим. Таких знаков в *ZX-Spectrum* 12 штук: ABCEHKMOPTX и число 3. Эти слова имеют самую высокую классификацию.

На подобном принципе в 1993 году были созданы российские автомобильные номера (ГОСТ Р 50577-93). На улицах Москвы и Петербурга они начали появляться примерно в октябре 1993 года. В автомобильных номерах вместо числа «3» ввели букву «у», которая по задумке должна была восприниматься как строчный «у».

Рис. 579. Буква «у» на автомобильном номерном знаке. Солнечное-2. Фото 12 июня 1997 года.

Сорт второй. Строчные латинские буквы, которые в интерпретации *ZX-Spectrum* имеют схожесть с кириллицей от 90 до 100%. Этих букв 14 штук: `abcegkmnoprtuxy`. Слова, которые собираются чисто из строчных букв, относятся к этому сорту. Например: `гопома, психопат.`

К третьему сорту относятся слова, которые можно написать с заглавной буквы. Например: `Сорт`.

Все остальные сорта будут с меньшим коэффициентом эстетичности и в моей книге не используются, поэтому подробно останавливаться не имеет смысла. Например, слова четвертого сорта являются такими, которые возможно написать латинскими буквами разного регистра. Например: `КОпРОТЕРАпЕВТ, пургеН.`

Пятый сорт слов вообще малоиспользуемый. В него входят слова, в которые включены латинские буквы, с коэффициентом похожести на кириллические около 50%. К символам 5-го сорта можно причислить буквы `R, N, w` и число `4`. Выглядят слова и фразы с такими знаками крайне отстойно, но по общему контексту воспринимаются. Например: `KRACHAR wAnO4KA`. Короче, в эту категорию входит любой шлак из символов.

Четвёрка дизайна в виде заглавной буквы «Ч», могла бы серьёзно расширить ассортимент слов и фраз (*например, очко*) но, к сожалению, в *ZX-Spectrum* подобное написание не используется.

Фразы, составленные из похожих по написанию букв, также делятся на четыре условные категории по коэффициенту эстетичности:

– ФРАЗА, НАПИСАННАЯ ПОЛНОСТЬЮ ЗАГЛАВНЫМИ БУКВАМИ
– Каждое Слово В Предложении Начинается С Заглавной Буквы
(или Первое слово в предложении с заглавной буквы)
– предложение полностью состоит из строчных букв
– КОМБИНИРОВАННАЯ строка из Слов трёх ПРЕДЫДУЩИХ Категорий

Конечно, последняя категория на практике наименее предпочтительна.

В заключение приложения, топ 50 эстетически удавшихся первосортных фраз на кириллице, написанных латинскими буквами:

01 А В СОСНОВО МНЕ ХЕРОВО
02 АННА АРЕСТОВАНА ЗА ОТКАТ
03 В КАНАВЕ КТО-ТО КВАКАЕТ
04 В КОМАРОВО НА ОЗЕРЕ ОХРЕНЕННО
05 В КОМНАТЕ АНТОНА ЗРЕЕТ ТРАВКА
06 В КОСТРОМЕ НА РАССВЕТЕ КРАСОТА
07 В КРАСНОЗНАМЕНСКЕ РОЗОВОЕ ЗАРЕВО
08 В НЕВЕ ОТ ВЕТРА МОТАЕТ МЕТЕОР
09 В НЕВСКОМ ВОЕНКОМАТЕ СТРЕМНОВАТО
10 В СНТ ОРЕХОВО ОТ ВЕТРА НЕТ СВЕТА
11 В РОСТОВЕ КОРОНОВАН ВОР В ЗАКОНЕ
12 В СОРРЕНТО НЕТ ТОРРЕНТОВ
13 В РЕМЗОНЕ НА СТО ЗАМЕНЕН МОТОР
14 В ТАРАСОВО НА ОЗЕРЕ ТАРЗАНКА
15 ВАРВАРА ЕСТ МАРС В МЕТРО
16 ВАХТЕР ЗАКРОЕТ ВОРОТА НА ЗАМОК
17 ВЕТЕР ВОЕТ В СТАРОМ ОКНЕ
18 КОСМОНАВТ В КАСКЕ МАХАЕТ АВТОМАТОМ
19 КОТОВСКОЕ ОЗЕРО ЗАРАСТАЕТ МХОМ
20 КРАН ЗАСТРЕВАЕТ В АРКЕ МОСТА
21 КРАТЕР В КРАТОВО РАЗМЕРОМ С АНАНАС
22 МАКОВОЕ СНОТВОРНОЕ НАВЕРНОЕ ТОРКНЕТ
23 МАТРЕНА МАТВЕЕВНА ЗОВЕТ АНТОНА НА СЕНОКОС
24 МАТРОС НА АВРОРЕ МОЕТ ОТСЕК
25 МОНТЕР МОССВЕТА НЕСЕТ МОТОК ТРОСА
26 НА АНТОНОВА-ОВСЕЕНКО ОМОН ЗАМЕТАЕТ НАРКОМАНОВ
27 НА КОММЕРСАНТОВ СНОВА НАСТАЕТ ОХОТА
28 НА ОХТЕ ЗАСРАНА ОСТАНОВКА ТРАМВАЕВ
29 НА СЕВЕРО-ВОСТОКЕ ОМСКА СНОВА МОРОЗНО
30 НА СТЕНЕ РАМКА В КРАСНОМ ОТТЕНКЕ
31 НЕВЕСТА СТРАСТНО СТОНЕТ В КАРЕТЕ
32 ОКСАНА НАКАТАЕТ РАССКАЗ О САМАРСКОМ КЕНТЕ
33 ОТ ТОСНО К МОСКВЕ СОСТАВ НЕСЕТ РОМАНА
34 ОТМОРОЗОК РАЗНЕС РЕСТОРАН В АВТОВО
35 ОХРАНА ХВАТАЕТ МАКСА НА КАССЕ ЗА ВОРОВСТВО ТОРТА
36 РАКЕТА НЕСЕТ АСТРОНАВТА В КОСМОС
37 РЕЕК В ХОЗТОВАРАХ ХВАТАЕТ НА ВСЕХ
38 РЕКА ВТЕКАЕТ В АСТРАХАНСКОЕ МОРЕ
39 РОЗА В КОМАТОЗЕ ОТ НАРКОЗА
40 САКЕР НЕ СТАНЕТ ХАКЕРОМ
41 САНЕК МОТАЕТ СРОК ЗА ВОРОВСТВО
42 СВЕТА К ВЕСНЕ НАСОСЕТ НА САМОКАТ
43 СТАС РАССЕКАЕТ НА МЕРСЕ В ОЗЕРКАХ
44 ТАКСА КАКАЕТ В СКВЕРЕ НА ТРАВЕ
45 ТВЕРСКОЕ МОРЕ РАЗМЕРОМ С ОКЕАН
46 ТОМА ТОМНО ЗЕВАЕТ НА ТАХТЕ
47 ТРАКТОР В СОВХОЗЕ ТОНЕТ В НАВОЗЕ
48 ХЕР ВМЕСТО НОСА - НОС ВМЕСТО ХЕРА
49 ХОХМА ХАЗАНОВА РАЗОРВЕТ ОТ СМЕХА
50 ХОР КАСТРАТОВ ЗАВОЕТ НА АРЕНЕ КРАСНОКАМЕНСКА

ПРИЛОЖЕНИЕ С
Команды языка God Mode

Версия: 3.0 от 28 августа 2025 года.

Краткое описание

Данный алгоритмический язык родился спонтанно при попытке создать компактное описание совершаемых действий, чтобы хоть немного ужать материал в стихийно разрастающейся книге. Как в любом языке программирования оператор состоит команды и значения, которое может быть числовым и текстовым. Удалось это сделать или нет – решать тому, кто прочтёт эту книгу.

Общая структура языка

Имя_окна – открыть требуемое окно для выполнения действий.
Дальнейшая цепочка действий (если таковая имеется) подразумевается в текущем окне. Например:
Debugger – открыть окно отладчика кнопкой или комбинацией горячих клавиш.
Имя_окна [действие] – совершить определённое [действие] в окне.
Например:
Cassette Recorder [Play] – в окне виртуального магнитофона нажать кнопку воспроизведения или соответствующую комбинацию клавиш.
← ввод числового значения в память или регистр.

Основные команды языка «God Mode Algorithm»

Работа в окне Spectaculator

BASIC ← Набор Команд – набрать BASIC последовательность натуральным или традиционным способом.
Generate NMI – сделать одиночный NMI-выстрел клавишей «*F5*».
New File [.tzx] – создать и открыть для записи новый файл *.tzx* с требуемым именем (*Ctrl+N* → *Audio Cassette File (.tzx)*)
Open File [.scr] – загрузить требуемый *.scr* файл перетащив его мышкой на экран *Spectaculator* или после нажатия «*Ctrl+O*» выбрав соответствующий файл.
Pause [OFF] – приостановить работу программы *Spectaculator* кнопкой «*PAUSE*».
Pause [ON] – возобновить работу программы *Spectaculator* кнопкой «*PAUSE*».
Reset – выполнить сброс «Spectrum мира» в окне кнопочкой или комбинацией клавиш «*CTRL+R*».

Работа с окном магнитофона

Cassette Recorder – открыть окно виртуального магнитофона кнопочкой или нажатием комбинации клавиш «*Ctrl+K*».
Cassette Recorder [*.tzx] – в окно виртуального магнитофона перетащить мышкой требуемый *.tzx* файл, для загрузки в память.
Cassette Recorder [Record] – включить виртуальный магнитофон на запись кнопочкой Record или нажатием комбинации клавиш «*Ctrl+Alt+Space*».
Cassette Recorder [Stop] – остановить запись или воспроизведение на кассету виртуального магнитофона кнопочкой  «*Stop*» или нажатием комбинации клавиш «*Ctrl+Space*».
Cassette Recorder [Play] – включить виртуальный магнитофон на воспроизведение кнопочкой  «*Play*» или нажатием комбинации клавиш «*Ctrl+Space*».

Cassette Recorder [Eject] – вынуть виртуальную кассету из магнитофона, нажатием кнопочки  «*Eject*».

Работа с прочими окнами

Import Machine Code X – записать массив данных из файла по адресу X, внесением значения в одноименное окошко и нажатием кнопки «OK».

Работа в окне отладчика

Внешние команды отладчика и вспомогательных окон

Debugger – открыть окно отладчика нажатием комбинации клавиш «*Ctrl+ENTER*», подготовив среду программирования для ввода данных.

Dec – настройка отображения чисел в десятичном виде нажатием комбинации клавиш «*Ctrl+H*» или кнопочкой  в окне.

Go To X – внесение в окошко «*Go To*» числового значения для быстрого перемещения в нужную точку области памяти (*Ctrl+G → X → ENTER*) без переустановки Стрелочки.

Trace – запуск набранной программы путём выхода в программу клавишей «*F5*» или кнопочкой  .

Add Breakpoint X – поставить  малиновую точку прерывания на адрес X.

Remove Breakpoint X – удалить точку прерывания с адреса X.

ASCII – переключение отображения содержимого памяти на формат символов кнопочкой  или комбинацией клавиш «*CTRL+A*».

Внутренние команды

X ← Y – Ввод в ячейку памяти X значения Y, где X=0…65535, Y=0…255.

PC ← X – перемещение Стрелочки на указанный адрес.

AF ← AF-Y – вычесть требуемое значение из регистра (обычно для снятия флаговой галочки)

AF ← AF+Y – прибавить требуемое значение к имеющемуся в регистре (обычно для установки флаговой галочки)

ПРИЛОЖЕНИЕ 3
Примеры программ

Как там начинается приложение с примерами BASIC программ в самоучителях?

«*Китайская книга изменений для игры в ящеры, которая в итоге, разрывает всех на британский флаг от соответствия между футами и ярдами*».

Но цирк уехал еще в позапрошлой главе, поэтому ниже предлагаю вполне серьёзные программы.

Первый из этих алгоритмов позволяет запустить BASIC программу с любой строки прямо из отладчика, не прибегая к синтезу команд в буфере `E_LINE (23641)`.

```
Debugger
Dec
Go To 23610
23610 ← 255
23611 ← 128
23618 ← 1
23620 ← 1

Go To 65364
65364 ← 4867
```

```
SP ← 65364
PC ← 7030
Trace
```

По возврату в *Spectaculator* после команды «*Trace*» произойдёт немедленное выполнение первой строки с первого оператора.

Следующая программа развивает тему главы 27 «*Советы Неэкспертов: синтез BASIC области для остановки игр*». Доработанный алгоритм загружает фирменную программу *DEFLEKTOR* с навороченной защитой (прилагается к книге) и сохраняет её в полностью прозрачный «`Bytes: deflektor`». Кроме того устанавливается бесконечная жизнь, перевыбираются клавиши, а таблица рекордов очищается от нереальных результатов.

При определенной сноровке, этот алгоритм выполняется за 8-10 минут.

```
Cassette Recorder [Deflektor (original).tzx]
Debugger
Dec
Add Breakpoint 64792
Trace
BASIC ← LOAD ""ENTER
Remove Breakpoint 64792
```

Дополнительная чистка живых ячеек от мусора:

```
36402 ← 0 0
39799 ← 0 0
41232 ← 0 0
```

Установка бесконечной жизни:

```
34473 ← 0                    ; lives (для восстановления 34473 ← 61)
41784 ← 0                    ; energy (для восстановления 41784 ← 60)
42724 ← 0                    ; overload (для восстановления 42724 ← 165)
```

Сброс таблицы рекордов с именами:

```
40942 ← 48 48 48 50 53 54    ; 000256
40951 ← 48 48 48 49 50 56    ; 000128
40960 ← 48 48 48 48 54 52    ; 000064
40969 ← 48 48 48 48 51 50    ; 000032
40978 ← 48 48 48 48 49 54    ; 000016
40987 ← 48 48 48 48 48 56    ; 000008

40997 ← ХАВКА
41005 ← ЕРЕМА
41013 ← РН-60
41021 ← ОТКАТ
41029 ← ТЕТКА
41037 ← АНТОН
```

Корректировка клавиш управления режима KEYBOARD:

```
38905 ← 87                   ; переназначить паузу с клавиши P на  I
39056 ← 223 219 254 31 0     ; переназначить лево/право на привычные O и P
```

Принудительный выход в BASIC с воссозданием системных переменных:

```
Go To 23617
```

```
23617 ← 0
23665 ← 0
PC ← 4626
HL ← 65367
Trace
Cassette Recorder [Eject]
```

Создание нового .tzx

```
New File [Deflektor.tzx]
Cassette Recorder [Record]
Debugger
```

Запись блока с автосбросом: (BASIC аналог SAVE "deflektor "CODE 24576, 32724)

```
17996 ← 24576                  ; стартовый адрес для блока данных
18000 ← 3                      ; тип блока Bytes
18001 ← deflektor              ; имя заголовка на конце один пробел
18011 ← 32724 24576            ; длина и стартовый адрес для данных заголовка

IX ← 18000                     ; место размещения заголовка
SP ← 17996          ; установить SP-башенку в любое свободное место для автосброса
PC ← 2436                      ; переход на подпрограмму записи
Trace

Cassette Recorder [Stop]
```

На выходе появился готовый `Bytes:` с чистой эталонной игрой для модификаций. Теперь вы можете загрузить блок натуральным способом по команде `LOAD ""CODE`. Точка входа в игру – 33800.

REKЛАМА

УВАЖАЕМЫЙ ЧИТАТЕЛЬ!

Нужен ли вам SPECTACULATOR в 2025 году?

Если вы счастливый обладатель любого старенького ноутбука/компьютера и телефона с возможностью раздавать интернет по wi-fi, то без сомнений, можно сказать «ДА»!

Минимальные телодвижения в Google, с пятью минутами установки повысят винтажность вашей машины до 80 lvl, и уже не каждый олдфаг скажет, что это старый хлам домохозяек для сидения в соцсетях и просмотра сериалов.

Естественно, у Вас возникнет масса вопросов о необходимости обладания этой утилитой. Первым и самым важным вопросом, будет вопрос о программном обеспечении и навыках работы. Для начала порадуем: всё мыслимое и немыслимое программное обеспечение для SPECTRUM 48 с тысячами игр и вспомогательным софтом весит около гигабайта и легко передаётся через файлообменники за полчаса, во-вторых, в интернете есть такие сайты, как Virtual TR-DOS, World of Spectrum и прочие, причём довольно часто это бесплатный софт без всяких креков. Ну а теперь огорчим: 48К версии программ для магнитофона, в .tap и .tzx форматах, выкладывают в интернет не так часто, как хотелось бы, зато гораздо чаще доходят в формате .scl и .trd.

Это к вопросу о программном обеспечении.

Что ещё Вас заинтересует? Ну, вероятно, возможности, которые Вам предоставит установленная программа. А возможности немалые, а какие история умалчивает. Ну уж точно не квакающий диск на 80 Петабайт.

Если Вы ещё не решились, то посчитайте, во сколько Вам обойдётся это удовольствие для прочтения вышеидущего творения.

Десяток убитых часов в общественном транспорте, пролюбленные перекуры на полноценных 8-ми часовых рабочих сменах и любой немаковский компьютер: от старенького с Windows XPюшей, до понтового на Windows 10. Кроме того, Вам понадобятся большие колонки (мощностью 100 китайских МВт), чтобы от щемящих душу звуков работы команды LOAD "", соседи смежных квартир выпрыгивали из окон от безысходности. Впрочем, с наушниками приложение тоже будет прекрасно работать, но Вы оставите без шикарной музыки «Пи-и-и-и-и Тщю» детей, жену, тёщу, родителей и других обитателей квартиры, которые не смогут по достоинству оценить Ваши творения.

Итак, я надеюсь, что Вы уже готовы решиться к усовершенствованию своего компьютера и прочтению всего нижеследующего бреда.

Желаю успеха!

УВАЖАЕМЫЙ РАДИОЛЮБИТЕЛЬ!

Нужен ли Вам SPECTRUM 128?

Если Вы счастливый обладатель Spectrum 48, то без сомнений можно сказать «ДА»!

Минимальные аппаратные доработки повысят класс Вашей машины, и уже не каждый советский человек скажет, что это игрушка.

Естественно, у Вас возникнет масса вопросов о необходимости обладания новой машиной. Первым, и самым важным вопросом будет вопрос о программном обеспечении. Для начала порадуем: во-первых, все программное обеспечение, котрое Вы собрали для SPECTRUM 48, будет исправно работать на Вашей новой машине, во-вторых, начиная с 1986 года (т.е. года создания Клайвом Синклером SPECTRUM 128) почти все фирмы, выпускающие программное обеспечение для персональных компьютеров, перестроились. Теперь на рынок стали выходить версии программ как для SPECTRUM 48, так и для SPECTRUM 128, причем довольно часто совмещенные. Ну, а теперь огорчим: 128К версии программ доходят до нашей страны не так часто, как хотелось бы, зато, гораздо чаще доходят совмещенные.

Это к вопросу о программном обеспечении.

Что еще Вас заинтересует? Ну, вероятно, возможности, которые Вам предоставит новая машина. А возможности немалые. Если Вы программист-системщик, то Вам не надо объяснять, что такое электронный квазидиск на 80 Кбайт; если Вы пишете программы на Basic'e, то, вероятно, представляете себе, что такое экранный редактор; если Вы музыкант, то не мне Вам рассказывать, что такое трехголосый музыкальный инструмент с широчайшими возможностями. Даже если Вы просто пользователь, то и тут Вам поблажки: теперь для загрузки программы, Ваши усилия ограничатся нажатием на одну клавишу.

Если Вы еще не решились, то посчитайте, во что Вам выльется это удовольствие.

Десяток корпусов логики, две 8-ми Кбайтные микросхемы ПЗУ (2764) и линейка РУ5. Кроме того, Вам понадобится музыкальный процессор 8912, без которого, впрочем, машина будет прекрасно работать, но Вы останетесь без шикарной музыки, которую может воспроизводить SPECTRUM 128.

Итак, мы надеемся, что Вы уже готовы решиться на усовершенствование своего компьютера.

Желаем успеха!

Рис. 553. Оригинал рекламы из книги старой пиратской книги.

СОДЕРЖАНИЕ

Красный-Синий-Зелёный, он же EuroYura и Купчинский Раздолбай

ZX- Spectrum BASIC с Н⍰⍰⍰⍰⍰ЛЯ

Издание эксклюзивное, единственное и неповторимое

Главный Редактор: Желтая  *Стрелочка aka курсор*
Художественный редактор: Купчинский Раздолбай 
Корректор: Купчинский Раздолбай
Штатный Клоун: Гамнюков Фёдор Акакиевич
Идейные вдохновители: группы Красная Плесень и Сектор Газа.

Самоучитель по работе с BASIC-системой. Прочитав книгу, вы поймёте принцип работы некоторых подпрограмм из ПЗУ, научитесь вручную вводить машинный код в Spectaculator, издеваться над машинным стеком с системными переменными, и создавать демки, которые будут сами набирать программы на BASIC, запускать и управлять процессами.
Понадобится ли этот опыт в дальнейшей жизни, я хрен знает. Написал просто по фану.

ISBN 3-16384-6912-0

ООО «ПЕРВЫЙ ПРИХОД» 101010, г, Торчок, ул. Конопляная 16.
Подписано к печати: 05.09.2025. Формат: doc, pdf. Печать клавиатурная. Бумага туалетная. Усл. печ. ½-10*15. Тираж: 100500 экз. Заказ FFC0h.
Отпечатано в типографии ордена Трудовой Мозоли правой руки имени Дуни Кулаковой.
696969, г. Болдогонск, ул. Пронская, 69.