VBC — MSX-совместимый компилятор BASIC (памяти Егора Вознесенского)

Fix Me!

Основные разделы:

- Руководство пользователя
- Руководство программиста
- Руководство системного программиста

Файлы:

solidbasic.zip	оригинал
ibas.txt	оригинал
apguide.txt	оригинал
vbc.dsk на основе vbc27b	оригинал

Часть I. Руководство пользователя

Глава 1. Что такое VBC ?

Компилятор создавался с целью получения языка, максимально приближенного к MSX BASIC. Более того, поскольку возможна отладка с помощью интерпретатора, из компилятора исключена диагностика некоторых ошибок, которые можно выявить в процессе отладки на интерпретаторе.

Входной язык представляет собой несколько расширенную версию MSX BASIC, состоящую из ядра и расширенную следующими функциями:

- оператор цикла `WHILE … WEND` (эквивалентен таковому в IBM–бейсике (BASICA, GW-BASIC).
- средства поддержки раздельной компиляции (`PROCEDURE`, расширение оператора `CALL`, совместимость с MSX-C.
- опция управления компилятором `PRAGMA`.

Глава 2. Как запускать компилятор

Если у вас есть уже готовая программа на BASIC, то чтобы её оттранслировать достаточно лишь набрать команду:

```
A>BAS <имя файла> [ввод]
```

При этом подразумевается, что файл имеет расширение BAS.
Если при трансляции не возникнет ошибок, то вы станете обладателем COM-файла с именем исходного файла, содержащего вашу программу в оттранслрованном виде. Её можно сразу запустить.

Для более любознательных сообщаю, что компилятор можно запустить и без BAT-файла командой

```
A>VBC <имя файла>.<ext> [/C]
```

Имя файла нужно указать полное, с расширением.
Необязательный ключ `/C` указывает компилятору, что нужно произвести только проверку синтаксиса, без создания выходного файла.

При работе компилятор сообщит вам свою версию, может напечатать листинг программы (для этого следует включить в текст программы операторы PRAGMA LIST), а в конце работы — распечатает список всех объявленных переменных и количество обнаруженных ошибок. Если ошибок не было обнаружено, выводится сообщение

```
complete
```

Часть II. Руководство программиста

Глава 1. Совместимость со стандартами

1.1. ГОСТ-совместимость

Входной язык компилятора полностью включает в себя ЯДРО и расширен в соответствии с УРОВНЕМ1 и УРОВНЕМ2 расширения.
Кроме того, он поддерживает ряд опций, не предусмотренных в ГОСТ, а именно:

- средство управления компиляцией PRAGMA;
- средств для поддержки раздельной компиляции: EXTERN, GLOBAL, PROCEDURE;
- логические операции AND и OR в операторах IF могут быть условными;
- можно нумеровать только строки, к которым есть ссылка;
- реализован механизм присваивания, аналогичный таковому в языке С — присваивание по цепочке.

Кроме того, часть возможностей, специфицированных в расширенных модулях ГОСТ не реализована в связи с изменением модели распределения памяти времени компиляции (см. п.1.3).

1.2. Совместимость с MSX

Fix Me!

Компилятор соответствует MSX BASIC, с учётом пп. 1.1 и 1.3 и включает в себя:

- математические возможности — полностью;
- работу с памятью — всё, кроме ERASE;
- графику — полностью *);
- музыкальные возможности — согласно спецификации PSG *);
- спрайты — полностью;
- ввод–вывод — полностью, с исключениями согласно п. 1.3;
- управляющие операторы: все, добавлен цикл WHILE,
- исключение: не определён результат RESUME без параметров;

Соответствие MSX-2 стандарту:

- графика — полностью*);
- системные функции:
  - COPY SCREEN
  - GET TIME
  - SET TIME
  - GET DATE
  - SET DATE
  - SET BEEP
  - SET ADJUST
  - PUT KANJI
- функции работы с палитрой.

Соответствие MSX 2 + стандарту:

- режимы `SCREEN 10,11,12` (YJK–графика)
- `SET SCROLL`
- доступ к регистрам S1…S9 видеопроцессора

Примечание: *) — звёздочкой отмечены полностью совместимые с MSX-стандартом разделы, в которых имеется, однако, одно исключение: в макроязыке операторов `DRAW` и `PLAY` не определён параметр X (смотри руководство по MSX BASIC), что обусловлено особенностями работы откомпилированной программы с памятью.

Функции, соответствующие более позднему MSX-стандарту, могут быть реализованы на машине более раннего стандарта при условии наличия необходимых аппаратных средств и (или) BIOS. Например `SET SCROLL` можно выполнять на всех компьютерах с V9938, но при этом скроллинг будет только по вертикали.

Часть операторов (например `SET PASSWORD` или `SET VIDEO`), не реализованы или как не имеющие смысла в откомпилированной программе или как редко употребляемые. Их, при необходимости, можно эмулировать соответствующими комбинациями OUT'ов.

1.3. Отходы от стандартов

Отходы от стандартов обусловлены оптимизацией получаемого кода и невозможностью реализации части функций в модели компилируемой программы. Эти отходы перечислены ниже:

- Все функции ввода/вывода а также внешние функции получают и возвращают параметры только целого типа, например номер записи в операторах GET / PUT должен быть 1…65535;
- Память под массивы и переменные выделяется 1 раз при компиляции, поэтому оператор `ERASE` не определён, `CLEAR` очищает только строки и файлы и при объявлении массивов для указания размерностей следует использовать ТОЛЬКО константы.
- Массивы могут быть одномерные или двумерные — это досадное недоразумение, которое легко обойти, проэмулировав массив большей размерности.
- Типы переменных жёстко связаны с именами. Нельзя, например использовать в одной программе A и A$, а также массив с именем, совпадающим с именем простой переменной. Подробнее об этом в главе 3.
- Пробелы в тексте программы ЯВЛЯЮТСЯ разделителями.

- При разборе текста в качестве имени переменной берётся строка до ближайшего разделителя, поэтому возможны синтаксические ошибки в работавших на интерпретаторе программах, например:

  ```
  FOR I=LTO 100:…''
  ```

  даст

  ```
  Syntax error
  ```

  Чтобы ошибки не было, следует применить скобки либо после имени переменной ставить пробел:

  ```
  FOR I=(L) TO 100
  ```

  или

  ```
  FOR I=L TO 100
  ```

Для облегчения адаптации программ в конце данного руководства помещён список ошибок и возможные их причины.

Глава 2. Преимущества компилятора IBAS

Главным преимуществом откомпилированной программы является во много раз большая скорость её исполнения, что особенно заметно при целочисленных операциях. В данном случае мы имеем ускорение вычислений в 20…50 раз по сравнению с интерпретатором в зависимости от типа программы. (это для целых чисел). Скорость вычислений с плавающей точкой возрастает в 1.3…2.5 раза в основном за счет операций загрузки/записи. Скорость выполнения

строковых операций зависит от типа операции, но в целом немного выше, чем в интерпретаторе. Особенно быстро выполняются короткие циклы в целых числах с вычислением массивов внутри них.

Компилятор снабжен мощным оптимизатором и ориентирован на написание программ управляющего характера, в связи с чем в нем основным типом данных являются целые числа. Такие популярные функции, как RND и SQR имеют версии для целых аргументов, возвращающие также целое число и чрезвычайно быстродействующие. Деление при применении к двум целым также возвращает целое т.е. a% / b% =a% \ b% .

Компилятор совместим с MSX ASCII C при вызове из MSX BASIC внешних подпрограмм. Кроме того он имеет средства раздельной компиляции модулей, написанных на MSX BASIC.

Что касается скорости компиляции, то она несколько превосходит скорость компиляции аналогичной программы компилятором MSX-C, а за счёт использования некоторых ухищрений при оптимизации и специфических команд процессора Z80 оптимальность получаемого кода не меньше, а местами даже больше, чем у семантически аналогичной программы, написанной на C и откомпилированной компилятором MSX ASCII C.

На выходе компилятора получается программа на ассемблере в мнемонике Z80, что также лучше, чем мнемоника 8080 на выходе MSX ASCII C. При желании квалифицированный программист может внести изменения уже в файл ассемблера. Для облегчения таких работ у компилятора имеется опция, позволяющая вводить исходный текст программы на MSX BASIC в выходной файл в виде комментариев.

Библиотека компилятора обеспечивает достаточно быструю работу с файлами и имеет несколько функций для доступа к ресурсам ОС, позволяющих легко писать системные утилиты с современным, графическим интерфейсом.

В общем, данный компилятор является на данный момент практически единственным компилятором, собирающим в себе все возможности и на таком уровне совместимый со стандартом MSX.

Глава 3. Ядро компилятора

3.1. Переменные и массивы

Переменные и массивы идентифицируются по имени, причём оно может быть произвольной длины. В соответствии со стандартом на MSX BASIC, имена различаются по первым двум символам, первый из которых должен быть латинской буквой, а второй — буквой или цифрой. Не допускается использование русских букв и графических символов. Строчные латинские буквы переводятся в прописные.

Сразу после имени переменной может стоять необязательный символ типа:

%	для целой переменной
$	для символьной переменной
!	для переменной одинарной точности
#	для переменной двойной точности

Если таковой не стоит, то тип переменной определяется по первому символу имени переменной в соответствии с оператором DEF, имеющему вид:

```
DEF {INT|STR|SNG|DBL} <символ>[- <символ>]...
```
,

где

- INT, STR, SNG, DBL обозначают соответственно целый, символьный, одинарный и двойной типы,
- <символ> — первый символ имени переменной
- конструкция <символ А>-<символ Б> означает «от А до Б включительно»

В отличие от MSX BASIC по умолчанию все переменные определены

как целые, поэтому следует в программах первой строкой вставить `DEFDBL A-Z`. Напротив, конструкцию типа `DEFINT A-Z` можно опустить.

Здесь имеется и ещё одно отличие от стандарта MSX: как переменные всех типов, так и массивы в компиляторе распознаются *исключительно* по двум первым символам, в результате чего возникают ошибки в программе в следующих ситуациях:

- если в программе объявлены в разных местах переменные одинакового имени, но с разным постфиксом точности:
  - A и A$
  - X и X#
- если аналогичное произошло с массивом и простой переменной;

Кроме того, нельзя использовать массивы, объявленные по умолчанию.

Тип переменной устанавливается компилятором один раз при первой встрече согласно правилам, описанным выше, и остаётся неизменным на протяжении всей программы, поэтому символ точности можно указывать только один раз, что экономит текст программы и избавляет пользователя от возможных ошибок в результате пропуска такого символа в программе.

Все перечисленные отличия от стандарта MSX не являются прямыми противоречиями ГОСТ и введены для улучшения стиля программ, написанных на BASIC.

3.2. Выражения

3.2.1. Математические выражения

В выражениях можно применять следующие операции, причём в следующем приоритете (начиная с наинизшего) :

- операция инверсии «логическое не»

  ```
  NOT <выражение>
  ```

  `NOT` инвертирует все биты в значении выражения, стоящего после него. Если тип выражения отличен от целого, он приводится к целому.

- логические операции `AND`, `OR`, `XOR`, `EQV` Это двуместные операции; производятся побитно, за исключением операций `AND`, `OR` в операторе `IF`. В последнем случае, если такие операции связывают два или более выражения отношения, они выполняются условно, т.е. до тех пор, пока не будет окончательно ясен результат операции.

  Операция `EQV` соответствует `NOT XOR`

  Операции возвращают целый результат. Оба аргумента приводятся к целому типу.

- операции отношения > < = <> >= ⇐ Это двуместные операции, возвращающие целый результат в следующем виде: истине соответствует -1, ложному результату 0.
  Аргументами операции могут быть целые, плавающие числа и строковые выражения, причём числовые выражения приводятся к более точному типу, а более короткие строки считаются меньшими, чем более длинные при совпадении первых символов.

  Сравнивание строки с числом не допускается.

- операции сложения и вычитания +,-
- операции умножения и деления *, /, \, `MOD`

Операция «/» при применении к двум целым возвращает целый результат, т.е. работает аналогично «\».

Операции «\» и `MOD` преобразуют оба аргумента в целый тип. `MOD` возвращает остаток от деления.

1. операция возведения в степень ^
2. операция смены знака - (одноместная)
3. операции в скобках и оператор присваивания (см. 3.2.2)
4. функции (см. 3.2.3)

3.2.2. Оператор присваивания

Оператор присваивания имеет вид:

```
[ LET ] { переменная | массив } = <выражение>
```

Если такой оператор заключить в фигурные скобки, его можно использовать в выражениях следующим образом:

- Если присваивается целое целой переменной, то результатом будет это число.
- Если присваивается число с плавающей точкой (не важно, чему), то результатом будет это число.

Во всех остальных случаях результатом будет адрес переменной или массива, куда произошло присваивание.

Этот оператор является очень мощным средством рационализации кода программ, но так как его применение достаточно сложно, он рекомендуется только очень опытным программистам.

Пример применения оператора:

```
WHILE ( {A%={B%=B%-1}>0} ): … … :WEND
```

Здесь происходит B% итераций цикла `WHILE`, причём в каждой из них переменная A% содержит значение истинности для условия продолжения цикла.

3.2.3 Функции

В математических выражениях можно употреблять функции, определённые в MSX, причём их действие стандартно, поэтому здесь будут полно описаны лишь три функции, которые можно употребить не только в соответствии со стандартом.

Функция `SQR` Если её аргумент плавающее число, то результат плавающий, если же аргумент целый, то и результат целый, округлённый до ближайшего меньшего.

Функция `ABS` - аналогична `SQR`, естественно без округления.

Функция `RND` - если аргумент плавающий, работает стандартно в отличие от случая с целым аргументом, который описан ниже.

Если функция вызвана как `RND(M)`, где M - целое выражение, она возвращает строго случайное, равномерно распределенное целое число в интервале от 0 (включительно) до M (исключительно), положительное или отрицательное в зависимости от знака M.

Все функции работы с файлами, как уже упоминалось в главе второй, возвращают целые значения.

3.2.4. Строковые выражения

Строковые выражения могут иметь вид либо строковой константы, либо строковой переменной, либо строковой функции, либо того, другого и (или) третьего, соединенного знаком конкатенакции +.
Строковые операции полностью соответствуют стандарту MSX и поэтому здесь не рассматриваются.

3.2.5. Внешние подпрограммы и функции

Внешние подпрограммы и функции являются расширением оператора MSX BASIC `CALL`

Подпрограммы вызываются так:

- `CALL <имя> ( список параметров )`
- `_ <имя> ( список параметров )`

Последний оператор может использоваться как функция внутри числового выражения. Внешние функции совместимы с C, причем параметры и возвращаемый результат должны описываться как int либо unsigned, т.е. иметь длину 2 байта.

3.2.6. Функции, определяемые пользователем

Как и MSX BASIC, так и данный компилятор позволяют определять небольшие однострочные подпрограммы–функции отдельно от основного текста программы при помощи оператора `DEF FNx`

```
DEF FN< ид_символ > [(список_параметров)]=<текст>
```

Список параметров может включать до 16 выражений, разделенных запятыми. Идентификаторы параметров локальны и могут иметь такие же имена, как переменные внутри основной программы. Но в отличие от интерпретатора, внутри такой функции можно использовать переменные и массивы основной программы, если их имена не совпадают с именами формальных параметров:

```
DIM TT(10)
… …
DEF FNA(I)= TT(I) AND &h33
```

При использовании пользовательских функций с большими объемами строковых вычислений следует остерегаться переполнения строкового стека, что может дать ошибочный результат.

3.3. Управляющие операторы

3.3.1. Номенклатура управляющих операторов

К управляющим операторам относятся операторы, позволяющие изменить последовательность исполнения других операторов программы с целью реализации сложных алгоритмических структур. Входной язык компилятора содержит широкий набор таких операторов, включающий операторы безусловного перехода и перехода по значению, а также операторы ветвления, цикла и прерываний:

- `GOTO` — оператор перехода;
- `ON <выражение> GOTO` — переход по значению;
- `IF … THEN … ELSE` — оператор условия;
- `FOR … NEXT` — операторы цикла;
- `WHILE … WEND` — операторы цикла с неизвестным числом повторений.

Эти операторы полностью соответствуют ГОСТ и стандарту MSX (интерпретатору), но учитывая их особенную значимость для программирования ниже будет дано достаточно подробное описание каждого из них. Операторы прерывания, как особо важные и специфические, выделены в особую главу.

3.3.2. Операторы перехода

GOTO <метка> — безусловный переход на строку с номером <метка>. Если строки с указанным номером не существует, то ошибка диагностируется только на этапе ассемблирования.
ON <выражение> GOTO <метка>,<метка>, ... — после вычисления значение выражения приводится к целому и происходит переход на строку, номер которой в списке стоит на соответствующем по порядку месте. При выходе за пределы возможных значений или если соответствующий номер строки отсутствует, выполнение программы продолжается со следующего оператора.

В упомянутых операторах вместо слова GOTO можно употребить GOSUB, при этом переход будет осуществляться на подпрограмму, выход из которой осуществляется оператором RETURN

3.3.3. Оператор ветвления

Синтаксис этого оператора:

```
IF <выражение> GOTO <метка>
```

или

```
IF <выражение> THEN <операторы>[ ELSE <операторы>]
```

Если во второй форме сразу после ключевых слов THEN или ELSE должен стоять оператор безусловного перехода, то ключевое слово GOTO можно опустить, т.е. писать сразу номер строки.

3.3.4. Операторы циклов

Fix Me!

Существует два вида циклов: с заданным числом повторений и с неизвестным заранее числом повторений. Оба типа циклов реализованы в данным транслятором. В первом случае используются операторы FOR и NEXT, а во втором — WHILE и WEND со следующим синтаксисом:

```
FOR <переменная>=<начало> TO <конец> STEP <приращение>
```

Начальное значение присваивается переменной, являющейся счётчиком цикла, после чего выполняется тело цикла. Затем к счётчику прибавляется значение приращения и результат сравнивается с конечным значением. Если значение счётчика больше при положительном приращении или меньше при отрицательном, то выполнение цикла прекращается. В любом случае тело цикла выполняется по крайней мере один раз.

```
NEXT [<переменная>[,<переменная>...]]
```

последний оператор цикла. При отсутствии переменных закрывает последний цикл, при наличии нескольких — эквивалентен последовательности из нескольких операторов NEXT подряд. Попытка закрыть внешний цикл раньше внутреннего вызывает ошибку.

```
WHILE <выражение>
```

начало цикла с неизвестным числом повторений. Цикл выполняется до тех пор, пока <выражение> не равно нулю. Тело цикла может не выполняться ни одного раза.

WEND — конец цикла WHILE. Закрывает самый вложенный цикл.

Глава 4. Ввод-вывод

Компилятор BASIC включает в себя полную систему ввода и вывода в соответствии с требованиями системы MSX. Операторы ввода-вывода полностью стандартны и поэтому ниже не будет дано их полное описание, а они будут просто перечислены, с краткими комментариями при надобности.

- INPUT — стандартен полностью;
- LINE INPUT — стандартен полностью;
- PRINT — стандартен полностью, поддерживается форматный вывод при помощи оператора USING;
- INKEY$ — стандартен полностью;
- INPUT$(n) — стандартен полностью, имеется и файловый вариант этого оператора:

  ```
  INPUT$(LUN, n)
  ```

- OPEN — стандартен полностью, поддерживает системные устройства CRT: , GRP: , LPT:, при открытии дискового файла
- FOR OUTPUT стирает существующий, если он есть;
- CLOSE — стандартен полностью, при наличии вывода в файлы обязателен в конце программы;
- BLOAD — стандартен полностью, но нельзя делать

  ```
  BLOAD ,r
  ```

- BSAVE — стандартен полностью;
- GET, PUT — стандартны полностью;

Определены также следующие функции:

- CSRLIN
- POS
- FPOS
- LOC
- LOF
- EOF
- LPOS
- DSKI$
- DSKO$
- DSKF

Глава 5. Графика

5.1. Графические операторы

Fix Me!

Компилятор поддерживает все графические операторы:

PSET (X , Y) [,цвет][,лог] PRESET (X , Y) [,цвет][,лог]	поставить точку
LINE [(X1,Y1)] - [STEP](X2,Y2) [,цвет][,{| B | BF}][,лог] PAINT (X,Y)[,цвет1][,цвет2]; CIRCLE (X,Y),радиус [,цвет][,нач.угол][,кон.угол][,сжатие] COPY	во всех вариантах
DRAW <командная строка>	
функция POINT(X,Y)	возвращает цвет точки

Все эти операторы полностью совместимы с MSX, и подробнее об особенностях их применения и значении параметров можно узнать в любом руководстве по MSX BASIC, например, здесь

Логические операции те же, что и в MSX BASIC 2.x (для MSX 2): PSET, PRESET, AND, OR, XOR с и без префикса T. Они имеют смысл и действие только в экранных режимах MSX 2 SCREEN 5…8.

5.2. Спрайты

Полностью соответствуют MSX-спецификации; ниже приведен список операторов, которые контролируют спрайты. Подробные сведения об их использовании можно получить из литературы по MSX 1 и MSX 2, например здесь и здесь.

```
SPRITE$ ( N ) = <строка>
```

```
COLOR SPRITE( N ) = <число>
```

```
COLOR SPRITE$( N ) = <строка>
```

```
PUT SPRITE <номер>,(X,Y) [,цвет][,шаблон]
```

Определено также прерывание `ON SPRITE GOSUB`, возникающее при спрайтовых коллизиях. Операторы `COLOR SPRITE` могут работать только на MSX 2 в экранных режимах выше 3.

5.3. Вывод текста на графический экран

Для этого следует открыть системное устройство `GRP:` и выводить на него данные. Данные выводятся с последней позиции графического курсора.

Для машин, оборудованных Kanji ROM (это все японские MSX 2), можно применить оператор `PUT KANJI`:

```
PUT KANJI [[STEP](X,Y)],<номер кода>,[,цвет1][,цвет2][,PG]
```

где

- номер кода — целое двухбайтовое число, каждый из байтов которого находится в диапазоне 21h...7Fh
- `PG` — признак разбиения. Если он равен 0, то выводится весь символ, если 1-чётные строки, если 2 — нечётные.

Вниманию русскоязычных: Kanji ROM содержит не только японскую тарабарщину, но и все символы латинского, греческого и русского алфавита, причём есть даже буквы «ё» и большой твердый знак «Ъ».

5.4. Цвета

Цвета устанавливаются при помощи семейства операторов `COLOR`. Эти операторы позволяют задавать логические цвета на экране и палитру каждого из цветов.

`COLOR <передний план>,<задний план>,<бордюр>` — Задает цвет (логический). Любые из параметров здесь могут отсутствовать, при этом соответствующий цвет не будет изменен.

`COLOR=NEW` или `COLOR` (без параметров) — инициализирует палитру, цвета приводятся к стандартным в MSX.

`COLOR=RESTORE` — восстанавливает палитру из PT видеопамяти.

`COLOR=(<цвет>, R, G, B)` — задание палитры RGB для цвета <цвет>.

Операторы, отличные от первого в данном списке будут работать только на MSX 2, MSX 2 +.

Глава 6. Операторы системных установок

6.1. Область применения

Fix Me!

Операторы системных установок предназначены для реализации различных операций непосредственно связанных с аппаратным обеспечением MSX и делятся ниже на операторы для работы с экраном, LSI-SC, памятью и портами ВВ.

6.2. Работа с экраном

Fix Me!

`SCREEN` — устанавливает экранный режим и другие параметры системы. Имеют смысл все его параметры, все они полностью стандартны, поподробнее здесь.

`VDP(n)` — псевдопеременная, соответствующая регистру VDP n; n обозначает следующее:

- 0...7 == R#0...R#7 (чтение и запись),
- 8 == S#0 (только чтение),
- 9...28 == R#8...R#27 (чтение и запись,
- R#25,R#26,R#27 имеют смысл только на машинах с Yamaha V9958 (VDP), это машины MSX 2 +,
- 33...47 == R#32...R#46 (регистры команд, только запись).

Доступ к регистрам статуса 1...9 (только чтение) осуществляется как `VDP(-1)…VDP(-9)`.

`BASE(n)` — псевдопеременная, содержащая базовые адреса VRAM и позволяющая их изменять. Запись возможна только в переменные с индексами от 0 до 19.

`SET ADJUST (X,Y)` — сдвигает экран на -8..+7 точек по вертикали и горизонтали.

`SET SCROLL X,Y,<маска>, <2 стр>` — проводит ролик по вертикали (MSX 1, MSX 2) и горизонтали MSX 2 +. X может иметь значение от 0 до 511, Y от 0 до 255. Если `<маска>` не 0, то ролик дискретный, 8 линий по Y и 8 точек по X (16 для `SCREEN` 6 и 7). `<2 стр>` имеет смысл только на MSX 2 + и означает горизонтальный скроллинг по горизонтали через 2 соседние страницы VRAM (позволяет увеличить «ширину» окна).

`SET PAGE <активная>[,<дисплей>]` — установка страниц VRAM в режимах 5...8 (12). Если номер дисплейной страницы не указан, он полагается равным номеру активной страницы.

6.3. Работа с системной микросхемой "100 ног" (LSI-SC)

Fix Me!

Системный контроллер MSX 2 (MSX-SYSTEM) включает: звукогенератор PSG, часы реального времени, независимую память с батарейным питанием. Для работы с ними можно употреблять такие операторы:

`SOUND <регистр>, <значение>`	запись байта в регистр PSG
`BEEP`	подача сигнала
`SET BEEP <тип>, <громкость>`	установка параметров сигнала `BEEP` (числа в интервале 1...4), не работает на MSX 1
`SET TIME <строка>[,a]`	запись времени в часы
`GET TIME <переменная>[,a]`	чтение времени из часов
`SET DATE <строка>[,a]`	запись даты в часы
`GET DATE <переменная>[,a]`	чтение даты из часов

6.4. Работа с памятью и портами

CLEAR <длина строковой памяти>, <макс. адрес>	можно применять только в самом начале главной программы, так как оператор изменяет положение стека
MAXFILES = N	изменяет число доступных LUN для файловой системы BASIC. Очищает файловые дескрипторы, FCB и буфера
PEEK(адрес)	читает содержимое памяти по адресу
POKE <адрес>,<значение>	пишет в память. Следует соблюдать осторожность, чтобы не записать данные на программу при слишком маленьком адресе
INP(порт)	вводит байт из порта ввода
OUT <порт>,<значение>	выводит байт в порт
WAIT <порт>,<маска И>, <маска XOR>	ждёт ненулевого результата выражения INP(порт) AND <маска И> XOR <маска XOR> при разрешенных прерываниях

Глава 7. Прерывания

MSX BASIC включает несколько групп прерываний: таймерное, от аппаратных средств, от клавиши CTRL+STOP и от ошибки.

Для всех прерываний процедура использования одинакова: сначала нужно определить программу-обработчик оператором ON xxxx GOSUB, а затем разрешить прерывания оператором xxxx ON. Впоследствии можно запрещать прерывания оператором xxxx OFF или приостанавливать их xxxx STOP.

```
ON KEY GOSUB <a1>,… <a10> , KEY (#) ON|OFF|STOP
```

Прерывание по нажатию функциональных клавиш. Когда клавиша связана с прерыванием, она лишается строковой интерпретации

```
ON STRIG GOSUB <a1>,… ,<a5>, STRIG(#) ON|OFF|STOP
```

Прерывание происходит по нажатии кнопки пультов или пробела

```
ON INTERVAL=<N> GOSUB <a>
```

каждые <N> таймерных тиков (1/60 сек.) происходит прерывание

```
ON SPRITE GOSUB
```

происходит прерывание при обнаружении наложения спрайтов

```
ON STOP GOSUB
```

вызывает прерывание по нажатию CTRL+STOP.
При этом остановки программы не происходит. STOP OFF вообще отключает возможность остановки программы CTRL+STOP, STOP ON разрешает её, но и прерывание тоже (если оно было установлено ранее.)

```
ON ERROR GOTO
```

определяет программу обработки ошибок, возврат из этой программы осуществляется оператором RESUME

Часть III. Руководство системного программиста.

Глава 1. Формат входных и выходных файлов

Входной текст представляет программу на языке MSX BASIC в текстовом (ASCII) представлении с длиной строки до 254 символов. Строки могут нумероваться только при надобности, строки на которые нет ссылок, можно не нумеровать. Порядок следования номеров строк также произвольный, но не должно встречаться нескольких строк с одинаковым номером. Если строка оканчивается строковой константой, допустимо не ставить закрывающей кавычки, однако это рекомендуется делать.

При чтении исходного текста повсюду в тексте, кроме строковых констант, буквы преобразуются в большие латинские. Разборка на операторы происходит построчно, при этом в качестве очередного оператора берется минимальная подстрока из очередной лексемы, где лексема — часть однородных А/Ц символов, ограниченная каким-либо разделителем. Разделителями являются: конец строки, пробел, все специальные символы, а также при разборке числа — любой другой символ, не соответствующий набору возможных в данном числе символов.

Оператор комментария должен быть последним или единственным в строке. Любой текст после комментария игнорируется.

Выходной результат трансляции — файл на ассемблере Z80, состоящий из: заголовка, тела программы, дампа переменных и дампа строк. Дамп переменных объявлен как DSEG, остальные части программы — как CSEG, что позволяет на этапе сборки выносить зону переменных в любое удобное место. Рассмотрим входной и выходной результаты трансляции на примере маленькой программы. Итак, имеем на входе файл TEST со следующим содержимым:

TEST.BAS

```
pragma source
A=1:B=A+1'this is remark
print "A=";b
```

Вызываем компилятор:

```
a>VBC TEST.BAS
SOLID SOFT BASIC compiler V2.0vx
....
....
complete.
```

Так как в программе указана опция SOURCE, текст программы распечатывался на экране, а также в качестве комментариев включался в объектную программу на ассемблере.

TEST.ASM

```
; Solid Soft BASIC compiler V2.0 <- версия компилятора.
    .Z80
;
    CSEG    <------------ Конец заголовка.
;
;A=1:B=A+1'THIS IS REMARK: <- исходный текст
    ld    hl,00001h  <- программа
    ld    (@A ),hl
    inc    hl
    ld    (@B ),hl
;
;PRINT"A=";B:
    call    ?FINIT##   <- Инициализация строк
    ld    hl,?CONST+00000h
    call    ?LDSTR##   <- Получение константы
    ld    hl,(@B )
    call    ?FSTRX##
    call    ?INCLF##
    ld    de,00000h  <- Номер канала PRINT
```

```
    call    ?PRINT##
;:
;
    ret                   <-  конец программы
    DSEG                  <-  зона данных

@A: ds    00002h       <-  переменные
@B:     ds      00002h
    CSEG  <---------------  Опять сегмент кода
?CONST:                       <-  зона констант
    db    041h,03Dh,000h,00h  <-  текст "A=\0"
    END                   <-  конец файла
```

Зона строковых констант является сегментом кода, что соответствует соглашению по MSX ASCII C 1.0. Так как рекомендованным сборщиком для MSX BASIC является LINK, зона констант подклеивается к соответствующей программе, а зоны данных, т.е. переменные всех модулей выносятся в конец COM–файла. Работать с L80 не рекомендуется, так как он не инициализирует конструкции DS и поэтому может произойти накладка со строковыми переменными.

Глава 2. Форматы данных на этапе исполнения

Программы используют три типа данных — целые, плавающие и строковые. Все операции с плавающими числами производятся с двойной точностью, поэтому одинарная точность — лишь способ хранения в памяти, но не тип данных. Данные представляются в следующих форматах:

- Целые:
  два байта, <LSB><MSB> — стандартный способ хранения целых процессором Z80. Для представления отрицательных чисел используется дополнительный код.

- Плавающий тип:
  <характеристика><мантисса>, где мантисса — 3 или 7 байт для одинарной и двойной точности, содержащие упакованное BCD-число из 6 или 14 цифр. Характеристика содержит: старший знаковый бит (если =1 — число отрицательное) и порядок, (десятичный -63...+63), сдвинутый на 64 (1...127). Если байт характеристики нулевой, все число нулевое, байт характеристики 80h не определен. При записи числа в переменную одинарной точности она округляется, при чтении дополняется справа четырьмя нулями.

- Строковый тип:
  Это самый сложный тип. Данные этого типа делятся на две части: дескриптор строки, хранящийся в теле строковой переменной и собственно тела строки, хранящегося где-нибудь в другом месте (в строковой куче, файловом буфере) или вообще отсутствующего.

  Дескриптор строки имеет длину 3 байта и вид:

  - DW <адрес тела строки>
  - DB <длина строки>

  таким образом, максимальная длина строки составляет 255 байт. Если длина строки равна нулю, строка считается пустой, не имеет тела, и значение поля адреса тела строки игнорируется.

  Тело состоит из трёх полей:

  - DW <указатель на следующее тело>
  - DW <указатель на дескриптор>
  - <строка переменной длины>

  В случае, когда переменная является переменной FIELD, т.е. тело находится в буфере файла и длина строки постоянна, два первых поля отсутствуют, но при этом указатель в дескрипторе указывает не на начало тела строки, а на четыре байта ниже.

  Данные целого и плавающего типа полностью совместимы с интерпретатором MSX BASIC, строковые данные на нижнем уровне — несовместимы. Поэтому следует внимательно транслировать программы с использованием самодельного доступа к телам строк и VARPTR. Впрочем, автор компилятора таких программ пока не встречал.

Глава 3. Файловые буфера

Для оптимизации дискового ввода-вывода, а также для совместимости с MSX-интерпретатором в откомпилированной программе используется буферизованный ввод–вывод. При этом каждому каналу соответствует свой буфер с соответствующими системными зонами. Зоны эти таковы:

9 байт	FCB, совместимый с MSX BASIC
256 байт	файловый буфер
38 байт	DOS FCB

Формат FCB для MSX BASIC стандартен:

+0: MOD	режим файла (0 — не открыт, 1 — чтение, 2 — запись, 4 — прямой, 8 — добавление)
+1: DOSFC	ссылка на DOS FCB
+3: LEN	длина блока для режима 4; 0 == 256 байт.
+4: DEV	код устройства (0...8 для дисков, FCh...FFh для GRP:,CRT:,LST:
+6: POS	позиция в файловом буфере
+8: PPS	позиция PRINT, возвращается функцией FPOS

Формат DOS FCB стандартный для MSX-DOS 1.03 (38 байт — для функций прямого чтения-записи)

Файловая система использует вызовы DOS 26h, 27h для записи и чтения. При прямой записи/чтении длина блока соответствует длине указанной при открытии файла и обмен происходит по 1 блоку. При последовательном доступе длина блока — 1 байт и обмен идет по 256 блоков.

Библиотека подразумевает, что файлы прямого доступа — двоичные, а последовательные — текстовые. Поэтому при закрытии файла, открытого FOR OUTPUT или FOR APPEND, в конец его дописывается символ <EOF> (26h), и добавление в конец файла FOR APPEND осуществляется не с конца файла, а с момента первой встречи символа <EOF>.

Глава 4. Использование ресурсов

Компилятор и его библиотеки построены так, что легально используют все ресурсы MSX-компьютера. Возможно, это снижает производительность, но зато дает гарантию против конфликтов с различными системными программами. Для сокращения объёмов библиотек широко используется ROM BIOS и MSX-2 SUB-ROM.

Известно, что вызовы SUB-ROM на компьютерах в некоторых конфигурациях вызывает сбои. Это происходит, в частности, на комплекте 503 + FD-051 из–за неправильно написанного дискового ПЗУ. Для ликвидации этого недоразумения в комплект поставки входит программа SLOT(© Ф.Вагапов, 1989) для эмуляции нормального переключения слотов. Эту программу достаточно запустить один раз при начальной загрузке. SLOT — некоммерческая программа, она может быть беспрепятственно скопирована и использована для любых приложений.

Все функции управления экраном, за исключением `SET SCROLL`, обращаются к ПЗУ компьютера и осуществляют интерфейс с аппаратурой, строго соблюдая спецификации MSX. Режимы `SCREEN 10,11,12` на MSX 2 + включаются также на аппаратном уровне.

Глава 5. Оптимизация

Компилятор представляет собой однопроходный транслятор с локальной многофазной оптимизацией кода. По умолчанию оптимизатор включен на полное выполнение всех операций.

Выходной ассемблер имеет команды Z-80, что позволяет в некоторых случаях даже без оптимизации получать более оптимальный код, нежели код от транслятора MSX-C.

Оптимизатор отключаемый, локальный, единицей оптимизации является строка исходного текста. В связи с этим циклы, в особенности связанные с инициализацией или простыми векторными операциями, рекомендуется

размещать на одной строке, что позволяет компилятору сильнее оптимизировать код программы.

Оптимизация производится на уровне промежуточного кода и включает простую замену и замену с проверками, склеивание константных выражений и оптимизацию управляющих конструкций.

Оптимизация гарантирует сжатие кода целочисленных и управляющих конструкций не менее чем 1,3...2 раза по сравнению с неоптимизированным кодом.

Неоптимизированный код по качеству соответствует коду ТУРБО-ПАСКАЛЯ или AZTEK-C и чрезвычайно медленный, из–за обилия стековых операций.

В результате оптимизации НЕ ГАРАНТИРУЕТСЯ ПОРЯДОК ВЫЧИСЛЕНИЯ, поэтому, если требуется точный порядок вычислений, оптимизатор можно выключить.

Кроме того, рекомендуется отключать оптимизатор при первой трансляции длинных программ, так как в таких программах наверняка найдутся ошибки, а без оптимизатора программа будет оттранслирована значительно (в 2...3 раза) быстрее.

Оптимизатор включается и выключается при помощи операторов PRAGMA и имеет 3 уровня оптимизации:

- нет оптимизации
- простая оптимизация (PRAGMA OPTIMIZE:PRAGMA NOVECT)
- мощная оптимизация: реализуется по умолчанию, работают все алгоритмы оптимизации, происходит векторизация обращений к массивам. Поэтому, если нет нужды использовать уж очень оптимальную программу, или если у вас нет большого количества операций с массивами, этот режим можно отключить, так как векторизация требует длительного (относительно) времени.

Результат тестовых прогонов
При отключённом листинге и выводе исходного текста компилятор имеет следующие параметры:

- при оптимизации: скорость в 1,5...2,5 раза больше скорости MSX-C; оптимальность кода на простых операциях лучше или равна, на сложных равна или несколько хуже оптимальности кода для MSX-C.
- Элемент ненумерованного списка без оптимизации: скорость в 3 раза больше MSX-C, качество кода 1:1 соответствует коду, порождаемому ТУРБО-ПАСКАЛЕМ (отличия состоят только в специфике ПАСКАЛЯ).

Качество кода проверялось вручную, т.е. смотрелась возможность замены конструкций на более оптимальные по скорости и занимаемой памяти для небольшой циклической программы, содержащей основные операторы языка и работу над массивами в цикле.

Касательно скорости компиляции следует заметить, что большие программы даже если и компилируются быстро, то линкуются чрезвычайно медленно. Так, программа в 530 строк на MSX BASIC (правда чрезвычайно сложная и с большим количеством ввода-вывода и строковых операций) линковалась редактором LINK в течение 17 минут.

Сравнение с MSX BASIC
При прогоне программы в целых числах, с выводом на экран и строковыми операциями во вложенных циклах большой длительности компилятор показал скоростные характеристики 7:1...(9:1) без (при) оптимизации. При выполнении программы без вывода это соотношение намного больше (30:1...50:1).

Глава 6. Раздельная компиляция

Средства поддержки раздельной компиляции обеспечивают создание модульных программ и подключение внешних модулей, написанных на С или ассемблере, а также секционирование программы внутри одного файла.

Оператор PROCEDURE, синтаксис:

```
PROCEDURE <name>
```

Объявляет последующий код до конца файла либо до следующего оператора PROCEDURE подпрограммой с глобальным именем <name>. Имя <name> соответствует соглашению об именах MSX-C.
Процедуры в одном файле разделяют все переменные, и не надо объявлять их внешними или глобальными.

Оператор GLOBAL, синтаксис:

```
GLOBAL <var1> [,<var2> ...]
```

Определяет переменные `<var1>` … глобальными.

Оператор `EXTERN`, синтаксис:

```
EXTERN <var1> [,<var2> ...]
```

Определяет переменные `<var1>` … внешними. Они должны быть также объявлены глобальными в другом модуле программы, но не в этом же файле.

Оператор `PRAGMA`, синтаксис:

```
PRAGMA <параметр>
<параметр>::={OPTIMIZE,NOOPTIMIZE,LIST,NOLIST,SOURCE,VECT
                NOVECT,NOSOURCE,WORD_DATA,BYTE_DATA}
```

Определяет режим работы компилятора:

- `OPTIMIZE/NOOPTIMIZE` — включает/выключает оптимизацию.
- `LIST/NOLIST` — включает/подавляет вывод листинга.
- `SOURCE/NOSOURCE` — включает/подавляет вывод листинга в комментарии ассемблерного текста.
- `VECT/NOVECT` — включают/подавляют векторизацию.
- `BYTE_DATA` — используются, если в операторе `DATA` во всей `WORD_DATA`-программе только числа: позволяет сэкономить память и ускорить выполнение.

ПРИМЕЧАНИЕ: `PRAGMA SOURCE` вызывает вывод листинга и на экран.

Устанавливаются по умолчанию: `VECT`, `OPTIMIZE`, `NOLIST`, `NOSOURCE`.

Приложение А. Список функций, имеющихся в данной реализации

Функции плавающей арифметики:

- `SIN`
- `COS`
- `ATN`
- `TAN`
- `LOG`
- `EXP`
- `ABS`
- `SQR`
- `CDBL`
- `CSNG`
- `RND`
- `VAL`

Целочисленные функции:

- `POS`
- `STICK`
- `STRIG`
- `PAD`
- `PDL`
- `CVI`
- `EOF`
- `RND`
- `DSKF`
- `POINT`

- LOC
- LOF
- FPOS
- LEN
- ASC
- CSRLIN
- LPOS
- INSTR
- ABS
- SGN
- SQR
- INP
- PEEK
- VPEEK
- VARPTR
- INT
- CINT
- BASE
- VDP
- TIME
- USR

Функция USR при обращении к адресам менее 8000h переходит к ROM BIOS, иначе обращение идет к памяти ОЗУ. USR может принимать один целый аргумент через HL и так же возвращает результат. Функции ABS, SQR, RND описаны в II — 3.2.3.

Строковые функции:

- LEFT$
- RIGHT$
- INKEY$
- CHR$
- INPUT$
- ATTR$
- DSKI$
- STR$
- SPACE$
- MID$
- HEX$
- OCT$
- BIN$
- STRING$

Приложение B. Наиболее вероятные причины ошибок

Ниже будут приведены основные причины возникновения пяти самых часто встречающихся ошибок.

Syntax error — Очень часто встречается, иногда второй ошибкой после какой–нибудь другой. В этом случае она исчезает по удалении первой ошибки. Кроме того часто встречается при трансляции BASIC–программ, из которых удалены все пробелы между операторов. Способ исправления — поставить где необходимо пробелы.

Type mismatch	Чаще всего встречается при совпадении имён переменных разных типов (см. II — 1.3, 3.1). Способ исправления — переименовать совпавшие переменные
Bad parameter	Следует проверить соответствие синтаксиса оператора стандарту. Может возникнуть при объявлении массива с размерностью больше двух
Assignment to FIELD	Попытка присвоить значение полю FIELD при помощи обычного присвоения. Следует заменить его на LSET
Undefined array	при попытке использовать массив, объявленный по умолчанию, что невозможно в данном компиляторе. Следует объявить массив заранее

Приложение С. Комплект поставки

В комплект поставки входят следующие файлы на диске:

BAS.BAT	пакетный файл для нормальной компиляции
VBC.COM	компилятор BASIC
M80.COM	ассемблер
LINK.COM	редактор связей
SLOT.COM	программа для исправления ошибок SLOT CALL
BASIC.DOC	данное руководство
BK.REL	ядро компилятора
ILIB.REL	главная часть библиотеки
BASEND.REL	файл окончания
BITBLT.REL	файл подпрограмм BITBLT (его нужно употреблять, если в программе использованы операторы COPY)
SAMPLE.BAS	тексты демонстрационных программ

Ссылки

BASIC и Assembler с возможностью работы в Nextore DOS

Недооцененные возможности MSX Basic

SOLiD homepage: MSX Warez Stock

http://sysadminmosaic.ru/msx/vbc/vbc?rev=1620471471

2021-05-08 13:57