Описание операторов и функций интерпретатора Fractal-BASIC-Cortex.

Фирма Фрактал
Москва Зеленоград
www.fractal.com.ru
fractal@aha.ru

Список последних изменений и добавлений Fractal-BASIC-Cortex (архив см. приложение 7)**.**

2.03.2016 v3.02o

Для функции **DMX512( )** снято ограничение на максимальный адрес принимаемых данных. Теперь можно получать данные для всего диапазона кадра DMX512. Есть ограничение на число сохраненных в буфере данных – буфер приема 136 байт и поэтому в принятом кадре максимальный адрес сохраненных данных не может быть больше минимального адреса больше чем на размер буфера. Например, максимальный адрес с которым мы работаем 500, это значит, что для принятого кадра хранятся в буфере данные для адресов с 365 по 500. И при запросе данных из этого диапазона, данные будут предоставлены сразу из буфера. При запросе адреса выходящего за текущий диапазон, первый раз будет возвращены нулевые данные, а после прихода нового кадра - реальные данные для этого адреса. При этом произойдет автоматическая корректировка диапазона с учетом нового минимального/максимального адреса.
При запросе адреса вызывающего корректировку границ принимаемых адресов, стирается весь буфер и новые валидные данные для всего нового диапазона появятся только после прихода нового кадра.
После сброса контроллера приемный буфер расположен с начала кадра, и после прихода кадра данных будет сразу доступен диапазон 0 -136.
Если данные по DMX512 не обновляются более 1с, то в соответствии с протоколом, это расценивается как потеря связи. В этом случае все данные обнуляются. И в служебный байт с адресом 0 заносится значение 0xFF. При нормальном обмене в соответствии с протоколом – мастер записывает туда 0. Это может быть использовано для диагностики работы линии. Обращение к служебному байту по адресу 0, не приводит к корректировке адресного диапазона в меньшую сторону.

5.01.2016 v3.02n

Добавлена функция **DMX512( )**. Она позволяет работать модулю в качестве slave - устройства на шине DMX512.
После первого вызова функции **DMX512( )**, канал RS485 переходит в режим работы с протоколом DMX512.
Модуль работает только на прием. При этом модуль принимает весь кадр от мастера. В текущей версии поддерживается адреса с 0 до 136 включительно. При этом пользователь может использовать информацию полностью всего кадра. В параметре функции указывается адрес и функция возвращает текущее значение переданное мастером для этого адреса. Пользователь получает возможность использования множества каналов параллельно для создания нестандартных конечных устройств, или можно получить несколько «стандартных» конечных устройств в одном приборе.
Возвращаемое значение канала находится в диапазоне 0 … 255.
После первого вызова функции массив значений всех каналов обнулен.

```
Пример:
10 x = DMX512(0) ;инициализация канала DMX512
20 pwm(8) = 10000 ; инициализация TIM8 в ШИМ с периодом 10000 мкс = 100 Гц
100 a = DMX512(15): b = DMX512(16): c = DMX512(63) ; получение значений каналов
110 pwm(81) = a / 256 ; выдача ШИМ сигнала на канал 1 TIM8
120 pwm(82) = b / 256 ; выдача ШИМ сигнала на канал 2 TIM8
130 pwm(83) = (a + b + c) / (3 * 256) ; выдача ШИМ сигнала на канал 3 TIM8
140 delay = 0.1 ; задержка на 0.1с
150 goto 100
```

20.02.2014 v3.02f

Для CAN стала доступна скорость 125 кГц.

5.02.2014 v3.02e

Для операторов PWM стал доступен TIM5.

15.12.2013 v3.02d

Добавлен оператор **LCD(200)**. Он позволяет записывать команды непосредственно в алфавитно-цифровой ЖКИ (совместимый с HD44780). Так же, он позволяет задавать вид курсора:

```
LCD(200) = 12     ;курсор выключить
LCD(200) = 14     ;курсор включить
LCD(200) = 15     ;курсор мигающий
```

Добавлен оператор **LCD(201)**. Он позволяет записывать данные непосредственно в алфавитно-цифровой ЖКИ (совместимый с HD44780).

Добавлена возможность объявления или изменения линии для управления направлением передачи драйвера RS485 для каждого из пяти UART. Для этого надо вызвать оператор RS485f#x401,(0) и указать номер пина(x – номер UART):

```
rs485f#5401,(0) = 0xCB  ;выбрать линию PC11 для UART5
rs485f#4401,(0) = 3     ;выбрать линию 3 для UART4
```

При этом , если номер пина указать = 0 , то прекратится сопровождение для этого канала.

Оглавление

Общие замечания

BASIC-программа загружается через терминальный порт - USB порт в режиме виртуального COM-порта.
Текст передается построчно в ASCII кодах, в конце строки должен присутствовать символ(ы) 0x0D(+0x0A).
Контроллер возвращает эхо-сигнал на каждый принятый символ.
Текст загружаемой программы может состоять из собственно строк программы и комментариев.
Комментарии следуют после символа «;» или оператора REM.
Комментарии при записи в модуль автоматически игнорируются и следовательно место в памяти не занимают.
Строки программы обязательно должны быть нумерованы, в начале строки должен стоять номер.
Максимальный номер строки 65535(!не путайте с количеством строк).
Максимальное количество строк зависит от типа модуля и параметров заданных в операторе MTOP.
Узнать максимальное количество строк данной конфигурации можно вызвав директиву INFO.
Число символов в одной строке(включая пробелы) не более 80. При использовании в строках программы русских символов, максимальная длинна строки уменьшается из за использования символов переключения.
!Если в процессе отладки произошло зависание программы, то нужно запустить контроллер в режиме загрузчика и набрать команду «K» (подробности см.описание соответствующего модуля).

Условные обозначения

<var> - переменная, например: ALPHA;
<expr> - выражение, например: (A+B*C)/D;
<iexpr> - выражение, принимающее только целые значения в диапазоне 0...65535
(дробная часть будет отброшена, если появится в результате вычислений);
<addr> - адрес (выражение, принимающее целые значения в диапазоне 0-65535);
<bit> - адрес бита (выражение, принимающее целые значения в диапазоне 0-255);
<len> - длина (выражение, принимающее целые значения в диапазоне 0-65535);
<text> - литерал (текстовая строка);
<cond> - условие - выражение, принимающее целое значение в диапазоне 0-65535(0000h-0FFFFh в шестнадцатеричной записи), причем нуль соответствует значению "ЛОЖНО", а все остальные значения - значению "ИСТИННО";
<opr> - оператор BASIC-а;
<ln> - номер строки – число в диапазоне 0...65535;
<i>,<j>,<n>,<m> - целые числа.
<s> - объект в операторах вывода (число, переменная, выражение, оператор).
| - "или" (возможно применение одного из 2-х приведенных обозначений).
[ ] – в прямоугольные скобки заключается необязательный параметр.

Применяемые символы

При записи программ на языке Fractal-BASIC применяются:
- десятичные цифры;
- строчные и заглавные буквы латинского алфавита;
- строчные и заглавные буквы русского алфавита(допускаются в операторах REM, строчных переменных и константах);
- знаки арифметических операций и служебные символы:

"+" - сложение;
"-" - вычитание или обозначение отрицательного числа;
"*" - умножение;
“”** - возведение в степень;
"/" - деление;
"=" - равенство;
">" - знак "больше";
"<" - знак "меньше";
">=" - знак "больше или равно";
"<=" - знак "меньше или равно";
"<>" - знак "не равно";
"**,**" - запятая;
“**( )**” - левая и правая круглые скобки;
"#" - служебный символ (направление потока вывода в канал I^2C, SPI или др.);
"**@**" - служебный символ (направление потока вывода в память);
"**:**" - двоеточие применяется для разделения операторов, записываемых в одной строке;
“**$**” - применяется для обозначения текстовых переменных.

При записи операторов, функций, имен переменных можно использовать только буквы латинского алфавита Исключение составляют примечания в операторах REM и значения строчных переменных, где можно использовать также заглавные и строчные буквы русского алфавита.

Переменные

Имена числовых переменных могут состоять из произвольных сочетаний латинских букв и цифр, но первой обязательно должна быть буква. В составе имени можно применять также символ "_" (нижний дефис).

Примеры имен: **A1; U_1; ALPHA; X(Y); VECT_X(55).**

Длина имени переменной — до 8 символов. Можно применять одно и то же имя для скалярной и индексированной переменных (**X** и **X(J)** – разные переменные).

Разрешается использовать большие и маленькие буквы. При этом регистр имеет значение.

Так, "X" и "x" – это разные переменные.

Рекомендация. Для увеличения скорости работы программы следует употреблять короткие имена.

Значения числовых переменных (кроме строчных) хранятся как числа в формате float (4-х байтовое представление **IEEE754** см. приложение 5), соответственно максимально/минимально допустимые значения числовых переменных : $\pm 2^{-126}$ / $\pm 2^{128}$.

При арифметических вычислениях и вычислении значений стандартных функций используется библиотека STMicroelectronics.

Индексированные переменные служат для образования многомерных числовых массивов. Максимальная размерность массива — 6 измерений, то есть допустим, например, массив MAS(2,3,4,4,3,2). Индекс заключается в круглые скобки. В качестве индекса может быть использовано выражение, числовое значение которого округляется усечением дробной части. Размерность числовых массивов не может быть более 255. Значение индекса может быть в пределах от 0 до 254.

Максимальное количество массивов 32.

Строчные переменные (литералы) могут быть только индексированными. Может быть объявлен один массив строчных переменных. Размер массива не более 254. Элементы массива имеют имена **$(0), $(1),… $(n)**, где **n** – размер массива. Объявленная длина строчной переменной не может быть более 254 символа. Фактическая длина может быть меньше объявленной длины. Место в памяти резервируется на объявленную длину плюс 1 байт – для размещения кода **0Dh**(конец строки).

При использовании в литералах русских букв следует помнить о том, что в длину переменной войдут коды переключения **0Eh, 0Fh**.

Константы

Числовые константы записываются непосредственно в тексте BASIC-программы. Отрицательные числа обозначаются обычным знаком "-". Допустимые формы записи:

- **41651** - целое число;
- **-12.345** – десятичная дробь (отрицательная);
- **1.2345e-1** – экспоненциальная форма представления числа;
- **0A2B3h** – шестнадцатеричное целое число. Недостающие 6 цифр заменены первыми буквами латинского алфавита: A, B, C, D, E, F. Первым символом обязательно должна быть одна из обычных десятичных цифр (0...9). В конце записи должна присутствовать буква "H" или "h".
- **0xBC34** - еше один способ записи шестнадцатиричного числа. В нем всегда первые 2 символа "0x" , а далее следует шестнадцатеричное число нужной длинны. При этом не нужно добавлять ноль перед числом начинающимся с буквы и символ "h" в конце.

Шестнадцатеричные числа Fractal BASIC воспринимает только целыми и положительными в диапазоне от 0 до 0FFFFh.

Текстовые (строчные) константы применяются для задания значений строчных переменных и вывода сообщений операторами печати. Строчная константа должна быть заключена в кавычки, например:

$(23)="Abrakadabra";

print "Сообщение на терминал".

Операторы распределения памяти BASIC-системы

DIM

Оператор **DIM <var$_1$>(n$_{11}$,n$_{12}$...n$_{1N}$), <var$_2$>(n$_{21}$,n$_{22}$...n$_{2M}$),...<var$_n$>(n$_{n1}$,n$_{n2}$...n$_{nZ}$)** резервирует место в RAM для массивов индексированных переменных <var$_1$>,<var$_2$>,...<var$_n$>. Под число отводится 4 байта. Каждый из массивов можно объявить в программе только один раз. Максимальная размерность массива – 6.
Нумерация членов массива ведется с «0».
Общее число элементов всех массивов не должно превышать 1920.

Пример:

```
;трехмерный массив MM и двухмерный TM
10 dim MM(5,7,3),TM(10,15)
```

DIM $

Оператор **DIM $(n),m** резервирует место в RAM для массива из **n** строк длиной по **m+1** байт (1 байт BASIC добавляет для размещения кода 0Dh). Следует помнить о том, что при применении русских символов, в длину строковой переменной войдут служебные коды:

- переключение на рус. регистр (0Eh);
- переключение на лат. регистр (0Fh).

Массив строковых переменных может быть только один.

Пример:

```
;строчный массив 5 строк по 20 символов
10 dim $(5), 20
```

CLEAR

Оператор **CLEAR** уничтожает все переменные и массивы.
Оператор **CLEAR <addr>,<len>** обнуляет область памяти длиной **<len>** байт начиная с адреса **<addr>.**

STATIC

Оператор **STATIC <var$_1$>,<var$_2$> ... <var$_n$>** объявляет переменные и резервирует для них место в RAM, но не инициализирует оные переменные. Оператор позволяет использовать сохраненные значения переменных после перезапуска программы после программного останова, но не сброса питания.

Операторы общего назначения

ABS

Функция **ABS(<expr>)** возвращает значение абсолютной величины <expr>.

Пример:

```
10 X=abs(Y*10)
```

AND

Оператор **<iexpr$_1$>.AND.<iexpr$_2$>** производит поразрядное логическое "И" со словами или байтами.

Пример:

```
10 X=i2c(80h).and.2Fh
```

ASC

Оператор **ASC ($(<i>),<j>) = <iexpr>** записывает в <j>-ю позицию строчной переменной $(<i>) символ с ASCII-кодом <expr>.

Функция **ASC ($(<i>),<j>)** возвращает численное значение ASCII-кода <j>-го, если считать слева, символа строчной переменной $(<i>).

Пример:

```
;если первый символ $(0) ”0”, то меняем его на пробел
10 if asc($(0),1)=30h then asc($(0),1)=20h
```

ATN

Функция **ATN(<expr>)** возвращает значение арктангенса <expr> в радианах.

Пример:

```
10 print atn(X)
```

CHR

Оператор **CHR(<iexpr> | <$(i),j)** применяется в операторах печати для вывода символов с ASCII-кодом <iexpr> или **j**-го символа строчной переменной **$(i)**.

Пример:

```
print using(##.##),chr(20h),"T1=",T1,spc(5)," T2=",T2,cr,
       \________/  \_____/  \__/  |  \___/   \__/  |  | |
         |           |        |    |   |       |    |  | |не переводит строку
         |           |        |    |   |       |    |  | перевод позиции в начало
         |           |        |    |   |       |    | переменная
         |           |        |    |   |       | строчная константа
         |           |        |    |   | печатает 5 пробелов
         |           |        |    | переменная
         |           |        | строчная константа
         |           | выводит символ с ASCII-кодом ,в данном случае 20h – «пробел»
         | устанавливает формат вывода численных значений
```

COS

Функция **COS(<expr>)** возвращает значение косинуса <expr>. Считается, что аргумент задан в радианах

Пример:

```
10 c=cos(A)
```

CR

Оператор **CR** применяется в операторах печати для перевода действующей позиции печати в исходное положение.

Пример:см. CHR

CSR

Оператор **CSR** устанавливает положение символьного курсора при выводе на ЖКИ или TFT.

По умолчанию значение CSR равно 0.

Каждый вывод на печать будет осуществляться начиная с указанной позиции до следующего изменения пользователем.

Если задать CSR = 127, то вывод будет производиться с текущей позиции , запомненной модулем после завершения прошлого вывода

DATA

Оператор **DATA <expr$_1$>,<expr$_2$>,...<expr$_n$>** служит для записи данных в тексте BASIC-программы.

Пример:

```
10 a=1:b=5
20 data a,a*2,3,b-a,b
;Читаем данные расположенные в операторе DATA
30 read v1,v2,v3,v4,v5
40 print v1,v2,v3,v4,v5 ;Выводим на терминал
;Для повторного применения оператора READ необходимо использовать оператор RESTORE
50 restore
60 read x1,x2,x3,x4,x5
70 print x1,x2,x3,x4,x5 ;Выводим на терминал
```

DO

Оператор **DO** организует циклические вычисления с выходом из цикла при выполнении определенного условия. Формат: ***DO: <opr$_1$>:<opr$_2$>:... <opr$_n$>: WHILE | UNTIL <cond>***

Пример:

```
; задержка на 0.5с
10 time=0: do: while time<0.5
```

END

Оператор **END** возвращает BASIC-систему в режим командной строки.

EXP

Функция **EXP(<expr>)** возвращает значение Неперова числа в степени <expr>.

При слишком малых и слишком больших значениях аргумента выдается сообщение "bad argument".

Пример:

```
10 input "Введите значение аргумента A = ",A
50 c=exp(A)
60 print "Значение Неперова числа в степени A - c=",c
150 goto 10
```

FOR

Оператор **FOR <var>=<expr$_1$> TO <expr$_2$> STEP <expr$_3$>** организует цикл.

где: <var> - имя переменной цикла;

<expr$_1$> - начальное значение <var> - возможно применение HEX значений;

<expr$_2$> - конечное значение <var> - возможно применение HEX значений;

<expr$_3$> - шаг изменения <var> - возможно применение HEX значений;

Циклический фрагмент программы должен завершаться оператором NEXT <var>.

Пример:

```
10 for I=1 to 100 step 2
20 print I: next I
```

GOSUB

Оператор **GOSUB <iexpr>** вызывает подпрограмму которая начинается со строки с номером соответствующим целой части значения **<iexpr>**.Возврат из подпрограммы производится оператором **RETURN.**

Пример:

```
10 X=1: Y=2: Z=3
20 gosub 100
30 print X
40 Y=Y+1: Z=Z+10
50 goto 20

100 X=sqr(Y+exp(Z))
110 return
```

GOTO

Оператор **GOTO <iexpr>** производит безусловный переход на строку с номером, соответствующим целой части значения **<iexpr>**.

Пример:см. GOSUB

IF THEN ELSE

Оператор **IF <cond> THEN <opr$_1$|ln$_1$> [ELSE <opr$_2$|ln$_2$>]** производит проверку условия <cond>,в качестве аргументов - возможно применение HEX значений.
Если <cond> истинно, выполняется оператор или производится переход на номер строки,
записанный после ключевого слова THEN. Если <cond> ложно - выполняется оператор после ELSE или, при отсутствии ключевого слова ELSE, выполняется следующая строка.

1. Условие <cond> проверяется на равенство нулю (отличное от нуля значение – "ИСТИНА"), поэтому вполне допустима запись вида:

IF <переменная> then ...

2. Результат вычисления выражений, содержащих операции сравнения, может принимать только два значения – 0 или 0FFFFh (65535).

3. Допустимо применение записей вида:

- **IF<cond> THEN <opr$_1$> <:> <opr$_2$> ...< :> <opr$_n$> ;**
- **IF<cond> THEN <opr$_1$> ELSE <opr$_2$>< :> <opr$_3$> ...< :> <opr$_n$>.**

Пример:

```
;Программа поиска slave - адресов устройств на шине I2C
11 for i=0 to 0ffh step 2
12 a=i2c#i,(0)
13 a=i2ca
14 if a=0 then phb I    ;если 0, значит slave ответил
15 next I
```

INPUT

Оператор **INPUT ["<text>",] <var$_1$>,<var$_2$>,...<var$_n$>** выводит приглашение (<text>) к вводу числовых или текстовых значений переменных <var$_1$>...<var$_n$> на терминал, после чего оператор должен ввести с клавиатуры терминала соответствующее количество значений. Числовые значения вводятся в одной строке, разделенными запятыми, а значения текстовых строк должны заканчиваться клавишей ENTER. Приглашение (<text>) может отсутствовать - в этом случае BASIC печатает "?". **При неправильном вводе выдаются сообщения "Try again", "Extra ignored"**.

Пример:

```
10 input "Введите значение аргумента A = ",A
90 t=int(A)
100 print "Целая часть значения A =",t
150 goto 10
```

INT

Функция **INT(<expr>)** возвращает целую часть значения <expr>.

Пример:см. INPUT

KEY

Функция **KEY** возвращает значение ASCII-кода клавиши, нажатой во время выполнения программы. Если ни одна из клавишей не нажата, возвращаемое значение равно нулю. Данная функция предназначена для оперативного изменения хода программы.

Пример:

```
;Программа выводит на терминал значение ASCII-кода нажатой клавиши.
10 K=key
20 if K=0 then 10
30 phb K,cr,
40 goto 10
```

LEN

Функция **LEN $(i)** возвращает длину строчной переменной $(i) в байтах.

Пример:

```
10 x=len $(0)
```

LET

Оператор **LET<var>=<expr>** присваивает значение переменной. Ключевое слово "LET" может не присутствовать.

Пример:

```
10 LET A=0FFFFH
```

LOC

Функция **LOC(<var>|$(i))** возвращает адрес, по которому во RAM располагается ранее определенная числовая или строчная переменная <var>|$(i). В случае с числовой переменной возвращенный адрес указывает на первый байт числа, а если переменная строчная, то возвращенный адрес указывает на крайний слева символ строчной переменной.

Пример:

```
10 DIM $(10),20
20 $(5)="123456"
30 print $(5)
40 MEM(LOC($(5))+2)=20H;заменим 3-й символ на пробел(счет идет с 0-го символа)
50 print $(5)
```

LOG

Функция **LOG(<expr>)** возвращает значение натурального логарифма <expr>.

NEXT

Оператор **NEXT <var>** проверяет условие выхода из цикла и, если цикл не закончен, модифицирует значение переменной <var> и передает управление на начало цикла.

Пример:см. FOR

NOT

Функция **NOT(<iexpr>)** возвращает инверсное значение двухбайтового слова.

Пример:

```
10 F=not(D)
```

ON

Оператор **ON <expr> GOTO|GOSUB <ln$_0$>,<ln$_1$>,...<ln$_n$>** организует ветвление по нескольким направлениям. Текущее значение <expr> указывает, какой из имеющихся в операторе номеров строк <ln> (по порядку записи слева направо) будет использован для перехода или вызова подпрограммы, т.е. если <expr>=0, используется <ln$_0$>, если <expr>=1, то <ln$_1$> и т.д.

Пример:

```
10 input «Выберите номер подпрограммы (от 1 до 3)»,X
20 on X gosub 100,200,300
<...>
100 <Подпрограмма №1>
<...>
200 <Подпрограмма №2>
<...>
300 <Подпрограмма №3>
<...>
```

ONERR

Оператор **ONERR <ln>** устанавливает номер строки для перехода на обработку арифметической ошибки (переполнение, потеря значимости, деление на ноль).

Пример:

```
10 onerr 100
20 Y=X/0
30 end
100 print “Арифметическая ошибка”
110 reti
```

ONINT1

Оператор **ONINT1 <ln>** устанавливает номер строки для перехода на подпрограмму обработки аппаратного прерывания линии 31(контакт 19 кросса) для MCU32.

Пример:

```
10 onint1 100
20 print RLDT : delay=0.1
30 goto 20
;Подпрограмма обработки прерывания INT1
100 print "INTERRUPT INT1" : reti
```

OR

Оператор **<iexpr$_1$>.OR.<iexpr$_2$>** производит поразрядное логическое "ИЛИ" со словами или байтами.

Пример:

```
10 X=X.or.((i2c(7).and.01h)*16)
```

PHB

Оператор **PHB [<s_1>,<s_2>,...<s_n>]** действует аналогично оператору PRINT, но выводит числовые значения на терминал в целочисленном шестнадцатеричном виде. Числовые значения от 0 по 255 выводятся в виде двух HEX цифр (при необходимости печатается левый нуль). Значения от 256 по 65535 выводятся в виде 3-х или 4-х HEX цифр.

Пример:

```
phb using(##.##),chr(20h),"T1=",T1,spc(5)," T2=",T2,cr,
     \________/  \_____/  \__/  |  \___/   \__/  |  | |
      |           |        |    |   |       |    |  | |не переводит строку
      |           |        |    |   |       |    |  | перевод позиции в начало
      |           |        |    |   |       |    | переменная
      |           |        |    |   |       | строчная константа
      |           |        |    |   | печатает 5 пробелов
      |           |        |    | переменная
      |           |        | строчная константа
      |           | выводит символ с ASCII-кодом ,в данном случае 20h – «пробел»
      | устанавливает формат вывода численных значений
```

PHW

Оператор **PHW [<s_1>,<s_2>,...<s_n>]** действует аналогично оператору PRINT, но выводит числовые значения на терминал в целочисленном шестнадцатеричном виде. Числовые значения от 0 по 65535 выводятся в виде 4-х HEX цифр (левые нули печатаются).

PI

Функция **PI** возвращает значение числа "пи" (3.1415926).

Пример:

```
10 print "Синус 45 градусов=",sin(pi/4)
```

POP

Оператор **POP <var_n>,<var_{n-1}>,...<var_1>** уменьшает указатель вычислительного стека, снимает значения с верхушки стека и присваивает их переменным.

Примечание: операторы PUSH и POP применяются, в основном, для обмена числовыми данными с подпрограммами. Следует обратить внимание на то, что оператор POP считывает из стека данные в обратном порядке.

Пример:

```
5  X=1: Y=1: Z=1: N=1: M=1:
10 A=1: B=3: C=4: D=10
20 push A,10,B,D,C
30 print X,Y,Z,N,M
40 pop Y,M,X
50 print X,Y,Z,N,M
60 pop Z,N
70 print X,Y,Z,N,M
```

PRINT

Оператор **PRINT [<s_1>,<s_2>,...<s_n>]** выводит информацию на терминал. где: **<s_1>,<s_2>,...<s_n>** - объекты вывода. Объекты вывода должны быть разделены запятыми. В качестве объектов вывода могут быть применены:

- число;
- переменная;
- выражение;
- строчная константа (текст в кавычках);
- строчная переменная **$(i)**;
- оператор **USING(<form>)**;
- оператор **TAB(<iexpr>)**;
- оператор **CR**;
- оператор **SPC(<iexpr>)**;
- оператор **CHR(<iexpr> | <$(i),j)).**

При вводе текста программы вместо ключевого слова "**PRINT**" можно употреблять сокращения - "**P.**" или "**?**". При отсутствии объектов вывода оператор просто переводит строку. Если строку переводить не надо, то после **<s_n>** следует поставить запятую.

PUSH

Оператор **PUSH <$expr_1$>,<$expr_2$>,...<$expr_n$>** помещает вычисленные значения выражений в вычислительный стек увеличивая, соответственно, указатель стека.
Примечание: операторы PUSH и POP применяются, в основном, для обмена числовыми данными с подпрограммами. Следует обратить внимание на то, что оператор POP считывает из стека данные в обратном порядке.

Пример: см. POP

READ

Оператор **READ <var_1>,<var_2>,...<var_n>** производит последовательное присваивание переменным <var> значений выражений, записанных в операторе DATA. Примечание: количество считываемых данных не должно превышать количество данных, содержащихся в последнем из выполненных операторе DATA.

Пример: см. DATA

REM

Оператор **REM <text>** предназначен для записи комментариев в BASIC-программе.
Разрешено использование латинских и русских букв в обоих регистрах.

RESTORE

Оператор **RESTORE** устанавливает в исходное состояние указатель чтения операторов READ, если необходимо повторное считывание данных из оператора DATA.

Пример: см. DATA

RETI

Оператор **RETI** возвращает управление в прерванную программу (см. ONTIME, ONERR, ONINT1).

Пример:

```
10 onerr 100
20 Y=X/0
30 end
100 print "Арифметическая ошибка"
110 reti
```

RETURN

Оператор **RETURN** возвращает управление на строку, следующую за строкой, из которой производился вызов подпрограммы оператором **GOSUB.** *Пример:*

```
10 X=5
20 gosub 100
30 print Y
40 end
100 Y=sin(X)+cos(X/3)
110 return
```

RND

Функция **RND** возвращает псевдослучайное число в диапазоне (0... 1.0). *Пример:*

```
10 X=rnd
```

ROT#

Функция **ROT#<n>, (<iexpr>)** возвращает *циклически* сдвинутое влево на **<n>** разрядов значение **<iexpr>.**
<n> может задаваться константой, переменной или выражением и принимать значения от 0 до 15.
<iexpr> рассматривается как целое число в диапазоне 0…0xFFFF. Производится сдвиг 16-битного слова. Если **<iexpr>** больше 0xFFFF, то выдается сообщение об ошибке. *Пример:*

```
10 N = 12
20 B = rot#N,(A)
```

RST

Оператор RST производит перезапуск BASIC-системы.

SGN

Функция **SGN(<expr>)** возвращает значение знака величины <expr>.
Если значение <expr> положитнльное, возвращается 1.
Если значение <expr> равно нулю, возвращается 0.
Если значение <expr> отрицательное, возвращается -1. *Пример:*

```
10 input "Введите значение аргумента A = ",A
110 R=sgn(A)
120 print "Определяет знак A ( если '-'=-1: '+'=1: 0=0)=",R
150 goto 10
```

SIN

Функция **SIN(<expr>)** возвращает значение синуса <expr>. Считается, что аргумент задан в радианах.

SPC

Оператор **SPC(<expr>)** применяется в операторах печати для вывода пробелов в количестве <expr>. Максимальное количество пробелов =80. *Пример: см. PHB*
Также, оператор используется для управления запретом/разрешением вывода обрамляющих пробелов при печати числа. Параллельно с отключением вывода лишних пробелов запрещается вывод знака вопроса с последующим выводом числа(в случае когда число не вписывается в формат заданный оператором USING). Теперь в случае выхода числа за пределы формата, будут выводится символы "#" в форме заданной оператором USING. Данная возможность будет полезна при особенно работе с ЖКИ.

Для отключения вывода лишних пробелов и печати неформатного числа, необходимо в операторе SPC указать параметр 100. Для включения вывода пробелов необходимо в операторе SPC указать параметр 101.

По умолчанию вывод ведется как и раньше - с пробелами и печатью неформатных чисел.
Примеры:

```
print spc(100) ;отключить вывод пробелов и вывод неформатных чисел
print spc(101) ;включить вывод пробелов и вывод неформатных чисел
```

SQR

Функция **SQR(<expr>)** возвращает значение корня квадратного <expr.> При отрицательном значении аргумента выдается сообщение "bad argument". *Пример:*

```
10 A=sqr(B)
```

STOP

Оператор **STOP** останавливает выполнение BASIC-программы. На терминал выводится сообщение "Stop - in line <N>", где <N> - номер строки, в которой расположен следующий оператор.

TAB

Оператор **TAB(<iexpr>)** применяется в операторах печати для установки действующей позиции печати в позицию <iexpr>.

TAN

Функция **TAN(<expr>)** возвращает значение тангенса <expr>. Считается, что аргумент задан в радианах. При неправильно заданном аргументе выдается сообщение "divide by zero". *Пример:*

```
10 input "Введите значение аргумента A(в радианах) - ",A
70 T=tan(A)
80 print "Значение TAN(A)=",T
100 goto 10
```

UNTIL

Оператор **UNTIL <cond>** проверяет истинность условие <cond> и, если условие не соблюдено(<cond>=0), производит передачу управления к началу цикла. В противном случае выполняется следующий за UNTIL оператор. Применяется в паре соператором **DO.** *Пример:*

```
10 time=0: do: until time>0.5       ;задержка 0.5 сек.
```

USING

Оператор **USING(<form>)** устанавливает формат вывода численных значений.

<form>= 0 задает свободный формат. Число будет напечатано в виде десятичной дроби или, при необходимости, в экспоненциальной форме.

<form>= F<n> - экспоненциальная форма с **<n>** значащими цифр.

<form>= ##.## - шаблонная форма. Количества символов "#" до и после десятичной точки устанавливают количества знакомест для целой и дробной частей выводимого числа.

Если число не укладывается в шаблон, печатается "**?**", а число выводится в свободном формате. В случае если предварительно был сделан запрет на вывод чисел не вписавшихся в заданный формат(оператор SPC(100)), будут выведены символы "#" в формате заданным USING.

Общее количество символов "**#**" не должно быть больше 7. Установленный формат вывода действует до последующего применения оператора **USING** (в пределах оператора **PRINT**). Вместо ключевого слова "**USING**" можно употреблять сокращение - "**U.**".

Пример: см. PHB

VAL

Функция **VAL $(i)** возвращает численное значение строчной переменной, представляющей собой запись числа. Если в переменной содержатся посторонние символы, выдается сообщение "bad argument".

WDOG

Оператор **WDOG** управляет режимом работы сторожевого таймера(встроенного в микроконтроллер узла Independent watchdog). После сброса микроконтроллера работа этого узла запрещена. Пользователь, при необходимости, может разрешить его работу, указав период в секундах срабатывания watchdog. При этом в программе необходимо будет предусмотреть периодический сброс watchdog гарантировано чаще чем было указано при инициализации оператора. Если произойдет переполнение watchdog , то модуль будет сброшен и программа запустится сначала. Диапазон задаваемых времен 0.01 … 26с. По умолчанию принято значение =1с. Узел watchdog имеет собственный тактовый генератор. У этого генератора довольно большой разброс по опорной частоте – приблизительно до 1.5 раз в обе стороны, это надо учитывать при выборе значения оператора WDOG.
В теле оператора DELAY watchdog тоже не сбрасывается, поэтому необходимо учитывать и суммарное время нахождения программы DELAY.

```
        Пример использования:
17 WDOG = 2.5     ;зададим критическое время =2.5с
…
1458 WDOG   ;сброс, должен выполняться заведомо чаще чем 2.5с
…
```

WHILE

Оператор **WHILE <cond>** проверяет истинность условие <cond> и, если условие соблюдено (<cond>=1), производит передачу управления к началу цикла. В противном случае выполняется следующий за WHILE оператор. Применяется в паре соператором **DO.** *Пример:*

```
; задержка 0.5 сек.
10 time = 0: do: while time < 0.5
```

XOR

Оператор **<iexpr$_1$>.XOR.<iexpr$_2$>** производит поразрядное логическое "исключающее ИЛИ" со словами или байтами. *Пример:*

```
10 X = X.xor.((i2c(7).and.01h)*16)
```

Работа с USB, UART, RS485 / MODBUS

В зависимости от типа модуля, могут быть доступны до 5-ти последовательных UART и терминальный USB каналы. Каналы UART1 и 3 имеют как правило физический интерфейс RS485. Каналы UART2, 4, 5 доступны на линиях ввода-вывода с уровнями 0...+3.3В и по умолчанию не проинициализированы. Но при первом обращении к ним, происходит автоматическая инициализация и после этого они могут работать с операторами/функциями RS485, PUT$, GET$.

Все каналы обрабатываются в полудуплексном режиме, т.е. в каждый момент времени, канал либо принимает, либо передает. У каждого канала свой буфер на 137 байт. Это обеспечивает прием-отправку до 128 байт данных в одном пакете. Все каналы работают полностью независимо и прозрачно.

Поддерживается работа каналов RS485 по протоколу MODBUS и в качестве мастера и в качестве slave-узла. Кроме этого поддерживается обмен оператором PUT$ и функцией GET$ всех каналов в посимвольном и пакетных режимах.

При работе качестве slave-узла MODBUS, поддерживается обработка 12 команд, обеспечивающих доступ к пользовательским областям ресурсов (RAM, Reg, EEPROM, Flash, ID, I2C, SPI).

При приеме и отправке пакетов в режиме MODBUS автоматически производится подсчет и проверка CRC16.

В соответствии с протоколом MODBUS, обмен информацией между мастером и slave производится пакетами только по инициативе мастера. Т.е. slave только отвечает мастеру на его запросы. За пакет принимается любая последовательная передача данных, следующая после паузы не менее 3.5 байт на текущей скорости обмена, с паузами между байтами не более 1.5 байт, и до ближайшей паузы более 3.5 байт.

Пакеты длиннее 137 байт игнорируются. Причем числом 137 ограничена общая длинна пакета. Поскольку в каждый отдельный момент времени контроллер работает либо на прием, либо на передачу, используется один буфер и для приема и для передачи.

Терминальный USB канал модуля поддерживает работу по протоколу MODBUS в качестве slave-узла. Терминальный канал может находиться в двух устойчивых состояниях: посимвольном для работы с BASIC-Terminal и пакетном для работы PIC18Terminal (MODBUS). Переключение между каналами осуществляется записью в ячейку EEPROM(202h) значения 0 для побайтного режима или значения 1…255 для пакетного режима. Это значение одновременно является адресом MODBUS для канала USB. Записанные изменения вступят в силу только после сброса питания. Новые модули поставляются с предустановленным побайтным режимом для USB канала.

!Обратите внимание: поскольку оператор / функция EEPROM оперирует с 16-битными словами(2 байта), то при адресации через MODBUS, где используется побайтовая адресация, нужно указывать вдвое больший адрес и при записи нового значения прописывать незначащий старший байт = 0.

```
Пример изменения режима из BASIC на пакетный, с присвоением адреса MODBUS = 0x12:
   EEPROM(202h) = 12h
Пример изменения режима из пакетного на побайтный:
   12 75 04 04 02 00 00 CRC16
```

Работа в качестве Slave устройства в сети MODBUS.

Первый байт в пакете запроса мастера - всегда адрес slave-узла.

Slave-adr на который будет отвечать модуль хранится в ячейках EEPROM(200h) UART3 и EEPROM(203h) UART1, производственная установка 4 и 2 соответственно.

Адрес может быть изменен пользователем оператором EEPROM и вступает в силу после сброса питания.

Кроме своего индивидуального адреса обрабатываются команды с циркулярным адресом, равным 0. Это удобно для передачи мастером информации во все узлы одновременно.

Второй байт в пришедшей телеграмме рассматривается как команда.

Далее, как правило, идут два байта адреса ресурса внутри контроллера - сначала старший, потом младший.

Следующий байт пакета в большинстве команд это число читаемых/записываемых байт данных.

Далее следуют собственно данные, если это предусмотрено командой.

Последние 2 байта пакета всегда интерпретируются как контрольная сумма CRC16. При несовпадении CRC16 запрос мастера игнорируется, ответный пакет не формируется. Так же ответ не формируется на запросы с циркулярным адресом т.е. при работе с циркулярным адресом имеют смысл только команды записи.

Если адрес узла совпал и CRC16 соответствует пакету, то производится разбор команды. В этом случае контроллер либо даст соответствующий запросу ответ, либо квитанцию с кодом ошибки. Ошибка выдается при приеме несуществующей команды(если не разрешена обработка телеграммы в пользовательской программе) или несоответствии параметров запроса команде. Например: в команде записи указано, что будет записываться 5 байт, а передано 6 байт данных.

В ответном пакете контроллер сохраняет структуру запроса. Первый байт ответа это свой адрес. Второй код команды. Третий и четвертый адрес ресурса. Пятый количество байт читаемых/записываемых данных или номер бита в битовых операциях. Последние 2 байта это CRC16 посчитанное контроллером для ответного пакета. При обнаружении несоответствия параметров конкретной команды или при несуществующей команде, контроллер формирует квитанцию об ошибке: первый байт - свой адрес, второй команда запроса с установленным в “1” старшим битом, третий - код ошибки, четвертый-пятый байты - CRC16.

Поддержанные в интерпретаторе команды аналогичны применяемым в наших модулях серий MCU4 и MCX5x. Перечень и подробное описание команд приведены в приложении 6.

Работа контроллера в качестве мастера в сети MODBUS.

RS485

Для обеспечения доступа ко всем каналам RS485 в операторе / функции RS485 предусмотрена адресация: для канала UART3 применяются адреса без изменений, для остальных каналов UART1, 2, 4, 5 указывается адрес соответственно на 1000, 2000, 4000, 5000 больше.

Операторы/функции RS485 осуществляют обмен по каналу RS485 с использованием протокола MODBUS. Каждый обмен всегда инициируется мастером и состоит из пакетов запроса мастера и ответа slave.
Пакеты содержат адрес slave-устройства, код кооперации, адрес ресурса внутри slave, число байт данных, данные и сопровождаются контрольной суммой CRC16. Операторы/функции RS485 рассчитаны на работу с изделиями разработки фирмы Фрактал и используют команды MOSBUS из резервной области(0x70 и 0x71). Это позволяет делать пакеты максимально компактными, а следовательно быстро передающимися.
!Обратите внимание, поскольку операторы/функции RS485 используют передачу данных по последовательному каналу, да еще и в двух пакетах, необходимо учитывать, что время их выполнения довольно велико. Оно в основном определяется выбранной скоростью обмена по RS485 и числом принимаемых/передаваемых байт данных.
Для уменьшения затрачиваемого времени рекомендуется, по возможности, записывать группы данных в одном пакете.

Оператор **RS485# <SlaveAddress>** устанавливает текущее значение Slave Address для операций ввода/вывода с помощью оператора **RS485**. Значения Slave Address могут быть в диапазоне (0...255).

Функция **RS485#** возвращает текущее значение Slave Address.

Функция **RS485A** возвращает диагностику последнего обмена оператора / функции RS485.
Если обмен прошел штатно (не было ошибок, совпал CRC16 и все параметры), будет возвращен 0.
В противном случае будет возвращен код ошибки. Код 0FFh возвращается в случае превышения timeout отведенного на ожидание ответа (10мс).

Оператор **RS485 [W/F][#<SlaveAddress>,](<WordAddress>)=<iexpr$_1$>,<iexpr$_2$>,... <iexpr$_n$>** обеспечивает доставку устройству с адресом <SlaveAddress> в одном пакете по протоколу MODBUS значений <iexpr$_1$>,..<iexpr$_n$>. Передаваемые данные <iexpr> размещаются по адресу <WordAddress>.
Значение <SlaveAddress> может быть в пределах 0...255.
<WordAddress> - адрес слова в адресном пространстве устройства **RS485** - может принимать значения 0...65535.
При наличии модификатора **[W]** выражения **<iexpr$_1$>,<iexpr$_2$>,... <iexpr$_n$>** рассматриваются как слова из 2-х байтов и могут принимать значение 0...65535. Если модификатор отсутствует, то когда значение <expr> меньше 256, в канал I^2C передается 1 байт, в противном случае – 2 байта. После исполнения оператора значение <SlaveAddress> остается в системной переменной и, если нет необходимости в его замене, при следующем использовании оператора может быть опущено: **RS485 (<WordAddress>)=<iexpr$_1$>,<iexpr$_2$>,... <iexpr$_n$>.**
При наличии модификатора **[F]** объектом обмена является переменная(4 байта).

Функция **RS485 [W/F][#<SlaveAddress>,(<WordAddress>)]** возвращает численное значение байта, слова или переменной, считанных в устройстве **RS485** по адресу **<WordAddress>**.

```
Пример записи слова из 2 байт в ячейку 0 узла MODBUS с адресом 2 с последующим
прочтением и печатью одного старшего байта:
   RS485W#2,(0) = 1234h
   PHB RS485(1)
```

Для изменения параметров UART надо применить оператор **RS485F#x400,(0) =** где x, это номер соответствующего UART(1...5) **.** Причем UART3 соответствуют и 0 и 3.
После знака равенства перечислить скорость(в бодах), битность(8), количество стоп бит(1 или 2), четность(0-отсутствует, 1-нечетность, 2-четность). При вызове необходимо перечислить обязательно все 4 параметра. Новые значения вступают в силу сразу и действуют до нового изменения или сброса питания. По включению питания всегда устанавливаются: 115200 бод, 8 бит, 1 стоп, без четности.
Так например, для канала UART1 варианта скорости 57600, 8-ми бит, одного стопового, без четности надо вызвать:

```
RS485F#1400,(0) = 57600, 8, 1, 0
```

Для объявления или изменения линии для управления направлением передачи драйвера RS485 надо вызвать оператор RS485f#x401,(0) и указать номер пина(x – номер UART):

```
rs485f#5401,(0) = 0xCB  ;выбрать линию PC11 для UART5
rs485f#4401,(0) = 3     ;выбрать линию 3 для UART4
```

При этом , если номер пина указать = 0 , то прекратится сопровождение для этого канала.

PUT$

Оператор **PUT$ [<addr>,<len>]** посылает в последовательный канал UARTx <len> байт из RAM начиная с адреса <addr>. При таком формате параметров производится работа с каналом UART3.
Если же вместо адреса указаны номера каналов 0...5(0 эквивалентен 3), то посылка происходит из буфера предварительно указанного для соответствующего канала при его инициализации.
Инициализация канала производится вызовом оператора PUT$ с адресом 0...5 (соответствует каналу) и указанием в параметре **<len>** адреса буфера с которым будет вестись обмен.
!При первом обращении оператора к выбранному каналу, обязательно долен быть проинициализирован адрес передающего буфера. Для каждого канала адрес свой! Адреса приемного и передающего буфера разные!
Для всех каналов можно установить режим работы функции GET$ пакетный(принимается слитная группа байт) или побайтный(производится накопление байт). Для установки режима –> **<addr>** =номер канала, **<len>** = 0x100 для пакетного режима и 0x101 для побайтного.
При обработке PUT$ телеграмма посылается в одном пакете(гарантировано слитно). При этом передача идет фоном по отношению к бейсик программе, т.е. оператор возвращается в программу сразу после начала передачи. Если будет произведена попытка повторного входа в него при не закончившейся прошлой передаче, будет произведено ожидание завершения прошлой передачи и только после этого будет начата новая.
Параметры в команде необязательны - в случае отсутствия оных (одного или нескольких) используются последние явно указанные (в предыдущих операторах PUT$). Но по включению питания эти параметры не установлены, поэтому при первом использовании оператора в программе их следует указать.
Применение оператора запрещает работу канала RS485 в режиме slave MODBUS до применения оператора / функции RS485. Примеры:

```
PUT$ ADR, N; посылка в канал UART3  N байт начиная с адреса ADR:
PUT$ 0,ADR1; Инициализация канала UART3 с указателем ADR1
PUT$ 2,0x100; Инициализация канала UART2 с таймаутом
PUT$ 1,0x101; Инициализация канала UART1 без таймаута
PUT$ 4,5; канал UART4, послать 5 байт из предварительно указанного буфера
```

GET$

Функция GET$ **[<addr>,<len>]** позволяет получить доступ к пришедшим по UART-м данным.
Данные могут принимаются как в пакетном режиме, так и в побайтном. В пакетном режиме(по умолчанию) принимается слитный пакет до паузы 3.5 байта, после чего прием останавливается до вызова GET$.
В побайтном режиме принимаются и накапливаются все пришедшие байты вплоть до вызова GET$.
Размер общего буфера, как уже упоминалось, 137 байт.
После вызова GET$ производится очистка внутреннего буфера.
Параметры **<addr>,<len>** и инициализация аналогичны PUT$. **<addr>** - адрес буфера(только канал UART3) или номер канала(при этом адрес буфера указывается при инициализации) в котором будут размещены данные, **<len>**– число байт которое будет взято из буфера. GET$ возвращает число равное реально пришедшему числу байт.
!При первом обращении функции к выбранному каналу, обязательно долен быть проинициализирован адрес приемного буфера. Для каждого канала адрес свой! Адреса приемного и передающего буфера разные!
!Обратите внимание, если заданное ожидаемое число байт меньше реально пришедшего количества, то часть данных будет утеряна, скопируется только запрошенное количество.
Для правильной работы GET$ необходимо переключиться в режим непротокольного обмена, для этого достаточно хотя бы раз использовать PUT$ или GET$.
Применение функции запрещает работу канала в режиме slave MODBUS до применения оператора / функции RS485.
Пример запроса о наличии сообщения длинной N, которое будет расположено с адреса ADR, количество байт в пришедшем пакете буде помещено в X (для канала UART3):

```
X = GET$ ADR, N
Указать адрес приемного буфера ADR2 для канала UART1
x = GET$ 1, ADR2
Канал UART1, проверить сколько пришло и передать в буфер 7 байт
X = GET$ 1,7
```

Работа с шиной I^2C

Интерпретатор поддерживает работу с шиной в режимах **multimaster** и **slave**.
При этом, первый байт пакета расценивается как slave-адрес I2C-устройства и признак чтения/записи. Второй *байт* при записи интерпретируется как адрес ячейки внутри I2C-устройства(адрес слова – word-адрес).

Режим SLAVE

В этом режиме модуль доступен по двум адресам. По одному адресу осуществляется доступ к пользовательской RAM, по другому к страничным регистрам. Доступ к пользовательской RAM делает доступным обмен непосредственно с переменными и массивами интерпретатора. Так к примеру, чтобы прочитать первую объявленную переменную бейсика, надо прочитать первые 4 ячейки RAM(0x0000-0x0003).
Поскольку word-адрес имеет размер всего один байт, без переключения страничного регистра "видны" только 256 байт. Для доступа ко всему объему пользовательской RAM сделана возможность записи смещения к этому адресу путем записи величины смещения в страничный регистр(указатель). Причем используется два разных указателя – отдельно для записи и отдельно для чтения. Это позволяет при необходимости «развести» буфера чтения и записи в разные области.
После сброса оба указателя равны нулю, что означает - выбранные страницы записи и чтения совпадают и расположены в самом начале пользовательской области.
Word-адрес указателя для буфера записи = 0.
Word-адрес указателя для буфера чтения = 4.
Заводские установки – для доступа к данным slave - адрес 0x02, для доступа к управляющим регистрам 0x04.
Пользователь может изменить эти адреса , при этом они сохраняются в EEPROM(адреса могут быть только четные!) :

```
Для установления адреса доступа к данным ->     I2C#256,(0) = NewAdr
Для установления адреса доступа к регистрам ->  I2C#258,(0) = NewAdr
```

Режим MASTER

Этот режим позволяет интерпретатору инициировать обмен информацией с модулями подключенными к шине I2C. Для этого используются соответствующие операторы и функции:

I2C#

Оператор **I2C# <SlaveAddress>** устанавливает текущее значение Slave Address для операций ввода/вывода с помощью оператора **I2C**. Значения Slave Address могут быть только четными в диапазоне (0...254).

Функция **I2C#** возвращает текущее значение Slave Address.

Функция **I2CA** возвращает "1", если в процессе обмена с устройством I^2C отсутствовало подтверждение или был превышен лимит времени ожидания, “0” свидетельствует о нормально произведенном обмене.

Оператор **I2C[W/F][#<SlaveAddress>,](<WordAddress>) (=<iexpr$_1$>,<iexpr$_2$>,... <iexpr$_n$>)** производит вывод в канал I^2C. Значение Slave Address может быть только четное в пределах 0...254. <WordAddress> - адрес слова в адресном пространстве устройства I^2C - может принимать значения 0...65535; если значение <addr> меньше 256, то в канал I^2C передается только 1 байт; <expr$_1$>,<expr$_2$>,... <expr$_n$> - данные для записи в канал I^2C. При наличии модификатора **[W]** выражения **<iexpr$_1$>,<iexpr$_2$>,... <iexpr$_n$>** рассматриваются как слова из 2-х байтов и могут принимать значение 0...65535. Если модификатор отсутствует, то когда значение <expr> меньше 256, в канал I^2C передается 1 байт, в противном случае – 2 байта. После исполнения оператора значение Slave Address остается в системной переменной и, если нет необходимости в его замене, при следующем использовании оператора может быть опущено: **I2C(<WordAddress>)=<iexpr$_1$>,<iexpr$_2$>,... <iexpr$_n$>.**
При наличии модификатора **[F]** объектом обмена является переменная(4 байта). В режиме slave-I2C обеспечивается целостность значения переменной. Пользователь может быть уверенным в том, что несмотря на распределенный во времени обмен, все 4 байта гарантировано из одного комплекта. Обязательным условием соблюдения целостности является размещение переменных по адресам кратным 4-м байтам, т.е. адрес переменной обязательно должен содержать в двух младших битах «0».

Функция **I2C[W/F][#<SlaveAddress>,(<WordAddress>)]** возвращает численное значение байта, слова или переменной, считанных в устройстве I2C по адресу **<WordAddress>**. Если значение выражения **<WordAddress>** меньше 256, то в канал передается 1 байт. Если значение **<WordAddress>** лежит в диапазоне от 256 по 65535 то оно передается в виде 2-х байтов. Комбинации Start и Slave Address выдаются дважды - на запись и на чтение. Когда **[#<SlaveAddress>, (<WordAddress>)]** отсутствуют, производится безадресное чтение - в канал передается текущее значение (из системной переменной) Slave Address с установленным битом чтения и считывается байт по текущему адресу.

Оператор **I2CB<n>[#<SlaveAddress>,](<WordAddress>)= <iexpr$_1$>, <iexpr$_2$>, ...<iexpr$_n$>** производит установку или сброс битов в байте, адресуемом Slave Address и Word Address. Установка или сброс производятся начиная с бита номер **<n>.** Число **<n>** может быть в диапазоне 0…7. Биты нумеруются начиная с младшего. Если **<iexpr>** = 0, то бит сбрасывается, в противном случае - устанавливается. Операция производится по принципу: «чтение-модификация-запись».

Функция **I2CB<n>[#<SlaveAddress>,](<WordAddress>)** возвращает значение бита в байте, находящемся в устройстве I2C. Функция возвращает значения “0’ или “1”.

Изменить скорость обмена для мастера по I2C можно вызвав оператор, указав Slave Address 400 и параметр:
- для 100 КГц нужно вызвать оператор `I2C#400,(0) = 0`
- для 400 КГц нужно вызвать оператор `I2C#400,(0) = 1`

После сброса устанавливается максимальная скорость обмена – 400 КГц.

При "зависании" шины, микроконтроллер по прошествии величины времени TIMEOUT * 100мкс реинициализирует работу шины. При этом сохраняется скорость работы канала, установленная на момент зависания шины. Для изменения величины TIMEOUT необходимо выполнить оператор `I2C#300,(0) = TIMEOUT`.
По умолчанию TIMEOUT равен 20, что соответствует 2мс.

Пример:
```
10 A=i2c#10h,(0)* i2c(1)      ;в модуле 10h перемножаем ячейки 0 и 1
;в зависимости от результата в этом модуле в ячейке 2 сохраняем бит 3
20 if A>100 then i2cb3(2)=1 else i2cb3(2)=0
30 if i2ca=0 then print”OK” else print”ERROR”;проверим подтверждение обмена
```

CSR

Оператор **CSR=<iexpr>** устанавливает исходное положение действующей позиции печати при выводе в канал I^2C. Нулевое значение соответствует крайней левой позиции. Значение <expr> может быть в пределах от 0 до 127 и перед выводом в канал I^2C суммируется с константой 80h.

PRINT#

Оператор **PRINT#[<iexpr>] , <s_1>,<s_2>,...<s_n>** выводит информацию в канал I^2C, SPI, или на ЖКИ. Объекты вывода как у оператора PRINT. Значение **<iexpr>** определяет адрес выходного устройства и может находиться в пределах 0...65535.

Если <iexpr> больше 0FFh, то оператор выводит поток в канал I2C, старший байт – Slave Address, младший байт Word Address. При зависании канала I^2C вывод прекращается. Факт зависания можно проверить, прочитав значение, возвращенное функцией I2CA (если канал завис, I2CA=1).

Если <iexpr> 0…0Fh, вывод идет в SPI. При <iexpr> 0…7 это значение перед началом обмена выставляется на линии адреса SPI и держится весь обмен до завершения оператора, после чего на линии адреса выдается состояние 7, что соответствует всем единицам. При <iexpr>=0Fh, обмен будет производиться без изменения состояния линий адреса.

Если <iexpr> = 10h, вывод идет на ЖКИ. Если речь идет о внешем символьном ЖКИ, подключаемым к линиям ввода/вывода, то до первого вывода на ЖКИ нужно применить оператор LCD. Для вариантов со встроенными ЖКИ инициализация не требуется.

После разбора с адресом и направлением вывода, в канал всегда выдается байт, определяющий позицию курсора; значение этого байта – смещенная на 80h величина, заданная оператором CSR. По умолчанию CSR=0. Если [<iexpr>] опустить, то вывод произойдет по прошлому значению <iexpr>.

PHB#

Оператор **PHB# [<iexpr>] , [<s_1>,<s_2>,...<s_n>]** аналогичен PHB, но выводит в канал I^2C.

PHW#

Оператор **PHW# [<iexpr>] , [<s_1>,<s_2>,...<s_n>]** аналогичен PHW, но выводит в канал I^2C.

Операторы для работы с шиной SPI

SPI

Если в функции или операторе присутствует необязательный параметр <CS> и его значение 0…7 то производится инициализация адресных линий - контактов кросса 2-4, эти линии настраиваются на выход и на них выдается заданная комбинация. Младший бит соответствует контакту 2, старший 4. После отработки оператора исходное состояние линий не восстанавливается.
До момента первого объявления <CS> = 0…7 эти линии находятся в третьем состоянии и могут быть использованы для других функций.

Оператор **SPI [W] [#<CS>,] [(<WordAddress>)] = <iexpr$_1$>,<iexpr$_2$>,... <iexpr$_n$>** производит вывод в канал SPI. <WordAddress> - адрес слова в адресном пространстве устройства SPI - может принимать значения 0...65535; если значение <addr> меньше 256, то в канал SPI передается только 1 байт; в противном случае – 2 байта, причем старший байт передается первым; <expr$_1$>,<expr$_2$>,... <expr$_n$> - данные для записи в канал SPI. При наличии модификатора **[W]** выражения **<iexpr$_1$>,<iexpr$_2$>,... <iexpr$_n$>**, рассматриваются как слова из 2-х байтов и могут принимать значение 0...65535. Если модификатор отсутствует, то когда значение <expr> меньше 256, в канал SPI передается 1 байт, в противном случае – 2 байта.
Оператор SPI теперь позволяет легко изменить номер режима по полярности и фазе тактового сигнала.
Режим по умолчанию №2.
Для смены режима нужно обратиться к несуществующим устройствам с адресами 100…103 соответственно.

```
Пример  - установить режим 1 :      spi#101 = 0
```

Функция **SPI [W] [#<CS>,] [(<WordAddress>)]** возвращает численное значение байта или слова, считанного в устройстве SPI по адресу **<WordAddress>**. После отработки функции линии устанавливаются в верхний уровень. Значение выражения **<WordAddress>** передается в канал SPI одним или двумя байтами, как и в операторе. Когда отсутствует **(<WordAddress>)**, производится безадресное чтение (только прием).

Функция **SPIB<n>[#<CS>,](<WordAddress>)** возвращает значение бита в байте, находящемся в устройстве SPI.Число <n>может быть в пределах от 0 до 7.Функция возвращает значения “0’ или “1”.

```
;Пример программы для работы с модулями ADC4-1.x
;Модули подключаются в без адресном режиме(без CS)
new
;сброс канала передачи
10 spi#7 = 0, 0FFh, 0FFh, 0FFh, 0FFh
;запуск преобразования Single Conversion Mode 16bit
20 spi = 38h, 40h
;пауза
30 delay = 0.1
;команда "чтение результата"
40 spi = 48h
;сам результат
50 XH = spi : XL = spi
; отобразим в HEX
60 print " HEX -> ", : phb XH, XL, : print "h  :",
; приведем к вольтам для ADC4-1.1
70 U1 = (XH * 256 + XL) / 65535 * 2.5 - 1.25
; приведем к вольтам для ADC4-1.2
75 U2 = (XH * 256 + XL) / 65535 * 20 - 10
; отобразим в вольтах
80 print "For ADC4-1.1 ->", U, "V"
85 print "For ADC4-1.2 ->", U, "V"
;зациклимся
90 goto 10
```

Операторы для работы с приборами MicroLAN

LAN

Оператор **LAN <s_1>,<s_2>,...,<s_n>** выполняет последовательность команд <s_i> обмена с объектами сети MicroLAN. Команды могут быть записаны в любом порядке и выполняются последовательно, слева-направо.
В операторе могут присутствовать следующие команды:

Z(<Line >,<iexpr>) - инициализирует сеть MicroLAN на линии соответствующей **<Line >**, выдает в сеть импульс сброса и проверяет импульс присутствия; если на линии нет приборов MicroLAN, следующие далее команды не выполняются, а управление передается на строку с номером <iexpr>**.**
Значения <Line> - номер порта-линии микроконтроллера. Параметр имеет очевидный формат – например линия порта A №4 может быть адресована как 0A4h , D №15 -> 0DFh.

Z - выдает в сеть импульс сброса, проверка импульса присутствия не производится;

T##<iexpr> – передает в сеть значение <iexpr> в виде двух байтов, причем младший байт передается первым;

T#<iexpr> – передает в сеть значение <iexpr> в виде одного байта;

T(<addr>,<len>) - передает в сеть <len> байтов из RAM начиная с адреса <addr>.

R(<addr>,<len>) - принимает из сети <len> байт и располагает их в RAM начиная с адреса <addr>; при приеме информации из сети вычисляется CRC8, вычисленный код остаётся в системной переменной CRC8.

C#<iexpr> - установка значения системной переменной CRC8.

C - передает в сеть один байт - содержимое системной переменной CRC8.

C(<iexpr>) - читает из сети один байт, а затем сравнивает его с содержимым системной переменной CRC8; в случае несовпадения происходит переход на строку с номером <iexpr>, при этом оставшиеся справа команды не выполняются;

D#<iexpr> – задержка на <iexpr> мс; здесь <iexpr> может принимать значения от 1 до 65535.

S(<addr>,<ln>) задает начальные условия и производит один цикл поиска устройства в сети. Номер найденного устройства будет размещен в 8 байтах RAM начиная с адреса (<addr>+8). Значение <ln> указывает номер строки для перехода при окончании поиска.

S - производит один цикл поиска устройства в сети. Номер следующего найденного устройства размещается в 8 байтах, следующих за ранее определенным номером.

Примечания.

1.Все команды, за исключением Z(<bit>,<ln>) могут быть применены только после инициализации сети.

2.При приеме информации из сети и передаче информации в сеть вычисляется CRC8.

3.Образующий полином для CRC8 имеет вид: $X^8+X^5+X^4+1$.

Пример простого поиска для DS18B20 (команда 0F0h) в предположении, что в сети не более 3-х устройств:

```
lan Z(0C6h,50),T#0F0h,S(1000h,60),Z,T#0F0h,S, Z,T#0F0h,S
```

```
В результате произведенных операций можем получить:
переход на строку 50, если в сети устройств нет;
переход на строку 60, если в сети не более 3-х устройств, причем их номера
будут зафиксированы по адресам 1008h, 1010h, 1018h соответственно;
выполнение следующей строки, если в сети более 3-х устройств (номера первых
3-х будут определены).
```

LANI

Оператор **LANI** является полным аналогом оператора LAN. Разница в том, что этот вариант оператора работает с Lan-устройствами не непосредственно через линии микроконтроллера, а через специальный чип DS2482 , подключаемый по каналу I2C. Этот чип разработан и изготавливается «авторами» MicroLan технологии – фирмой DALLAS(MAXIM). Он наилучшим образом адаптирован к работе с линиями MicroLan, обеспечивая формирование нужных диаграмм в «тонкостях». В перечне выпускаемых нами модулей есть модуль IO4-1.1 с одним из таких чипов DS2482-800. Он обеспечивает взаимодействие с 8-ю лучами MicroLan, дополнительно обеспечивая гальваническую развязку линий от системной I2C шины. Интерпретатор выполняя оператор самостоятельно ведет обмен с DS8482, скрывая всю рутину множества необходимых действий.
Синтаксис оператора полностью повторяет синтаксис LAN. Разница при задании параметра **<Line >.**
В нем указывается I2C адрес модуля и номер выбранной линии . **<Line >** может принимать значения от 0 до 255(0FFh). В старшей тетраде указывается младшая тетрада slave-адреса I2C выбранного модуля(в DS2482 старшая тетрада I2C slave-адреса всегда равна 3), а в младшей тетраде указывается номер линии 0…7.

DS18B20

Функция **DS18B20 (<Line>)** предназначена для работы с термодатчиками DS18B20.
Обязательный параметр <Line> - номер порта-линии микроконтроллера (абсолютная адресация) или номер линии на разъемах конкретного модуля(короткая - косвенная адресация). Более подробно см. PIN .
При первом обращении к конкретной линии, **DS18B20** настраивает указанную линию для работы с цифровым термодатчиком DS1820 и запускает процесс периодического обмена с ним. Первое обращение это инициализация и оно возвращает неправильный результат измерения(измерение еще не проведено, процесс только запущен).
При каждом обмене производится полная диагностика - КЗ, нет ответа(обрыв), не совпадение CRC8.
При возникновении одной из этих ситуаций, диагностика возвращается в самом результате:

-при несовпадении CRC8 возвращается значение "-100.0";
-при отсутствии ответа(обрыве) "-200.0";
-при КЗ на линии "-300.0".

При нормальном ответе термодатчика - возвращается температура в градусах Цельсия с максимальной точностью, которую дает датчик -> 0.0625 градуса. Автоматически, примерно раз в 750мс, производится чтение результата и запуск следующего преобразования. Запрашиваемый результат возвращается немедленно, без ожидания - берется последнее измерение из буфера.
Интерпретатор позволяет параллельно работать с датчиками на 31(48) линии, при этом это никак не отражается на быстродействии бейсик программы, обработка всех разрешенных датчиков идет фоном.
При работе с датчиками поддерживается безадресный режим с / без паразитным питанием.
Необходимым условием правильной работы является наличие подтягивающего резистора 1...3 кОм выбранной линии к +5В или +3.3В. На некоторых линиях он может быть уже установлен - см. описание модуля. Допускается запараллеливание подтяжек, но результирующее сопротивление не должно быть меньше 1 кОм.
Для всех сигнальных входов модулей MCX53-32 подтяжки не требуются, они включаются автоматически, при этом допустима только короткая - косвенная адресация этих линий.
Для случая паразитного питания надо не забывать объединять на датчике ножки "питание" и "общий".
При необходимости, можно работать и с другими типами датчиков: DS1820 и DS18S20. В этом случае потребуется результат делить на 8(у них разрешающая способность меньше в 8 раз), все остальное так же.

Примеры:

```
x = DS18B20(24) ; присваивание x результата измерения DS18B20 на линии 24
print DS18B20(0xA3) ; печать температуры датчика подключенного к линии PA3
```

DMX512

Функция **DMX512 (<adr>)** позволяет работать модулю в качестве slave - устройства на шине DMX512.
Обязательный параметр <adr> - адрес канала DMX512. Возвращаемое значение канала находится в диапазоне 0 … 255.
Часть кадра мастера размером 136 байт сохраняется в буфере. Это позволяет обрабатывать сразу множество каналов параллельно для создания нестандартных конечных устройств или можно получить несколько «стандартных» конечных устройств в одном приборе. После первого вызова функции **DMX512(<adr>)**, канал RS485 переходит в режим работы с протоколом DMX512 и массив значений всех каналов обнуляется.
Буфер приема 136 байт и поэтому в принятом кадре максимальный адрес сохраненных данных не может быть больше минимального адреса больше чем на размер буфера. Например, максимальный адрес с которым мы работаем 500, это значит, что для принятого кадра хранятся в буфере данные для адресов с 365 по 500. И при запросе данных из этого диапазона, данные будут предоставлены сразу из буфера. При запросе адреса выходящего за текущий диапазон, первый раз будет возвращены нулевые данные, а после прихода нового кадра - реальные данные для этого адреса. При этом произойдет автоматическая корректировка диапазона с учетом нового минимального/максимального адреса.
При запросе адреса вызывающего корректировку границ принимаемых адресов, стирается весь буфер и новые валидные данные для всего нового диапазона появятся только после прихода нового кадра.
После сброса контроллера приемный буфер расположен с начала кадра, и после прихода кадра данных будет сразу доступен диапазон 0 -136.
Если данные по DMX512 не обновляются более 1с, то в соответствии с протоколом, это расценивается как потеря связи и все данные обнуляются. В служебный байт с адресом 0 заносится значение 0xFF. При нормальном обмене в соответствии с протоколом – мастер записывает туда 0. Это может быть использовано для диагностики работы линии.
Обращение к служебному байту по адресу 0, не приводит к корректировке адресного диапазона в меньшую сторону.

Пример:

```
10 x = DMX512(435) ;инициализация DMX512 с максимальным сохраняемым адресом 435
20 pwm(8) = 10000 ; инициализация TIM8 в ШИМ с периодом 10000 мкс = 100 Гц
100 if DMX512(0) <> 0 then goto 150; проверка валидности данных
110 a = DMX512(300): b = DMX512(320): c = DMX512(435) ; получение значений каналов
120 pwm(81) = a / 256 ; выдача ШИМ сигнала на канал 1 TIM8
130 pwm(82) = b / 256 ; выдача ШИМ сигнала на канал 2 TIM8
140 pwm(83) = (a + b + c) / (3 * 256) ; выдача ШИМ сигнала на канал 3 TIM8
150 delay = 0.1 ; задержка на 0.1с
160 goto 100
```

Операторы для работы с шиной CAN

Эта группа функций/операторов доступна на модулях где установлен чип драйвера CAN.
Для модулей на базе STM32F103 использование этих операторов блокирует работу с USB. Это связано с архитектурой этих кристаллов, исключающей одновременную работу USB и CAN. Неудобства, создаваемые этой особенностью, частично компенсируются тем, что USB канал в наших модулях используется в основном для загрузки программы. И поэтому в тех случаях когда необходимо использование канала CAN , узел CAN инициализируется не при старте, а при первом вызове операторов/функций CAN$ и ONCAN$. Процесс загрузки программы через USB и последующая работа с каналом не пересекаются. Остается, конечно, неудобство что нельзя при работе с CAN использовать USB для вывода терминальных сообщений, но как правило это можно либо обойти, либо воспользоваться другими каналами вывода(RS485, I2C, SPI, вывод на ЖКИ).

CAN$

Функция / оператор **CAN$ <addr>[,<n>]** позволяет послать пакет данных(Data Frame) или пакет удаленного запроса(Remote Frame) со стандартным(St_ID) или расширенным(Ex_ID) идентификатором.
До вызова функции, в памяти должен быть размещен передаваемый кадр - последовательно 4 байта ID и требуемое количество данных : 0...8 байт.
ID размещается с выравниванием к младшему разряду. Для случая с Ex_ID, оставшиеся старшие 3 бита игнорируются, для St_ID игнорируется 21 старший бит.
В качестве параметров оператора/функции указываются адрес расположения в памяти передаваемого кадра, а так же число отправляемых байт данных n содержащихся в кадре. Если n = 10...19, то размещенный в памяти ID будет трактоваться как Ex_ID 29 бит, если n = 0...9 или отсутствует, то как St_ID 11 бит.
Если параметр n равен 9, 19 или отсутствует, то будет послан кадр "Remote Frame".
Функция возвращает номер mailbox (1...3) в который попал этот пакет или число 4, в случае если все mailbox заняты и пакет не размещен.

ONCAN$

Оператор **ONCAN$ < line >, <addr>[,<n>]** позволяет настроить узел CAN на прием пакетов. До первого вызова оператора должен быть подготовлен буфер размером не менее 12 байт, где заранее размещены фильтр ID 4 байта, его маска 4 байта и будут размещаться приходящие кадры, включающие в себя 4 байта ID и данные до 8 байт.
ID и маска выравниваются к младшему биту. Установка бита(ов) маски в 1 делает соответствующий(е) бит ID существенным при фильтрации приходящих пакетов.
Параметр оператора n устанавливает режим фильтра:
Если n < 10 то будут приниматься только кадры с St_ID, если 10 <= n < 20 то только кадры с Ex_ID.
Если младший десятичный знак n = 9 то будут приниматься только кадры "Remote Frame", если n = 0 то только кадры "Data Frame".
Если параметр n отсутствует, то будут приниматься кадры "Remote Frame" и "Data Frame" с любым типом ID.
Для этого случая в памяти должны быть размещены ID и маска в формате Ex_ID 29 бит.
После прихода кадра удовлетворяющего условиям фильтра и маски, с 0-го по 3-й байт будет размещен пришедший ID, а начиная с 4-го байта - данные. При этом будет передано управление подпрограмме с номером строки указанном в первом параметре оператора: `line`. Второй параметр оператора : `adr` - это адрес ячейки начала буфера приема.
Можно объявить до 7 разных подпрограмм с индивидуальными комплектами параметров. Комплекты параметров привязываются к номеру строки обработчика. Параметры уже объявленной подпрограммы можно переобъявить вызвав этот оператор с номером строки этой подпрограммы и адресом нового буфера. В случае переобъявления параметров обязательно надо выбрать другой заранее подготовленный буфер с комплектом фильтра и маски, т.к. текущий буфер занят приемом до момента переинициализации.
Возврат из подпрограммы обработки этого прерывания должен осуществляться оператором RETI.

CAN

Функция **CAN** возвращает признак стандартного или расширенного ID и число байт данных в пришедшем кадре. При значениях 0...9 это St_ID, при 10...19 - Ex_ID. Последняя десятичная цифра 0...8 это число пришедших байт данных. Если последняя десятичная цифра равна 9, значит пришел кадр удалённого запроса данных.

Оператор **CAN** позволяет установить скорость работы шины.
Доступны величины 1000, 500, 250, 125, 100, 50, 20, 10 кБит/с.
По умолчанию скорость предустановленна на 1000 кБит/с. Если для работы требуется другая скорость, то до первого использования CAN$ или ONCAN$ нужно указать одну из перечисленных скоростей.

```
Пример №1 передача:
5 delay=5 ;пауза при старте, для возможности останова программы до инициализации CAN
10 dim $(2),12 ;буфера для исходящих телеграмм
20 adr_1=loc($(0)) ;получим адрес буфера передачи 1
```

```
30 adr_2=loc($(1)) ;получим адрес буфера передачи 2
40 mem(adr_1)=0x78,0x56,0x34,0x12 ;занесем ID 29 бит 0x12345678
50 mem=0x11,0x22,0x33,0x44,0x55 ;занесем данные 5 байт
60 mem(adr_2)=0x65,7,0,0 ;занесем ID 11 бит 0x765
70 mem=0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17 ;занесем данные 8 байт;
80 can=20 ; выбираем скорость 20 кБит/с
;основной цикл
100 if pin(23)=1 then goto 200 ;определим состояние линии 23 и выберем телеграмму
110 x1=can$ adr_1,15 ;пошлем буфер №1 5 байт с расширенным ID, x1->подтверждение
120 goto 300
200 x2=can$ adr_2,8 ;пошлем буфер №2 8 байт со стандартным ID, x2->подтверждение
300 delay=0.1 ;пауза
310 goto 100 ;циклимся
```

Пример №2 прием:

```
5 delay=5 ;пауза при старте, для возможности останова программы до инициализации CAN
10 dim $(2),12 ;приемные буфера
20 adr_1=loc($(0)) ;получим адрес буфера 1
30 adr_2=loc($(1)) ;получим адрес буфера 2
40 can=20 ; выбираем скорость 20 кБит/с
50 mem(adr_1)=0x78,0x56,0x34,0x12 ;занесем фильтр 29 бит 0x12345678
60 mem=0xFF,0xFF,0xFF,0x1F ;занесем маску - значимы все 29 бит
70 mem(adr_2)= 0x65,7 ;занесем фильтр 11 бит 0x765
80 mem(adr_2 + 4)=0xFF,7 ;занесем маску - значимы все 11 бит
90 ONCAN$ 1000,adr_1,10 ;инициализация приема в буфер adr_1 ,обработчик 1000, ExID
95 ONCAN$ 2000,adr_2,0 ;инициализация приема в буфер adr_2 ,обработчик 2000, StID
;
100 delay=0.01
105 pin(24)=0: pin(25)=0 ;гасим светодиоды
110 goto 100
;
;попадаем сюда при приходе пакета от CAN с ID=0x12345678
1000 pin(24)=1 ;вкл светодиод
;в буфере adr_1 находится пришедший кадр
1010 n_1=can ;число пришедших байт и тип ID(при исп.примера №1 будет равен 15)
1020 reti ;возврат из обработчика
;
;попадаем сюда при приходе пакета от CAN с ID=0x765
2000 pin(25)=1 ;вкл светодиод
;в буфере adr_2 находится пришедший кадр
2010 n_2=can ;число пришедших байт и тип ID(при исп.примера №1 будет равен 8)
2020 reti ;возврат из обработчика
```

Операторы для работы с аппаратными ресурсами

PIN

Оператор **PIN (<Line>)** предназначен для задания последующего входного режима работы линии, либо задает выходное состояние линии. Обязательный параметр <Line> - номер порта-линии микроконтроллера (абсолютная адресация) или номер линии на разъемах конкретного модуля(короткая - косвенная адресация).

Параметр <Line> имеет очевидный формат:

- если он в шестнадцатеричном выражении начинается с буквы соответствующей выбранному порту микроконтроллера, адресуется именно выбранная линия соответствующего порта микроконтроллера;
- если он находится в пределах 0…1Fh (0…31), то адресуется линия модуля с учетом разводки на разъемы. Так, на модуле MCU32-1.x пользователю доступны 32 линии ввода вывода, попадающие на разъемы модуля группами по 8. Соответственно при такой адресации не нужно заботиться о том какие именно линии каких портов попадают на конкретный разъем, а просто адресовать их подряд.

Например, линия порта A №4 может быть адресована как 0A4h, D №15 -> 0DFh , а линия модуля MCU32-1.x IO3 (контакт №3 X2) как 3, линия KR19(контакт №19 X1) как 1Fh.

Такой двойной подход позволяет максимально упростить пользователю работу с линиями, не теряя при этом гибкость для более тонкого использования всех возможностей конкретного модуля, конкретного микроконтроллера.

Варианты значений:

Номер режима	Описание	Результат
0	**Дискретный выход 0В без ограничительного резистора**	**Выход = 0**
1	**Дискретный выход +3.3В без ограничительного резистора**	**Выход = 1**
0x10	Дискретный выход 0В с ограничительным резистором 1К	Выход = 0 через 1К
0x11	Дискретный выход +3.3В с ограничительным резистором 1К	Выход = 1 через 1К
0x40	**Дискретный выход альтернативной функции push-pull**	**Выход аппаратного ресурса PP**
0x41	**Дискретный выход альтернативной функции open-drain**	**Выход аппаратного ресурса**
0x80	**Режим дискретного входа с подтяжкой 0В / ~ 40 К**	**X = PIN(N) ; X = 0/1**
0x81	**Режим дискретного входа с подтяжкой +3.3В / ~ 40 К**	**X = PIN(N) ; X = 0/1**
0x82	**Режим плавающего дискретного входа 0 / +3.3В**	**X = PIN(N) ; X = 0/1**
0x90	Режим дискретного входа с подтяжкой 0В / 1 К (делитель 0 / +24В)	X = PIN(N) ; X = 0/1
0x91	Режим дискретного входа с подтяжкой +3.3В / 1 К («сухой контакт»)	X = PIN(N) ; X = 0/1
0xC0	**Аналоговый режим потенциального входа 0…+3.3В**	**X = ADC(N) ; X = 0 … 3.3**
0xD0	Аналоговый режим потенциального входа 0…+10В(+24В)	X = ADC(N) ; X = 0 … 24
0xD1	Режим работы с термодатчиками PT1000	X = ADC(N) ; X = 0 … 3.3
0xE0	Аналоговый режим токового входа 0…20мА	X = ADC(N) ; X = 0 … 20

Доступность того или иного режима см. в описании на соответствующий модуль. В некоторых случаях для реализации конкретного режима требуется установка соответствующих джамперов. Жирным шрифтом выделены режимы доступные во всех модификациях.

Примеры:

```
pin(31) = 1       ;выдача 1 на линию 31(для модуля MCU32-1.x это KR19)
pin(0xA7) = 0     ;выдача 0 на линию 7 порта A
pin(3) = 0xD1     ;настройка линии 3 для последующей работы в режиме PT1000
```

Функция **PIN(<Line>)** возвращает состояние указанной линии 0 или 1.

Режим линии остается таким, как задан ранее в операторе PIN.

Если ранее инициализация не проводилась, то линия будет работать в режиме плавающего дискретного входа 0x82.

Если линия до этого была настроена как выход, то произойдет автоматическая настройка в режим 0x81-дискретный вход с подтяжкой+3.3В / ~40К.

Примеры:

```
A = pin(0B5h)     ;опрос состояния линии 5 порта B
B = pin(3)        ;опрос состояния линии 3
```

ADC

Функция **ADC(<Line>)** предназначена для получения измерения выбранного канала АЦП.

Параметром функции является номер аналогового канала (0…15).

Линии имеющие функции аналогового входа перечислены в описании соответствующего модуля.

При вызове функции происходит автоматическая инициализация соответствующей линии в режим аналогового входа.

Для модулей с расширенными возможностями конфигурации входов(MCX53-32.x), возможна предварительная конфигурация входных цепей, которая осуществляется оператором PIN. При этом формат результата будет соответствовать выбранному режиму.

При работе в обычном режиме функция возвращает измерение в одном из трех форматов заданных оператором ADC. По умолчанию в вольтах.

При первом обращении к каналу, который до этого момента не был аналоговым, возможен возврат неверного значения.
```
Пример измерения канала ADC4 :      X = adc(4)
```

В функции присутствует возможность выбора формата данных. Формат один для всех каналов АЦП.
Возможны 3 варианта:

- формат 0 (по умолчанию) -> величина в вольтах 0…3,3 В;
- формат 1 -> величина соответствующая коду АЦП 0…4095;
- формат 2 -> величина в долях от целого 0…1.

Для выбора формата необходимо обратиться к несуществующим каналам 100, 101, 102 соответственно.
```
Пример установки формата «код АЦП»  :     X = adc(101)
```
Также функция позволяет получить сведения о температуре кристалла и напряжении REF.

Функция ADC(200) возвращает значение в вольтах, измеренное на этом датчике.
Температуру в Цельсиях можно рассчитать по формуле T (in °C) = (1.43 - ADC (200)) / 0.0043 + 25.

Функция ADC(201) возвращает значение REF в вольтах. В качестве опорного напряжения для встроенного АЦП используется основное напряжение питания микроконтроллера 3.3В, оно задается отдельным внешним для микроконтроллера стабилизатором. Встроенный REF имеет типичное значение = 1.2В. При необходимости, пользователь может использовать его измерение для дополнительной калибровки точности своих измерений.

DAC

Оператор предназначен для выдачи аналогового сигнала на выбранный канал ЦАП.
Параметром является номер аналогового канала (1…2).
Линии имеющие функции аналогового выхода перечислены в описании соответствующего модуля.
При вызове оператора происходит автоматическая инициализация соответствующей линии в режим аналогового выхода. Оператор устанавливает значение ЦАП (0…3,3В) в соответствии с выбранным форматом данных.
По умолчанию выбран формат «в вольтах».
```
Пример выдачи на DAC2 1.45В:       dac(2)=1.45
```
Для выбора формата данных необходимо обратиться к несуществующим каналам 100, 101, 102 соответственно.
Формат один для всех каналов ЦАП.
Возможны 3 варианта:

- формат 0 (по умолчанию) -> величина в вольтах 0…3,3 В;
- формат 1 -> величина соответствующая коду записываемому в ЦАП 0…4095;
- формат 2 -> величина в долях от целого 0…1.

```
Пример установки формата «доля от целого»  :   dac(102) = 0
```

LCD

Оператор **LCD(<Address>) = <iexpr$_1$>,<iexpr$_2$>,... <iexpr$_n$>**
обеспечивает поддержку работы модулей с Fractal-BASIC-Cortex с ЖКИ и OLED индикаторами совместимыми с символьными с системой команд HD44780 или с графическими индикаторами с шиной I2C и контроллером PCF8531.
Для работы с первыми оператор используется для назначения линий подключения и обязательной инициализации режимов, для работы со вторыми инициализация не требуется, а оператор применяется для установки яркости, контрастности и цвета подсветки.
Вывод сообщений на ЖКИ осуществляется через оператор PRINT#.
В адресе PRINT# для вывода на LCD надо указывать 16.

Для случая внешних символьных ЖКИ, вызов оператора инициализирует работу с ЖКИ по конкретным линям. Для подключения используется тетрадный режим обмена, что позволяет подключать индикаторы всего по 6 линиям : 4 данные и 2 управляющие. Доступны все линии присутствующие на разъемах модуля. При этом не имеет никакого значения последовательность или очередность линии, любая из 6 линий может быть назначена на любой вывод разъема. Для инициализации параметр **<Address>**оператора должен указываться 100 и после обязательно перечисляются все 6 линий.
Выбранные пользователем линии просто перечисляются в последовательности: RS, E, D4, D5, D6, D7.
Линия ЖКИ “R/W” не используется, т.к. в индикатор производится только запись и на этом можно сэкономить. Вывод “R/W” на индикаторе надо замкнуть на землю. Адреса линий задаются как в операторе PIN. Таблицу соответствия см. описание соответствующего модуля.
```
Пример инициализации:        LCD(100) = 0A3h,0B7h,0A1h,0A5h,0C6,0A2h
еще вариант       LCD(100) = 1,0,6,8,2,15
```
Для управления позицией вывода – курсором, нужно использовать оператор CSR.
Для двух строчных ЖКИ как правило вторая строка начинается с 40 –го знакоместа.
Оператор LCD(200) может задавать вид курсора:
```
LCD(16) = 12      ;курсор выключить
LCD(16) = 14      ;курсор включить
LCD(16) = 15      ;курсор мигающий
```

Этот же оператор можно использовать для записи команд непосредственно в контроллер ЖКИ (HD44780).
Оператор **LCD(201)** позволяет записывать байт данных непосредственно в алфавитно-цифровой ЖКИ (совместимый с HD44780).

!!!Рекомендации по подключению сивольных ЖКИ:
Программно, как уже упоминалось, можно использовать все доступные пользователю линии.
НО, поскольку микроконтроллер использует для своего питания напряжение 3.3В, его *ВСЕ* выходные каскады могут либо выдавать уровни соответствующие 0 и 1(0 и 3.3В), либо *ЧАСТЬ* линий толерантных к +5В могут работать по схеме с открытым стоком, что в паре с внешним подтягивающим резистором дает размах +5В для единичного состояния. Следовательно, выбор конкретного типа ЖКИ может накладывать ограничения на часть линий модуля.
Самый простой вариант(но не самый распространенный) – ЖКИ для которых потенциал 3.3В уже гарантировано является логической единицей(см.спецификации ЖКИ). Как правило, такие ЖКИ сами могут питаться от 3.3В. Такие индикаторы можно подключать к любым доступным линиям микроконтроллера. При этом варианте нужно указать интерпретатору, что состояние логической единицы будет кодироваться активным состоянием выхода – логической единицей(3.3В). Это делается вызовом LED(106) = 1. Для этого случая внешние резисторы не нужны.
Второй вариант это ЖКИ у которых входной уровень логической единицы больше 3.3В. Для подключения таких индикаторов можно использовать только линии толерантные к +5В. Для MCU32-1.x это все 16 линий попадающие на кроссовый разъем. Эти линии могут работать по схеме с «открытым стоком». Для получения на них размаха логического сигнала близкого к +5В нужно выбранные линии «подтянуть» резисторами 1.5К…10К к +5В.
При использовании MCU32-1.x совместно с некоторыми модификациями кросс - модулей CRS10-1.x, линии подаваемые на ЖКИ можно не подтягивать, т.к. подтяжки уже стоят на модулях CRS10-1.x(см. соотв. схемы).
По умолчанию, после сброса для оператора LCD выбран режим LED(106) = 3, при котором состояние логической единицы кодируется состоянием выхода «вход», что эквивалентно обрыву и следовательно при наличии подтягивающего резистора там будет +5В.

Для графического встроенного ЖКИ с контроллером PCF8531 подключенного по I2C
Установка цветов `LCD(140) = 0…14`,
Для раздельного задания R, G, B соответственно `LCD(141…143) = 0…64`
Для управления контрастностью применяется `LCD(110) = 0x80…0xFF`
Для управления режимом вывода применяется оператор `LCD(111) = …`
Для очистки/заполнения/инверсии применяется оператор `LCD(130) = …`
Для управления курсором применяется оператор `LCD(120) = x , y, <rej>`
Более подробная информация в описании соответствующего модуля.
Для модуля MCX53-20.x доступно управление звуком: `LCD(150) = тон`,
`LCD(151) = длительность`. Тон задается из расчета длительности периода сигнала в микросекундах, длительность в миллисекундах.

COUNT

Оператор / функция **COUNT** предназначены для счета импульсов от внешних источников и работы с энкодерами. Подсчет происходит аппаратно, независимо от работы BASIC- программы.
Доступно до трех(в зависимости от типа модуля) независимых 16 битных счетчиков. Для запуска счета необходимо вызвать оператор COUNT с параметрами инициализации. В параметрах указывается:
- номер аппаратного таймера (2, 3, 8) -> третья тетрада считая от младшей,
- режим счета(счет с/без обнулением при прочтении, энкодер, энкодер с обнулением) -> вторая тетрада от младшей,
- комбинация входа(ов)(для счетчика до 4 линий, для энкодера до 2 пар линий) -> младшая тетрада,
- начальное состояние счетчика (0…0xFFFF ⇔ 0…65535).

При инициализации выбранная(ые) линии автоматически инициализируются как входы с подтяжкой.
В режиме энкодера подсчет ведется при каждом изменении любого из сигналов, т.е. на период получается 4 .
Т.к. используются аппаратные таймеры, то для подачи входного сигнала доступны не все линии, а только конкретные комбинации для конкретного типа модуля. Доступные комбинации приведены в описаниях на модули.
Функция COUNT возвращает число накопленных импульсов(шагов для энкодера). Есть 2 группы режимов с обнулением и без обнуления при прочтении.
Оператор COUNT заносит в счетчик требуемую величину.

Примеры:

```
count(213h)=7 ;инициализация TIM 2 в счетчик без обн. на 3 вход с предустановкой 7
count(821h)=5 ;инициализация TIM 8 в счетчик с обн. на 1 вход с предустановкой 5
count(831h)=4 ;инициализация TIM 8 в энкодер без обн. на 1 комб. с предустановкой 4
count(342h)=0 ;инициализация TIM 3 в энкодер с обн. на 2 комб. с предустановкой 0
count(3)=5678 ;занесение в TIM 3 значения 5678
x = count(8)  ;чтение TIM 8
```

PWM

Оператор **PWM** (<iexpr>)=<expr> позволяет использовать аппаратные ШИМы микроконтроллера. Одновременно доступно до 12 линий ШИМ и до 9 линий с фазовым регулированием. Точное число доступных линий определяется конкретным типом модуля(см. описания модулей).
Для работы в режиме ШИМ в интерпретаторе доступны одновременно до трех таймеров(2, 3, 5, 8) по четыре канала(1-4) в каждом. Каналы 2 и 5 используют одни и те же линии. Возможность выбора между каналами 2 и 5 может пригодиться в модулях, где один из них уже задействован на внутренние ресурсы модуля. Так, например в модуле MCX53-21.2x, TIM2 задействован на выдачу звука на встроенный излучатель, вместо него без потери функциональности можно пользоваться TIM5.
Поддержаны 2 режима работы оператора: обычный и с фазовым регулированием. Для обычного режима доступны 4 выхода, для фазового регулирования 3 выхода и одна линия является входом фазового детектора. Входом фазового детектора всегда является первый канал таймера.

Время периода(для фазового режима это глубина регулирования) задается в микросекундах индивидуально для каждого из таймеров. Для выбора режима фазового регулирования эта величина задается со знаком "-".
Диапазон задания периода/глубины регулирования от 1 до 65535мкс.
Величина ШИМ задается индивидуально для каждого выбранного канала в долях от целого: 0 ... 1.0. Минимальный квант выбран равным 1 мкс. Соответственно чем меньше период, тем разрешающая способность ШИМ меньше. При максимальном периоде она равна 1/65536 (16 бит).
Для инициализации необходимо сначала настроить режим таймера выбранного канала. Для этого в параметре указывается номер таймера, после равенства задается период ШИМ в микросекундах.

Примеры:

```
pwm(2)=10000 ;инициализация TIM 2 в ШИМ с периодом 10000 мкс = 100 Гц
pwm(8)=500 ;инициализация TIM 8 в ШИМ с периодом 500 мкс = 2000 Гц
pwm(3)=-7000 ;инициализация таймера 3 с глубиной регулирования 7мс
```

При необходимости можно многократно изменять период таймера.
Инициализация не переводит линии микроконтроллера в режим ШИМ. Для начала работы выбранной линии необходимо обратиться к ней, указав значение ШИМ, при этом линия автоматически настраивается на выход (режим push-pull) и начинает выдавать ШИМ сигнал с заданными параметрами. Линия указывается в параметре вместе с номером таймера.
При этом десятки это номер таймера, единицы это номер выбранного канала этого таймера.
Если вместо номеров 1...4 указать соответственно 5...8, то указанная линия(больше номера на 4) то режим выхода этой линии будет open-drain(открытый сток).
Соответствия ножкам разъемов и количество доступных линий см. в описании на конкретный модуль.

Примеры:

```
pwm(21)=0.5 ;перевод соотв.линии канала 1 TIM 2 на выдачу ШИМ скважностью 50%
pwm(34)=0.123 ;перевод соотв.линии канала 4 TIM 3 на выдачу ШИМ скважностью 12.3%
pwm(82)=0.0001 ;перевод соотв.линии канала 2 TIM 8 на выдачу ШИМ скважностью 0.01%
```

Замечания по режиму фазового регулирования:
На линию приходящую на первый канал каждого из таймеров подается сигнал с фазового детектора. В простейшем случае вполне достаточно всего 2 компонентов:

- резистора сопротивлением порядка 220 кОм / не менее 0.5 Вт и допускающим падение на нем не менее 400В;
- транзисторной оптопары с биполярным входом, например PC814.

На вход оптопары через резистор подается сетевое переменное напряжение, а выходной транзистор подключается непосредственно между землей модуля и входом фазового детектора без дополнительных компонентов. При этом не должно быть никаких соединений между высоковольтной частью и модулями, сигнал передается от сети только через оптопару. Подтягивающие резисторы к выходу оптопары не нужны.
Второй, третий и четвертый каналы таймеров могут быть проинициализированы как выходы на фазовое управление. К ним через ограничивающий резистор можно подключать оптотриаки *без "детектора нуля"*. Для оптотриаков с входным управляющим током 10 мА это резистор порядка 200 Ом.
Данные рекомендации даны из расчета, что пользователь имеет необходимые знания нужные для работы с высоким напряжением и мерах безопасности при этом.
!Если у Вас нет нужной квалификации для работы с сетевым напряжением - не используйте наши модули для этого!
Управление нагрузками осуществляется занесением необходимых значений в ШИМ нужного канала.
Пользователь имеет возможность самостоятельно задавать глубину регулирования в зависимости от конкретной реализации своей схемы и типа нагрузки.
Поскольку основой для реализации этой функции служат аппаратные таймеры, микроконтроллер не тратит программного времени на фазовое регулирование, оно работает полностью прозрачно и независимо от программы.

Операторы для работы со встроенным ЖКИ и звуком (только для модулей МСХ53-20.х)

POINT

Оператор **POINT** выводит точку с координатами X и Y.

```
Пример:     POINT 107, 23     ; вывод точки с координатами x=107, y=23
```

LINE

Оператор **LINE** выводит линию с координатами от прошлого значения POINT или LINE до X и Y.

```
Пример:     LINE 67, 14 ; вывод линии в точку x=67, y=14
```

RECT

Оператор **RECT** выводит прямоугольник с координатами от прошлого значения POINT или LINE до X и Y.

```
Пример:     RECT 49, 30 ; вывод прямоугольника в точку x=49, y=30
```

COLOR

Оператор **COLOR** управляет яркостью и цветом подсветки ЖКИ. Если указан один параметр, то он расценивается как цвет из палитры 14 основных цветов. Если заданы три параметра, то они задают яркость красного, зеленого и синего соответственно. Диапазон 0…100.

```
Пример:     COLOR 8     ; включить ярко красную подсветку
            COLOR 50,50,50    ; включить подсветку белого на половинную яркость
```

BEEP

Оператор **BEEP** <expr>,<expr> обеспечивает подачу звукового сигнала на динамик, в модулях где он предусмотрен, или на выбранную линию. Первый параметр – частота в герцах(100...20000), второй – длительность в секундах 0…9.999с. После обработки этого оператора интерпретатор сразу переходит к обработке следующего, т.е. воспроизведение звука не приостанавливает работу программы. Для случаев, когда требуется ожидание завершения звука, например проигрывание мелодии, длительность необходимо указывать на 10 больше. Если в качестве частоты указан 0, то будет проиграна пауза нужной длительности.

В модулях где не предусмотрена установка динамика можно выдать звуковой сигнал на одну из предусмотренных линий. В этом случае требуется предварительная инициализация оператора. Для выбора доступно до 12 линий(определяется конкретным типом модуля см. описания модулей). Для работы оператор использует один из трех(2, 3, 8) таймеров. Выбранный таймер нельзя будет использовать для других целей, например для работы с ШИМ.

Т.е. если конкретный таймер инициализируется для работы с BEEP, то не получится использовать свободные каналы этого таймера для работы с ШИМ.

При инициализации оператора в параметре необходимо указать номер таймера и канала - таймер в десятках, канал в единицах.

!Обратите внимание при инициализации параметр ОДИН!

```
      Примеры:
beep 21 ;перевод соотв.линии канала 1 TIM 2 на выдачу звукового сигнала
beep 32 ;перевод соотв.линии канала 2 TIM 3 на выдачу звукового сигнала
beep 84 ;перевод соотв.линии канала 4 TIM 8 на выдачу звукового сигнала
```

!!!В модулях в которых предусмотрен динамик, проводить инициализацию канала звука не надо!

```
Пример «Былененький он зел»:
10 beep 1335,10.5
20 beep 1000,10.5
30 beep 1335,10.5
40 beep 1000,10.5
50 beep 1335,10.5
60 beep 1260,10.25
70 beep 0,10.25
80 beep 1260,11
```

Операторы и функции для работы с памятью данных (RAM) и регистрами

PRINT@

Оператор **PRINT@<addr>,<s_1>,<s_2>,...<s_n>** производит "печать" в память. Выводимые коды располагаются начиная с адреса <addr>.

MOVE

Оператор **MOVE <$addr_1$>,<$addr_2$>,<len>** копирует <len> байтов начиная с адреса <$addr_1$> в область памяти с начальным адресом <$addr_2$>. Оператор может быть применен для выделения подстроки из строчной переменной, например:

```
move loc($(0)) +3 ,loc($(1)),8  : mem(loc($(1))+8)=0
```

- в строчную переменную $(1) скопированы 8 символов (начиная с 3-го, если считать от нуля) из строчной переменной $(0). Код “0” - маркер конца строчной переменной.

CMP

Функция **CMP(<$addr_1$>,<$addr_2$>,<len>)** сравнивает байтовые последовательности с начальными адресами <$addr_1$>,<$addr_2$> и длиной <len> байтов в RAM и возвращает нуль, если последовательности идентичны, в противном случае - отличное от нуля число, равное количеству идентичных байтов. Функция может быть применена для сравнения строчных переменных. При **<$addr_1$>** или **<$addr_2$>** отрицательном, происходит адресация Flash (по модулю). Это позволяет сравнивать RAM с Flash и т.д. *Пример:*

```
if cmp(loc($(0)),loc($(1))+5,10) then 110 else 100
```

- если первые 10 символов $(0) и символы с 5 по 15 $(1) идентичны, перейти на строку программы 100, иначе - на строку 110.

MEM

Оператор **MEM[W](<addr>)=<$iexpr_1$>,<$iexpr_2$>,…<$iexpr_N$>** записывает значения байтов или слов в RAM начиная с адреса <addr>. Записываемые значения задаются **<$iexpr_1$>,<$iexpr_2$>,…<$iexpr_N$>**. Значения **<addr>** и **<iexpr>** могут быть в диапазоне 0…65535. **<addr>** не совпадает с абсолютными адресами Cortex-M3. Он покрывает область данных интерпретатора.
Младший по адресу в RAM байт соответствует младшему байту в операторе/функции.

Функция **MEM[W][(<addr>)]** возвращает значения байтов или слов RAM. Адрес следующего байта или слова запоминается в системной переменной. Например оператор **Print mem(100),mem,mem,mem** распечатает содержимое байтов по адресам 100, 101,102,103.

Оператор **MEMB#<n> , (<addr>)= <$iexpr_1$>, <$iexpr_2$>, ...<$iexpr_n$>** производит установку или сброс битов в байте RAM, адресуемом выражением <addr>. Установка или сброс производятся начиная с бита номер **<n>.** Число **<n>** может быть задано константой, переменной или выражением и быть в диапазоне 0…15. Биты нумеруются начиная с младшего. Если **<iexpr>** = 0, то бит сбрасывается, в противном случае – устанавливается. Операция производится по принципу: «чтение-модификация-запись».

Функция **MEMB#<n> , (<addr>)** возвращает значение бита в байте RAM по адресу **<addr>**. Функция возвращает значения “0’ или “1”. *Пример:*

```
10 mem(0)=55h,2
20 phb mem(1),mem(0)
```

CRC16 / CRC8

Функции **CRC16 <addr>,<len>** и **CRC8 <addr>,<len>** вычисляют соответственно CRC16 для MODBUS RTU и CRC8 для MicroLan для области RAM или FLASH начальным адресом <addr> из <len> байт.
Для CRC16 максимальное число ячеек 65535, для CRC8 - 256. Если адрес указан как положительное число, то идет работа с RAM, если отрицательный, то с FLASH.

REG

Оператор **REG[W](<addr>)=<iexpr>** устанавливает значение байта или слова в области регистров по адресу <addr>. Записываемое значение задается <expr>. При записи слова оператор **REGW** размещает младший байт по адресу <addr>, а старший – по адресу <addr+1> .
Оператор / функция REG имеют в качестве параметра адрес в диапазоне 0…0FFFFh.
При этом диапазон 0… 7FFFh соответствует адресам Cortex-M3 0x4000 0000…0x4000 7FFF,
диапазон 8000h…0FFFFh соответствует адресам Cortex-M3 0x4001 0000…0x4001 7FFF.
В эти области попадают основные регистры внутренних узлов.

Функция **REG[W](<addr>)** возвращает значения байта или слова нижней половины I_RAM или регистра специальных функций по адресу <addr>. При чтении слова функция REGW() рассматривает первый считанный байт как младший.

Оператор **REGB#<n> , (<addr>) = <iexpr$_1$>, <iexpr$_2$>, ...<iexpr$_n$>** производит установку или сброс битов в байте, адресуемом выражением <addr>. Возможно обращение только по четным адресам, т.е. регистры рассматривается как 16-ти битные. Установка или сброс производятся начиная с бита номер **<n>.** Число **<n>** может быть задано константой, переменной или выражением и быть в диапазоне 0…15. Биты нумеруются начиная с младшего. Если **<iexpr>** = 0, то бит сбрасывается, в противном случае – устанавливается. Операция производится по принципу: «чтение-модификация-запись».

Функция **REGB#<n> , (<addr>)** возвращает значение бита в байте по адресу **<addr>**. Функция возвращает значения “0’ или “1”. Ограничения те же , что и в операторе.

Операторы работы с FLASH-памятью.

FLASH

Оператор / функция FLASH**[W]** предназначены для работы с Flash –памятью данных пользователя.
В сравнении с оператором/функцией FLS, они значительно упрощают работу с данными сохраняемыми в пользовательской области Flash-памяти данных. Единицей при работе с Flash является переменная(4 байта).
Адресация прямая – к элементу массива, нумерация начинается с 0-го элемента буфера и т.д. до конца буфера.
Максимальное количество переменных которые пользователь может сохранить в буфере выдает директива MTOP, при разделении Flash памяти на части – память текста BASIC-программы и память долговременно сохраняемых пользовательских данных. Новый модуль содержит указатель в значении которое предоставляет весь объем тексту программы и нет буфера данных. Пред первым применением FLASH нужно обязательно воспользоваться директивой MTOP исходя из объема программы и ее статуса(отладка или законченный проект).
Оператор FLASH[(<Address>)]=<expr0>,<expr1>...<exprn> где Address - индекс в массиве сохраняемых переменных (4-хбайтовых float значений), который образуется в пользовательской части FLASH выше области отведенной под BASIC-программу. Одна единица индекса – одна переменная (4 байта Flash). При отсутствии Address в операторе запись производится в ячейку, следующую за последней записанной. Следует помнить, что запись возможна только в предварительно стертую ячейку, иначе будет выдано сообщение об ошибке. Стереть ячейку можно оператором ERASE.
Функция FLASH[(<Address>)] . Возвращает значение сохраненной переменной (4-х байтовое float-значение) из ячейки с индексом Address. Если индекс в функции не указан, то читается ячейка с индексом, следующим за последним прочитанным. То есть указатели чтения и записи независимы.
При использовании модификатора [W] данные <expr0>,<expr1>...<exprn> рассматриваются как бинарные слова 2 байта(16бит). (<Address>) указывается с точностью до байта, т.е. он соотносится с адресацией FLASH без модификатора как 4:1.
Адресация сделана байтовой, чтобы для бинарных данных она совпала с адресацией используемой при доступе через MODBUS(там идет тоже адресация байта). Обращение допускается только по четным адресам, т.к. из за особенностей применяемых чипов , обеспечивается доступ только к 16-битным словам.
FLASH и FLASHW оперируют одним и тем же массивом Flash памяти и начинаются с одной и той же ячейки 0.

ERASE

Оператор ERASE(<Address>) стирает ячейку с индексом Address. Следует учесть, что поскольку блоки FLASH стираются по 2 Кбайта, то будут стерты все ячейки, попавшие в данный блок – всего 512 4-хбайтовых float-ячеек.

EEPROM

Функция/оператор **EEPROM (<Address>)** предназначены для энергонезависимого сохранения ограниченного количества данных пользователя и конфигурации системы. Поскольку в применяемых кристаллах STM32F103, EEPROM отсутствует, его работа сэмулирована на базе основной Flash.
Прямое сохранение пользовательских данных во Flash имеет два недостатка:

- необходимость стирания блока перед записью, если записываемая ячейка была использована до этого(не 0xFF);
- ограниченный ресурс перезаписей Flash (10 000).

Оператор/функция EEPROM, как и внутренний алгоритм сохранения текста BASIC –программы лишены этих недостатков. Хотя, конечно, ресурс все равно не бесконечен. Реально ресурс повышается примерно на порядок при использовании средней загруженности занимаемых объемов. Так, если Вы используете обновление ячейки с одним и тем же адресом N раз, это эквивалентно использованию N ячеек один раз. Т.е. речь идет не о ресурсе каждой ячейки, а общем количестве перезаписей.
Пользователю предоставлено 256 слов по 16 бит, кроме этого есть системные ячейки для хранения параметров системы. Пользовательские адреса 0…255. Кроме пользовательской области , существует область системных установок. В этой области допустимо использовать только приведенные в описании адреса.

```
Пример записи ячейки с адресом 0  :      eeprom(0) = 55AAh
Пример чтения ячейки с адресом 5  :      X = eeprom(5)
```

TSTFL

Функция TSTFL предназначена для быстрой проверки доступности для записи нового значения переменной по конкретному адресу массива Flash данных пользователя. Другими словами она проверяет на «стертость» комплекта из четырех байт по указанному адресу. В случае если ячейки «чистые» возвращается 1, если нет, то 0.
Небольшая сложность в определении этого факта без этой функции состоит в том, что функция FLASH возвращает переменную в формате «плавающая». И значение чистой переменной, когда все 4 байта равны 0xFF, для формата **IEEE754** не соответствует никакому числу.

```
Пример проверки ячейки с адресом 0 и стирания !!!ВСЕГО блока если она не чистая:
if tstfl(0) = 0 then erase(0)
```

Операторы для работы со счетчиком реального времени

TIME

Оператор **TIME=<iexpr>** устанавливает начальное значение счетчика времени.
Значение <iexpr> может быть в диапазоне 0...65535 с.

Функция **TIME** возвращает значение счетчика реального времени. Дискретность счета времени - 0.01 сек. Максимальное значение счетчика - 65535.99 с.

```
Пример задержки на 0.5 с:
10 time = 0: do: while time < 0.5
```

DELAY

Оператор предназначен для простой организации временнЫх задержек в программе.
Ему могут присваиваться значения от 0.01 с до 65.535 с.
Работа оператора DELAY прерывается событиями Ctrl-C, ONINT1, ONTIME.

```
Пример задержки на 3.5 с   : delay = 3.5
```

RLDT

Оператор **RLDT=< day >,<month>,< year >,<hour>,<min>,<sec>** устанавливает часы реального времени.
<year>=<текущий год> - 2000
<month> - номер месяца (1=январь,2=февраль ... 12=декабрь)
<day> - день месяца
<hour> - часы
<min> - минуты
<sec> - секунды
Например, чтобы установить часы реального времени в состояние: 11 часов 51 минута 20 секунд 24 декабря 2010 года надо подать команду: `RLDT = 24, 12, 10, 11, 51, 20`

Функция **RLDT** выдает значение часов реального времени в формате <day>-<month>-<year> <hour>:<min>:<sec>
Узел часов реального времени в отсутствие основного питания, питается от литиевого элемента. Поэтому при установленном элементе питания ход часов не зависит от наличия основного питания. Функция календаря учитывает високосные года. Вывести на печать время можно либо напрямую: `PRINT RLDT`
либо через строчную переменную, например

```
10 DIM $(10),20
20 $(2)=RLDT
30 PRINT $(2)
```

В качестве часов реального времени используется блок RTC Cortex-M3. В блоке используется внешний резонатор с частотой 32 768 Гц. Опорная частота делится до 1 Гц и этой частотой тактируется 32-битный счетчик. Исходя из глубины счетчика видно, что переполнение его произойдет более чем через 100 лет. Поэтому используется сквозной счет секундных импульсов, а реальная дата пересчитывается интерпретатором из текущего состояния счетчика. Другими словами нет аппаратных ресурсов впрямую соответствующих году, месяцу и т.д. , а происходит вычисление даты из состояния одного счетчика. При расчете учитываются високосные года.
Нулевое состояние счетчика принято соответствию 0:00:00 1 января 2000 года.
Данная информация может быть полезна для простого формирования временнЫх меток : при наступлении события достаточно запомнить состояние 32 битного счетчика. Ниже приведен пример считывания состояния счетчика:

```
10 rldt=24,12,10,11,51,20     ;установим время
20 print rldt                 ;проверим
30 phw regw(2818h), regw(281Ch),"h" ;посмотрим состояние счетчика RTC
40 delay=2                    ;подождем 2 секунды
50 phw regw(2818h), regw(281Ch),"h" ;убедимся что состояние изменилось на 2с
```

CLOCK

Функция **CLOCK** возвращает число эквивалентное текущему времени.
Время берется из RTC (узла часов реального времени).
Формат числа: десятые и сотые - это секунды, десятки и единицы - это минуты, тысячи и сотни - это часы.

```
Пример 1:
   A = clock ;если RTC в этот момент содержал значение 23ч59м58с, то A = 2359.58
Пример 2: включаем прибор на линии 3   с 8:00 до 8:30
   if (clock > 800) .and. (clock < 830) then pin(3)=1 else pin(3)=0
```

Прерывания BASIC-системы

ONPIN#

Оператор **ONPIN# [<ln> ,](pin)[=fun]** устанавливает номер строки **<ln>** для перехода на подпрограмму обработки программного прерывания при изменении состояния линии **pin** модуля. При этом для каждой из 36 возможных линий может указывается собственный обработчик. Номер линии необходимо указывать так называемый "короткий номер"(не линию порта микроконтроллера, например 0xA0 - это неправильно).
Значение **fun** задает ожидаемое изменение - фронт (1), спад (0) или деинициализация(0xFF) индивидуально для каждой линии. Параметры могут быть изменены последующими обращениями к оператору. При этом, если адрес обработчика не меняется, его можно не указывать.
Поскольку это программная эмуляция прерываний по изменению линий, существует временное ограничение на минимальную длительность импульсного сигнала - это величины порядка долей или единиц миллисекунд.
Возврат из обработчика должен быть обязательно оператором RETI. Пример:

```
10 onpin#2000,(24)=1 ;инициализация по фронту линии 24 с вызовом 2000 строки
```

ONTMR#

Оператор **ONTMR#[<ln> ,](n)[=time]** устанавливает номер строки **<ln>** для перехода на подпрограмму обработки программного прерывания при окончании указанного временного интервала **time** таймера **n(**0...7).
Можно использовать до 8 таймеров с вызовом соответствующего номеру таймера обработчика.
Время указывается в секундах в диапазоне 0.01 ... 4294967.
Если при повторном запуске таймера нет необходимости менять адрес обработчика, то его можно не указывать.
Возврат из обработчика должен быть обязательно оператором RETI. Пример:

```
10 ontmr#3000,(3)=1.23 ;инициализация таймера 3 на 1.23с с вызовом 3000 строки
```

До окончания работы обработчиков ONPIN и ONTMR, вновь возникшие запросы запоминаются, но не обрабатываются.

ONKEY

Оператор **ONKEY<ln>**устанавливает номер строки **<ln>** для перехода на подпрограмму обработки программного прерывания при получении символа по терминальному каналу. После инициализации все байты кроме cntrl-C будут вызывать передачу управления на указанную строку.
Возврат из обработчика должен быть обязательно оператором RETI.
Если важен не только факт прихода но сам код, то первым же оператором необходимо считать код функцией KEY, иначе он будет потерян.
Если было послано сообщение, то вероятней всего будет код последнего символа. Поэтому при использовании программы BASIC-Terminal, можно установить только факт нажатия, так как этот терминал производит только построчную передачу в канал. При использовании посимвольных терминалов, можно дифференцировать отдельные символы.

ONERR

Оператор **ONERR <ln>** устанавливает номер строки для перехода на обработку арифмети-ческой ошибки (переполнение, потеря значимости, деление на ноль). *Пример:*

```
10 onerr 100
20 Y=X/0
30 end
100 print “Арифметическая ошибка” : reti
```

ONINT1

Оператор **ONINT1 <ln>** устанавливает номер строки для перехода на подпрограмму обработки аппаратного прерывания линии 31(контакт 19 кросса) для MCU32.

Пример:

```
10 onint1 100
20 print RLDT : delay=0.1
30 goto 20
;Подпрограмма обработки прерывания INT1
100 print "INTERRUPT INT1" : reti
```

ONTIME

Оператор **ONTIME <iexpr>,<ln>** устанавливает время и номер строки для подпрограммы обслуживания программного прерывания по счетчику времени (см. оператор TIME). <expr>- значение времени (в секундах), по которому должно произойти прерывание программы, может принимать значения в диапазоне 0...65535 с. <ln> - номер строки, с которой начинается подпрограмма обработки прерывания.
Примечание. Возврат из подпрограммы обработки прерывания производится оператором RETI.

ONCAN$

См. раздел *Операторы для работы с шиной CAN*

Директивы (операторы выполняемые только из командной строки)

Директива ***RUN*** очищает память от всех переменных и запускает BASIC-программу.
В случае отсутствия программы, выдает версию BASIC

Директива ***HRUN*** запускает BASIC-программу без очистки переменных и массивов.

Директива ***NEW*** очищает память от программ и переменных.

Директива ***CONT*** продолжает выполнение BASIC-программы, остановленной оператором STOP или нажатием "Ctrl+C" на клавиатуре терминала. Для облегчения отладки можно вводить символ **“ \ ”** вместо ключевого слова CONT.

Директива ***LIST*** **<ln$_1$>-<ln$_2$>** производит вывод листинга фрагмента BASIC-программы (со строки <ln1> по строку <ln2>) на терминал. Числовые параметры не обязательны.

Директива ***SQUEEZE*** производит упорядочение программы во FLASH-памяти. Высвобождает неиспользуемые секторы памяти и ускоряет выполнение программы. Рекомендуется выполнять после коррекции программы через BASIC-терминал.

Директива ***REBOOT*** производит передачу управления ***Boot-loader Fractal*** версии не ниже 1-07.
Это позволяет произвести обновление версии интерпретатора без сброса питания и изменения состояния джамперов.
Работа с ***Boot-loader Fractal*** описана в «описании модуля MCU32».

Директива ***MTOP = <iexpr>*** используется для разделения пользовательской Flash-памяти контроллера на две части: кольцевой буфер текста BASIC-программы и буфер для долговременного сохранения пользовательских данных. *Применение MTOP вызывает стирание BASIC- программы* и размещение вновь заносимой программы с начала кольцевого буфера. Параметром директивы является число равное максимальному числу строк BASIC-программы помещающемуся в буфере. Причем речь идет не о максимальном номере строки, а именно о количестве строк. Поскольку Flash память разделена на сектора, запрашиваемое число строк округляется до величины соответствующей концу начатого сектора. Директиву необходимо применять только в случае если пользователь собирается пользоваться сохранением данных во Flash, в остальных случаях она не имеет смысла, и даже вредна, в том смысле , что уменьшает размер кольцевого буфера BASIC-программы и следовательно уменьшает ресурс циклов перепрограммирования этого участка. Параметр устанавливаемый директивой(число строк) запоминается в энергонезависимой памяти, поэтому нет необходимости лишний раз ее вызывать. Общая рекомендация – пользоваться ей не чрезмерно часто. Так, не рекомендуется на этапе отладки вносить ее в текст загружаемой программы по аналогии с NEW (тем более это не имеет смысла, т.к. однажды указанный размер буфера запоминается и нет необходимости его повторять).
Общая рекомендация: на этапе отладки всегда указывайте число строк кратно превышающее размер Вашей программы(чем больше тем медленнее будет расходоваться ресурс перепрограммирования Flash-памяти), а после того как программа отлажена – примените директиву с реально получившимся размером программы и при этом будет максимально возможный размер буфера для сохранения данных. Минимальное значение MTOP = 64.
После применения директивы интерпретатор возвращает число равное максимальному числу переменных которое можно сохранить в пользовательском буфере данных. Эта величина зависит от типа микроконтроллера(размера Flash) и объема зарезервированного буфера BASIC-программы. Причем обратите внимание, возвращаемое число это именно количество переменных, а не байт(одна переменная занимает 4 байта).
В новом модуле изначальное значение MTOP сохранено максимально возможным для этого типа микроконтроллера, это соответствует максимально возможному размеру кольцевого буфера под текст программы и отсутствию буфера под долговременно сохраняемые пользовательские данные.
При необходимости вернуть уже инициализированную директиву в состояние эквивалентное новому модулю (вся память используется под текст программы, место под данные не отведено), нужно применить MTOP = 0 .
Обратите внимание – работа с оператором/функцией EEPROM не имеет отношения к этим областям Flash-памяти и следовательно значение MTOP не оказывает никакого влияния на их работу.
Пример установки MTOP на буфер размером в 500 строк:

```
MTOP = 500
```

Директива ***INFO*** используется для выдачи сведений о модуле.
Директива выводит:
- текущее значение даты – времени энергонезависимых часов реального времени;
- тип модуля;
- тип микроконтроллера;
- серийный номер контроллера;
- версию интерпретатора;

- версию Boot-loader;
- текущий адрес MODBUS для USB канала;
- текущий адрес MODBUS для RS485 канала;
- текущий slave - адрес I2C области управляющих регистров;
- текущий slave - адрес I2C области данных;
- количество строк BASIC – программы зарегистрированных директивой MTOP;
- размер буфера для сохранения переменных во Flash зарегистрированных директивой MTOP;
- максимальное количество переменных в BASIC;
- максимальное количество индексированных переменных(элементов массивов);
- максимальное количество символов в символьных массивах;
- количество строк в текущей BASIC – программе;
- температура кристалла в градусах;
- текущая разница опорных напряжений Vdd и Vint_ref (в качестве опорного напряжения для встроенного АЦП используется основное напряжение питания микроконтроллера 3.3В, оно задается отдельным внешним для микроконтроллера стабилизатором. Кроме этого в самом микроконтроллере имеется собственный источник опорного напряжения номиналом 1.2В. Величина отношения в процентах показывает текущую разницу между ними. Как правило, при нормальном питании, типичное значение этой разницы в пределах одного процента);
- адрес технической поддержки;
- сайт разработчика.

Директива ***PROTECT*** защищает пользовательскую программу от считывания и копирования. В случае подачи директивы LIST в модуле где взведена защита, будет выдано сообщение о том что включена защита от считывания. Отключить защиту можно только директивой NEW, которая автоматически сотрет текст программы.

ПРИЛОЖЕНИЕ 1 ***Сообщения Basic системы***

"**Ready**" - готовность к вводу из командной строки.
"**Try again**" - введенная по приглашению оператора INPUT информация некорректна. Требуется повторить ввод.
"**Extra ignored**" - произошла попытка записи в строковую переменную большего количества символов, чем было объявлено в операторе DIM, лишние символы проигнорированы.
"**Stop - in line <ln>**" - программа остановлена оператором STOP или нажатием "Ctrl+C" на клавиатуре терминала. Выполнение программы может быть продолжено со строки <ln> вводом оператора CONT из командной строки.

Сообщения об ошибках

"Error: invalid line number"- в операторе применен номер несуществующей строки.
"Error: this variable is not defined" — данная переменная не объявлена
"Error: in loop FOR must be TO" — цикле FOR отсутствует в оператор TO
"Error: nesting level too big" — уровень вложенности больше допустимого
"Error: too many cikl or subroutine return" — нарушение условия возврата из цикла/ подпрограммы: отсутствует оператор GOSUB, соответствующий данному оператору RETURN (возврат из подпрограммы без вызова оной), либо отсутствует оператор DO, соответствующий данному оператору WHILE/UNTIL (встретился оператор завершения цикла при отсутствии оператора начала оного).
"Error: it must be THEN" — в цикле IF отсутствует оператор THEN
"Error: in loop ON must be GOTO or GOSUB" — в цикле ON отсутствует оператор GOTO/GOSUB
"Error: arg of loop ON too big - no branch" — значение аргумента в операторе ON больше, чем количество указанных переходов
"Error: stack border" — переполнение программного стека
"Error: this array also defined" — вторичное определение массива
"Error: error definition of array" — в операторе DIM не указана размерность массива
"Error: array border" — нарушение границ массива
"Error: no second commas" — нет закрывающей кавычки
"Error: (not find" — нет открывающей скобки
"Error: bad arg" - применен аргумент, значение которого выходит за пределы, допустимые для данного оператора или функции.
"Error: bad syntax" - BASIC-система обнаружила синтаксическую ошибку во время выполнения (интерпретации) строки программы.
"Error: math. Error" — арифметическая ошибка — переполнение, потеря значимости или деление на ноль
"Error: array size" - для массива, объявляемого оператором DIM:
не хватает места в памяти; размер объявляемого массива превышает 254; неверное имя индексированной переменной.
"Error: too many GOTO - stack full" — слишком много переходов в программе — стек переполнен
"Error: line longer then 80 symbol" — длина строки больше 80 символов.
"Error: no data" — нет данных в буфере для исполнения оператора READ.
"Error: unerased flash cell" — нестертая ячейка FLASH — запись данных невозможна.

ПРИЛОЖЕНИЕ 2 ***Распределение памяти RAM***

```
*****************************************************************
* Распределение памяти  RAM                                     *
* 0x0000-0x1FFF - значения переменных                           *
* 0x2000-0x3DFF - значения индексированных переменных (массивов) *
* 0x3E00-0x5DFF - значения строчных переменных                  *
* 0x5E00-0x7DFF - имена переменных                              *
* 0x7E00-0x7EFF - имена индексированных переменных (массивов)   *
* 0x7F00-0x7FFF – размерность + адрес начала массивов           *
*****************************************************************
```

ПРИЛОЖЕНИЕ 3 ***Встроенный протокол MODBUS***

В модулях MCU32 реализован встроенный протокол MODBUS. Работа через RS485 ведется через отдельный UART с использованием этого протокола.

ПРИЛОЖЕНИЕ 4 ***Параметры Fractal-BASIC-Cortex***

```
EEPROM(200h)  -адрес MODBUS канала RS485 UART3, производственная установка 4.
EEPROM(201h)  -адреса I2C, старший байт - slave-адрес области управляющих регистров,
           младший байт  - slave-адрес области данных,
           производственная установка 4 и 2 соответственно.
EEPROM(202h)  -адрес MODBUS канала USB, производственная установка 0 – соответствует
           выключенному режиму MODBUS для канала USB.
EEPROM(203h)  -адрес MODBUS канала RS485 UART1, производственная установка 2.
EEPROM(204h)  - старший байт - значение контрастности для модулей с ЖКИ,
           младший байт  - цвет подсветки по умолчанию
```

ПРИЛОЖЕНИЕ 5 ***Формат хранения переменных***

Значения переменных (кроме строчных) хранятся как числа в формате float (4-х байтовое представление IEEE754), соответственно максимально/минимально допустимые значения числовых переменных : +-2**[-126] /+-2**[128].

IEEE754 одинарная точность (single precision)

Знак – 1 бит, показатель степени – 8 бит (смещение 127), мантисса – 23 бита + 1 скрытый. Всего 32 бита.

Точность «в знаках»

В двоичных знаках понятно – раз мантисса 24 бита, значит и точность 24 двоичных знака. Разберёмся с десятичными. Поскольку для точности «в знаках» масштаб не имеет значения, умножим мантиссу на 2^{24}. Показатель степени на точность не влияет, так что его можно оставить как есть. Получившееся целое число мантисса представляет точно. Для его представления в десятичном виде нужно:

$\log_{10}2^{24} = 24 * \log_{10}2 = 7,2$ десятичных знака

Это значит, что погрешность двоичного представления числа не превосходит половины седьмого десятичного знака, даже чуть меньше.

Примеры:

Для примера, представим в этом формате число 1234,5.

- Переводим в двоичную форму. 1234,5 = 1024 + 210 = 2^{10} + 128 + 64 + 16 + 2 + 1/2 = 100 1101 0010,1 = …
- Теперь в экспоненциальную и нормализуем … = 1,00110100101 * 2^{10} = …
- Вспоминаем о смещении показателя степени … = 1,00110100101 * $2^{137-127}$ = 1,00110100101 * $2^{1000\ 1001\ -\ 0111\ 1111}$ = …
- Записываем (вспоминаем о скрытом бите) … = 0 – 1000 1001 – (1) 001 1010 0101 0000 0000 0000

Ещё несколько характерных примеров приведено в таблице :

	Знак	Показатель степени	Мантисса	Значение
1234,5	0	1000 1001	001 1010 0101 0000 0000 0000	1234,5
Ноль	0	0000 0000	000 0000 0000 0000 0000 0000	0
Один	0	0111 1111	000 0000 0000 0000 0000 0000	1
Наименьшее ненормализованное	0	0000 0000	000 0000 0000 0000 0001 0000	$1,4*10^{-45}$
Наибольшее ненормализованное	0	0000 0000	111 1111 1111 1111 1111 1111	$1,8*10^{-38}$
Наименьшее нормализованное	0	0000 0001	000 0000 0000 0000 0000 0000	$1,8*10^{-38}$
Наибольшее нормализованное	0	1111 1110	111 1111 1111 1111 1111 1111	$3,4*10^{38}$
Бесконечность	0	1111 1111	000 0000 0000 0000 0000 0000	

ПРИЛОЖЕНИЕ 6 *Поддерживаемые команды MODBUS*

Основной формат команды мастера для команд 0x70…0x7C

ADRMOD	COMAND	Adr_Hi	Adr_Lo	N	DATA_1	…	DATA_N	CRC_Lo	CRC_Hi

Команды поддерживаемые прошивкой.

03h **Read Holding Registers** (чтение регистров типа 4)
04h **Read Input Registers** (чтение регистров типа 3)
10h **Preset Multiple Registers** (запись регистров типа 4)
70h команда чтение RAM (оперативной памяти контроллера)
71h команда запись RAM (оперативной памяти контроллера)
72h команда чтение REG (регистровой памяти контроллера)
73h команда запись REG (регистровой памяти контроллера)
74h команда чтение EEPROM (энергонезависимой памяти данных)
75h команда запись EEPROM (энергонезависимой памяти данных)
76h команда чтение FLASH (памяти программ)
77h команда запись FLASH (памяти программ)
78h команда чтение идентификатора контроллера
79h команда запуск встроенного теста или инициализация контроллера
7Ah команда чтение шины I2C
7Bh команда запись шины I2C
7Ch команда ОБМЕН по шине SPI

Максимальное количество данных в пакете - не более 128 байт.

03h -> Команда Read Holding Registers

ADRMOD	03h	Adr_Hi	Adr_Lo	0	N*reg	CRC_Lo	CRC_Hi

ответ

ADRMOD	03h	N*2	data 1 Hi	data 1 Lo	…	data N Hi	data N Lo	CRC_Lo	CRC_Hi

04h -> Команда Read Input Registers

ADRMOD	04h	Adr_Hi	Adr_Lo	0	N*reg	CRC_Lo	CRC_Hi

ответ

ADRMOD	04h	N*2	data 1 Hi	data 1 Lo	…	data N Hi	data N Lo	CRC_Lo	CRC_Hi

10h -> Команда Preset Multiple Registers

ADRMOD	10h	Adr_Hi	Adr_Lo	0	N*reg	N*2	data1 Hi	data1 Lo	…	dataN Hi	dataN Lo	CRC_Lo	CRC_Hi

ответ

ADRMOD	10h	Adr_Hi	Adr_Lo	0	N*reg	CRC_Lo	CRC_Hi

70h / 72h(eeprom(0x205)=0) -> Команда ЧТЕНИЕ RAM / REG

ADRMOD	70h / 72h	Adr_Hi	Adr_Lo	N	CRC_Lo	CRC_Hi

ответ

ADRMOD	70h / 72h	Adr_Hi	Adr_Lo	N	DATA_1	…	DATA_N	CRC_Lo	CRC_Hi

71h / 73h(eeprom(0x205)=0) -> Команда ЗАПИСЬ RAM / REG

ADRMOD	71h / 73h	Adr_Hi	Adr_Lo	N	DATA_1	…	DATA_N	CRC_Lo	CRC_Hi

ответ

ADRMOD	71h / 73h	Adr_Hi	Adr_Lo	N	CRC_Lo	CRC_Hi

72h(eeprom(0x205)=1) -> Команда ЧТЕНИЕ бита RAM

ADRMOD	72h	Adr_Hi	Adr_Lo	N_Bit	CRC_Lo	CRC_Hi

ответ

ADRMOD	72h	Adr_Hi	Adr_Lo	N_Bit	0 / 0FFh	CRC_Lo	CRC_Hi

73h(eeprom(0x205)=1) -> Команда ЗАПИСЬ бита RAM

ADRMOD	73h	Adr_Hi	Adr_Lo	N_Bit	0/1…0FFh	CRC_Lo	CRC_Hi

ответ

ADRMOD	73h	Adr_Hi	Adr_Lo	N_Bit	CRC_Lo	CRC_Hi

74h -> Команда ЧТЕНИЕ EEPROM

ADRMOD	74h	Adr_Hi	Adr_Lo	N	CRC_Lo	CRC_Hi

ответ

ADRMOD	74h	Adr_Hi	Adr_Lo	N	DATA_1	…	DATA_N	CRC_Lo	CRC_Hi

75h -> Команда ЗАПИСЬ EEPROM

ADRMOD	75h	Adr_Hi	Adr_Lo	N	DATA_1	…	DATA_N	CRC_Lo	CRC_Hi

ответ

ADRMOD	75h	Adr_Hi	Adr_Lo	N	CRC_Lo	CRC_Hi

76h -> Команда ЧТЕНИЕ FLASH

ADRMOD	76h	Adr_Hi	Adr_Lo	N	CRC_Lo	CRC_Hi

ответ

ADRMOD	76h	Adr_Hi	Adr_Lo	N	DATA_1	…	DATA_N	CRC_Lo	CRC_Hi

!Команды 76h и 77h доступны если в интерпретаторе директивой MTOP разрешена Flash память пользователя.
Для доступа к памяти с адресами больше 0xFFFF используется страничный доступ. Для задания номера страницы необходимо вызвать команду 76h с числом читаемых байт=0 и младшем байте адреса указать номер страницы. Номер будет сохранен до выключения питания или нового задания номера. После сброса номер страницы равен 0.

77h -> Команда ЗАПИСЬ FLASH

ADRMOD	77h	Adr_Hi	Adr_Lo	40h	DATA_1	…	DATA_64	CRC_Lo	CRC_Hi

ответ

ADRMOD	77h	Adr_Hi	Adr_Lo	40h	CRC_Lo	CRC_Hi

78h -> Команда чтение идентификатора

ADRMOD	78h	CRC_Lo	CRC_Hi

ответ

ADRMOD	78h	Id_1	…	Id_252	CRC_Lo	CRC_Hi

79h -> Команда инициализация контроллера

ADRMOD	79h	55h	0AAh	CRC_Lo	CRC_Hi

79h -> Команда запуск встроенного теста

ADRMOD	79h	11h	1	CRC_Lo	CRC_Hi

ответ

ADRMOD	79h	Diagn	77h	…	77h	CRC_Lo	CRC_Hi

7Ah -> Команда ЧТЕНИЕ ШИНЫ I2C

ADRMOD	7Ah	Slave_Adr	Word_Adr	N	CRC_Lo	CRC_Hi

ответ

ADRMOD	7Ah	Slave_Adr	Word_Adr	N	DATA_1	…	DATA_N	CRC_Lo	CRC_Hi

7Bh -> Команда ЗАПИСЬ ШИНЫ I2C

ADRMOD	7Bh	Slave_Adr	Word_Adr	N	DATA_1	…	DATA_N	CRC_Lo	CRC_Hi

ответ

ADRMOD	7Bh	Slave_Adr	Word_Adr	N	CRC_Lo	CRC_Hi

7Ch -> Команда ОБМЕН по шине SPI

ADRMOD	7Ch	CS	0	N	Dout_1	…	Dout_N	CRC_Lo	CRC_Hi

ответ

ADRMOD	7Ch	CS	0	N	Din_1	…	Din_N	CRC_Lo	CRC_Hi

Формат квитанции об ошибке:

Квитанция об ошибке

ADRMOD	Command+80h	N Error	CRC_Lo	CRC_Hi

КОДЫ ОШИБОК

0x01-несуществующий код команды

(!обратите внимание, если данный узел/модуль не поддерживает какой-либо ресурс, например не имеет шины I2C, то при соответствующем запросе будет выдана эта ошибка)

0x02-длинна телеграммы запроса не соответствует содержанию команды

0x03-запрос на 0 байт

0x04-запрошенное количество байт больше чем 128

0x05-номер бита >7

0x06-попытка записи в несуществующий адрес

0x07-попытка чтения закрытого ресурса

0x08-попытка записи во FLASH не 64 байта

0x09-попытка записи не с начала блока 64 байта

0x0A-попытка записи закрытой области FLASH

0x0B-ошибка верификации после записи FLASH

0x0C-ошибка при приеме обязательной последовательности 55h 0AAh

0x0D-при обращении к каналу I2C шина занята > 10мс

0x0E-коллизия при обмене по I2C

0x0F-нет сигнала подтверждения ACK от Slave-устройства I2C

0x10-смежный канал UART занят

ПРИЛОЖЕНИЕ 7 ***Архив изменений и добавлений***

Список отличий Fractal-BASIC-Cortex от Fractal-BASIC-PIC, а так же нововведений, добавлений и исправлений.

5.08.2013 v3.02

Оператор ONINT1 переведен на аппаратное прерывание(до этого, линия обрабатывалась программно).
Это позволяет теперь фиксировать запросы на прерывания по этой линии на аппаратном уровне с максимальным разрешением по времени. Синтаксис оператора не изменился.

Добавлена возможность изменять величину TIMEOUT для канала I2C. При "зависании" шины, микроконтроллер по прошествии TIMEOUT * 100мкс реинициализирует работу шины. При этом сохраняется скорость работы канала, установленная на момент зависания шины. Для изменения величины TIMEOUT необходимо выполнить оператор `I2C#300,(0) = TIMEOUT`.
По умолчанию TIMEOUT равен 20, что соответствует 2мс.

Добавлена возможность отключения вывода лишних пробелов, которые оператор PRINT вставляет перед печатью числа и после него.

Параллельно с отключением вывода лишних пробелов запрещается вывод знака вопроса с последующим выводом числа(в случае когда число не вписывается в формат заданный оператором USING). Теперь в случае выхода числа за пределы формата, будут выводится символы "#" в форме заданной оператором USING. Данная возможность будет полезна при особенно работе с ЖКИ.

Для отключения вывода лишних пробелов и печати неформатного числа, необходимо в операторе SPC указать параметр 100. Для включения вывода пробелов необходимо в операторе SPC указать параметр 101.

По умолчанию вывод ведется как и раньше - с пробелами и печатью неформатных чисел.

Примеры:

```
120 print spc(100) ;отключить вывод пробелов и вывод неформатных чисел
...
340 print spc(101) ;включить вывод пробелов и вывод неформатных чисел
```

При работе в последовательными каналами в режиме MODBUS, теперь доступна команда **0x77** - команда записи данных во Flash память. Из за аппаратных особенностей используемых микроконтроллеров возможна запись только четного количества байт по четным адресам. Если процесс записи встречает не стертые ячейки, то автоматически стирается сектор к которому эти ячейки относятся.

Для удобства работы со временем добавлена функция **CLOCK**. При вызове она возвращает число эквивалентное текущему времени. Время берется из RTC(узла часов реального времени).
Формат числа: десятые и сотые - это секунды, десятки и единицы - это минуты, тысячи и сотни - это часы.

Пример 1:

```
A = clock ;если RTC в этот момент содержал значение 23ч59м58с, то A = 2359.58
```

Пример 2: включаем прибор на линии 3 с 8:00 до 8:30

```
if (clock > 800).and.(clock < 830) then pin(3)=1 else pin(3)=0
```

31.03.2013 v3.01

Для модулей с установленным чипом драйвера CAN добавлены операторы/функция **CAN**. Они позволяют вести обмен данными через шину CAN.
Для модулей на базе STM32F103 использование этих операторов блокирует работу с USB. Это связано с архитектурой этих кристаллов, исключающей одновременную работу USB и CAN. Неудобства, создаваемые этой особенностью, частично компенсируются тем, что USB канал в наших модулях используется в основном для загрузки программы. И поэтому в тех случаях когда необходимо использование канала CAN , узел CAN инициализируется не при старте, а при первом вызове операторов/функций CAN$ и ONCAN$. Процесс загрузки программы через USB и последующая работа с каналом не пересекаются. Остается, конечно, неудобство что нельзя при работе с CAN использовать USB для вывода терминальных сообщений, но как правило это можно либо обойти, либо воспользоваться другими каналами вывода(RS485, I2C, SPI, вывод на ЖКИ).

Функция/оператор **CAN$** позволяет послать пакет данных(Data Frame) или пакет удаленного запроса(Remote Frame) со стандартным(St_ID) или расширенным(Ex_ID) идентификатором. До вызова функции, в памяти должен быть размещен передаваемый кадр - последовательно 4 байта ID и требуемое количество данных : 0...8 байт.
ID размещается с выравниванием к младшему разряду. Для случая с Ex_ID, оставшиеся старшие 3 бита игнорируются,

для St_ID игнорируется 21 старший бит.
В качестве параметров оператора/функции указываются адрес расположения в памяти передаваемого кадра, а так же число отправляемых байт данных n содержащихся в кадре. Если n = 10...19, то размещенный в памяти ID будет трактоваться как Ex_ID 29 бит, если n = 0...9 или отсутствует, то как St_ID 11 бит.
Если параметр n равен 9, 19 или отсутствует, то будет послан кадр "Remote Frame".
Функция возвращает номер mailbox (1...3) в который попал этот пакет или число 4, в случае если все mailbox заняты и пакет не размещен.
```
x = CAN$ adr, n
```

Оператор **ONCAN$** позволяет настроить узел CAN на прием пакетов. До первого вызова оператора должен быть подготовлен буфер размером не менее 12 байт, где заранее размещены фильтр ID 4 байта, его маска 4 байта и будут размещаться приходящие кадры, включающие в себя 4 байта ID и данные до 8 байт.
ID и маска выравниваются к младшему биту. Установка бита(ов) маски в 1 делает соответствующий(е) бит ID существенным при фильтрации приходящих пакетов.
Параметр оператора n устанавливает режим фильтра:
Если n < 10 то будут приниматься только кадры с St_ID, если 10 <= n < 20 то только кадры с Ex_ID.
Если младший десятичный знак n = 9 то будут приниматься только кадры "Remote Frame", если n = 0 то только кадры "Data Frame".
Если параметр n отсутствует, то будут приниматься кадры "Remote Frame" и "Data Frame" с любым типом ID.
Для этого случая в памяти должны быть размещены ID и маска в формате Ex_ID 29 бит.
После прихода кадра удовлетворяющего условиям фильтра и маски, с 0-го по 3-й байт будет размещен пришедший ID, а начиная с 4-го байта - данные. При этом будет передано управление подпрограмме с номером строки указанном в первом параметре оператора: `line`. Второй параметр оператора : `adr` - это адрес ячейки начала буфера приема.
Можно объявить до 7 разных подпрограмм с индивидуальными комплектами параметров. Комплекты параметров привязываются к номеру строки обработчика. Параметры уже объявленной подпрограммы можно переобъявить вызвав этот оператор с номером строки этой подпрограммы и адресом нового буфера. В случае переобъявления параметров обязательно надо выбрать другой заранее подготовленный буфер с комплектом фильтра и маски, т.к. текущий буфер занят приемом до момента переинициализации.
Возврат из подпрограммы обработки этого прерывания должен осуществляться оператором RETI.
```
ONCAN$ line, adr, n
```

Функция **CAN** возвращает признак стандартного или расширенного ID и число байт данных в пришедшем кадре. При значениях 0...9 это стандартный ID, при 10...19 - расширенный. Последняя десятичная цифра 0...8 это число пришедших байт данных. Если последняя десятичная цифра равна 9, значит пришел кадр удалённого запроса данных.
```
x = CAN
```

Оператор **CAN** позволяет установить скорость работы шины.
Доступны величины 1000, 500, 250, 100, 50, 20, 10 кБит/с.
По умолчанию скорость предустановленна на 1000 кБит/с. Если для работы требуется другая скорость, то до первого использования CAN$ или ONCAN$ нужно указать одну из перечисленных скоростей.
```
CAN = 250
```

```
Пример №1 передача:
5 delay=5 ;пауза при старте, для возможности останова программы до инициализации CAN
10 dim $(2),12 ;буфера для исходящих телеграмм
20 adr_1=loc($(0)) ;получим адрес буфера передачи 1
30 adr_2=loc($(1)) ;получим адрес буфера передачи 2
40 mem(adr_1)=0x78,0x56,0x34,0x12 ;занесем ID 29 бит 0x12345678
50 mem=0x11,0x22,0x33,0x44,0x55 ;занесем данные 5 байт
60 mem(adr_2)=0x65,7,0,0 ;занесем ID 11 бит 0x765
70 mem=0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17 ;занесем данные 8 байт;
80 can=20 ; выбираем скорость 20 кБит/с
;основной цикл
100 if pin(23)=1 then goto 200 ;определим состояние линии 23 и выберем телеграмму
110 x1=can$ adr_1,15 ;пошлем буфер №1 5 байт с расширенным ID, x1->подтверждение
120 goto 300
200 x2=can$ adr_2,8 ;пошлем буфер №2 8 байт со стандартным ID, x2->подтверждение
300 delay=0.1 ;пауза
310 goto 100 ;циклимся

Пример №2 прием:
5 delay=5 ;пауза при старте, для возможности останова программы до инициализации CAN
```

```
10 dim $(2),12 ;буфера приема
20 adr_1=loc($(0)) ;получим адрес буфера 1
30 adr_2=loc($(1)) ;получим адрес буфера 2
40 can=20 ; выбираем скорость 20 кБит/с
50 mem(adr_1)=0x78,0x56,0x34,0x12 ;занесем фильтр 29 бит 0x12345678
60 mem=0xFF,0xFF,0xFF,0x1F ;занесем маску - значимы все 29 бит
70 mem(adr_2)= 0x65,7 ;занесем фильтр 11 бит 0x765
80 mem(adr_2 + 4)=0xFF,7 ;занесем маску - значимы все 11 бит
90 ONCAN$ 1000,adr_1,10 ;инициализация приема в буфер adr_1 ,обработчик 1000, ExID
95 ONCAN$ 2000,adr_2,0 ;инициализация приема в буфер adr_2 ,обработчик 2000, StID
;
100 delay=0.01
105 pin(24)=0: pin(25)=0 ;гасим светодиоды
110 goto 100
;
;попадаем сюда при приходе пакета от CAN с ID=0x12345678
1000 pin(24)=1 ;вкл светодиод
;в буфере adr_1 находится пришедший кадр
1010 n_1=can ;число пришедших байт и тип ID(при исп.примера №1 будет равен 15)
1020 reti ;возврат из обработчика
;
;попадаем сюда при приходе пакета от CAN с ID=0x765
2000 pin(25)=1 ;вкл светодиод
;в буфере adr_2 находится пришедший кадр
2010 n_2=can ;число пришедших байт и тип ID(при исп.примера №1 будет равен 8)
2020 reti ;возврат из обработчика
```

12.02.2013 v3.00

Интерпретатор переведен на RTOS. Это позволило оптимизировать работу многих имеющихся функций, добавить новые, и подготовить почву для дальнейшего развития.

Добавлена функция **DS18B20**. Она максимально упрощает работу с цифровыми термодатчиками DS18B20 с интерфейсом MicroLan. Теперь не надо заботиться о протоколе, настройке линий, запуске преобразования и контроле CRC8. Достаточно просто вызвать эту функцию, указав необходимую линию, и без ожидания получить свежий результат измерения в градусах Цельсия с полной диагностикой линии. Интерпретатор теперь позволяет параллельно работать с датчиками на 31(48) линии, при этом это никак не отражается на работе бейсик программы.
При работе с датчиками поддерживается безадресный режим с / без паразитным питанием.
Диагностика подключения возвращается в самом результате:

-при несовпадении CRC8 возвращается значение "-100.0";
-при отсутствии ответа(обрыве) "-200.0";
-при КЗ на линии "-300.0".

После первого обращения к линии, внутренний автомат начинает опрос датчика на этой линии ~ каждые 750мс, постоянно обновляя результат.
При задании номера линии допустимо указывать и "короткий " номер линии и абсолютный номер соответствующий нумерации портов микроконтроллера.

Примеры:

```
x = DS18B20(24) ; присваивание x результата измерения DS18B20 на линии 24
print DS18B20(0xA3) ; печать температуры датчика подключенного к линии PA3
```

Добавлена возможность фазового управления тиристорными регуляторами мощности. Для этого доступно 3 группы каналов, каждая с отдельным входом детектора и тремя выходами. Всего до 9 выходов.
На линию приходящую на первый канал каждого из таймеров подается сигнал с фазового детектора. В простейшем случае вполне достаточно всего 2 компонентов:

- резистора сопротивлением порядка 220 кОм / не менее 0.5 Вт и допускающим падение на нем не менее 400В;
- транзисторной оптопары с биполярным входом, например PC814.

На вход оптопары через резистор подается сетевое переменное напряжение, а выходной транзистор подключается непосредственно между землей модуля и входом фазового детектора без дополнительных компонентов. При этом не должно быть никаких соединений между высоковольтной частью и модулями, сигнал передается от сети только через оптопару. Подтягивающие резисторы к выходу оптопары не нужны.

Второй, третий и четвертый каналы таймеров могут быть проинициализированы как выходы на фазовое управление. К ним через ограничивающий резистор можно подключать оптотриаки *без "детектора нуля"*. Для оптотриаков с входным управляющим током 10 мА это резистор порядка 200 Ом.
Данные рекомендации даны из расчета, что пользователь имеет необходимые знания нужные для работы с высоким напряжением и мерах безопасности при этом.
!Если у Вас нет нужной квалификации для работы с сетевым напряжением - не используйте наши модули для этого!
Управление нагрузками осуществляется через оператор **PWM**.
Для активации режима фазового регулирования при инициализации таймера оператором PWM задается глубина регулирования в микросекундах со знаком "-" . Для оператора PWM отрицательная величина длительности - признак режима фазового регулирования.

```
пример:  PWM(2)=-7000 ;инициализация таймера 2 с глубиной регулирования 7мс
```

Пользователь имеет возможность самостоятельно задавать глубину регулирования в зависимости от конкретной реализации своей схемы и типа нагрузки. Вероятно, это окажется полезным.
Для активации конкретного выходного канала и задания момента открытия используется оператор PWM с указанием номера таймера совмещенного с номером линии и величина открытия тиристора в долях от единицы - аналогично обычному режиму ШИМ оператора PWM.

```
пример:  PWM(23)=0.3 ;включить 3й канал таймера 2 с моментом открытия 30% от 7мс
```

Поскольку основой для реализации этой функции служат аппаратные таймеры, микроконтроллер не тратит программного времени на фазовое регулирование, оно работает полностью прозрачно и независимо от программы.

Для **RS485, PUT$, GET$** добавлена возможность работы со всеми доступными каналами UART(до 5). При первом обращении к каналу происходит его инициализация с параметрами 115200/8/1/0. При необходимости параметры можно изменить операторами RS485F#x400, где x - номер UART.

```
Пример:
...
120 rs485f#2400,(0) = 9600, 8, 1, 0 ; инициализация канала UART2
130 put$ 2, adr ; инициализация указателя передачи канала UART2 на adr
140 x=get$ 3, adr1 ; инициализация указателя приема канала UART3 на adr1
...
400 put$ 2,6 ; посылка в UART2 6 байт начиная с adr
...
620 x=get$ 3,14 ;запрос из UART3 14байт с размещением с adr1,x-число полученных
```

18.09.2012 v2.05

Накопительное обновление.

Для модулей имеющих в своем составе ЖКИ добавлены демо-режимы. Запуск их осуществляется если при подаче питания зажата соответствующая кнопка.

Для оператора **ONPIN** расширено число линий до 36. Это позволяет использовать линии разъемов расширения в MCX53-32.x .

Для операторов **ONPIN** и **ONTIME** заблокирована возможность повторного вхождения. До окончания работы обработчика, вновь возникшие запросы запоминаются, но не обрабатываются.

В оператор I**2C** добавлена возможность не записывать данные т.е. если написать i2c#A,(W): то уйдет пакет только со slawe и word адресами с признаком записи. Это сделано для совместимости с приборами где требуется посылка только 2 байтов - slave-адреса и одного байта. Для этого режима не предусмотрена диагностика ответа ACK.

Добавлена функция стирания BASIC- программы и занесения начальных установок. Функция работает аналогично директиве отладчика "K". Это полезно если возникнет "зависание" модуля при отладке программ.
При этом не нужно переходить в режим загрузчика для подачи директивы "K".
Для запуска процедуры нужно на "зависший" модуль одеть джампер, который предназначен для запуска загрузчика(см. описание на соответствующий модуль).
При этом загорится красный светодиод. После примерно 5с красный светодиод погаснет и загорится желтый. Это означает что бейсик программа удалена и занесены заводские настройки. Если снять джампер до загорания желтого светодиода, то модуль останется в исходном состоянии.
Если в момент "зависания" был открыт BASIC-Terminal, то необходимо его закрыть и после этого сбросить модуль(выключить и включить). После прошествия нескольких секунд, требующихся WIN на опознавание вновь подключенного USB устройства, можно запускать заново BASIC-Terminal.

Для модулей в составе которых есть ЖКИ и кнопки - MCX53-20 и MCX53-21, добавлена возможность генерации прерывания при нажатии любой из кнопок. Прерывание генерируется на линии SDO межмодульного разъема на который выведена шина I2C. Это позволяет мастер-модулю, при работе с этими терминальными модулями через межмодульный разъем, не заниматься периодическим опросом состояния кнопок, а воспользоваться оператором ONPIN для генерации прерывания в своей программе.

04.04.2012 v2.04

Добавлен оператор **ONPIN#**. Он позволяет прерывать программу и передавать управление подпрограммам, при изменении состояния линий модуля. При этом для каждой из 32 линий указывается собственный обработчик. Можно так же задавать ожидаемое изменение - фронт или спад индивидуально для каждой линии. При необходимости конкретное прерывание может быть деинициализированно или проинициализировано с другими параметрами.
Поскольку это программная эмуляция прерываний по изменению линий, существует временное ограничение на минимальную длительность импульсного сигнала - это величины порядка долей или единиц миллисекунд.
Первым параметром указывается номер строки подпрограммы обработчика.
Второй параметр - это номер линии, причем необходимо указывать так называемый "короткий номер"(не линию порта микроконтроллера, например 0xA0 - это неправильно).
Третий параметр - это ожидаемое событие : 0-> спад, 1-> фронт, 0xFF-> деинициализация.
Параметры могут быть изменены последующими обращениями к оператору. При этом, если адрес обработчика не меняется, его можно не указывать.
Возврат из обработчика должен быть обязательно оператором RETI.
Старый аналогичный оператор ONINT1 который позволял генерировать прерывание по спаду одной линии остался без изменений для совместимости программ. При написании новых программ рекомендуется использовать ONPIN#.

Пример:

```
onpin#2000,(24)=1 ;инициализация по фронту линии 24 с вызовом 2000 строки
```

Добавлен оператор **ONTMR#**. Он позволяет использовать до 8 таймеров с вызовом соответствующего номеру таймера обработчика.
Первым параметром указывается номер строки подпрограммы обработчика.
Второй параметр - это номер таймера : 0...7.
Третий параметр - это время в секундах : 0.001 ... 4294967.
Если при повторном запуске таймера нет необходимости менять адрес обработчика, то его можно не указывать.
Возврат из обработчика должен быть обязательно оператором RETI.
Старый аналогичный оператор ONTIME остался без изменений для совместимости программ. Он позволяет работать только с точностью до секунды в диапазоне до 65535с.

Пример:

```
ontmr#3000,(3)=1.234 ;инициализация таймера 3 на 1.234с с вызовом 3000 строки
```

Добавлен оператор **ONKEY.** Он позволяет прервать программу при получении символа по терминальному каналу и передать управление на указанную строку обработчика. После инициализации все байты кроме cntrl-C будут вызывать передачу управления на указанную строку.
Единственным параметром оператора является номер строки вызываемой подпрограммы.
Возврат из обработчика должен быть обязательно оператором RETI.
Если важен не только факт прихода но сам код, то первым же оператором необходимо считать код функцией KEY, иначе он будет потерян.
Если было послано сообщение, то вероятней всего будет код последнего символа. Поэтому при использовании программы BASIC-Terminal, можно установить только факт нажатия, так как этот терминал производит только построчную передачу в канал. При использовании посимвольных терминалов, можно дифференцировать отдельные символы.

Операторы разрешающие работу с прерываниями BASIC- программы сведены в отдельный раздел: **Прерывания BASIC-системы.**

Добавлена поддержка команд MODBUS 0x03 -> **Read Holding Registers** (чтение регистров типа 4), 0x04 -> **Read Input Registers** (чтение регистров типа 3), 0x10 -> **Preset Multiple Registers** (запись регистров типа 4).

Добавлена информация в раздел "Работа с шиной I2C".

10.01.2012 v2.03

Добавлен оператор **PWM**. Он позволяет использовать аппаратные ШИМы микроконтроллера. Одновременно

доступно до 12 линий ШИМ. Точное число доступных линий определяется конкретным типом модуля(см. описания модулей). Для работы в режиме ШИМ в интерпретаторе доступны до трех таймеров(2, 3, 8) по четыре канала(1-4) в каждом. Время периода задается в микросекундах индивидуально для каждого из таймеров. Диапазон от 1 до 65535мкс. Величина ШИМ задается индивидуально для каждого выбранного канала в долях от целого: 0 ... 1.0. Минимальный квант выбран равным 1 мкс. Соответственно чем меньше период, тем разрешающая способность ШИМ меньше. При максимальном периоде она равна 1/65536 (16 бит).
Для инициализации необходимо сначала настроить режим таймера выбранного канала. Для этого в параметре указывается номер таймера, после равенства задается период ШИМ в микросекундах.

Примеры:

```
pwm(2)=10000 ;инициализация TIM 2 в ШИМ с периодом 10000 мкс = 100 Гц
pwm(8)=500 ;инициализация TIM 8 в ШИМ с периодом 500 мкс = 2000 Гц
```

При необходимости можно многократно изменять период таймера.
Инициализация не переводит линии микроконтроллера в режим ШИМ. Для начала работы выбранной линии необходимо обратиться к ней, указав значение ШИМ, при этом линия автоматически настраивается на выход и начинает выдавать ШИМ сигнал с заданными параметрами. Линия указывается в параметре вместе с номером таймера. При этом десятки это номер таймера, единицы это номер выбранного канала этого таймера.
Соответствия ножкам разъемов и количество доступных линий см. в описании на конкретный модуль.

Примеры:

```
pwm(21)=0.5 ;перевод соотв.линии канала 1 TIM 2 на выдачу ШИМ скважностью 50%
pwm(34)=0.123 ;перевод соотв.линии канала 4 TIM 3 на выдачу ШИМ скважностью 12.3%
pwm(82)=0.0001 ;перевод соотв.линии канала 2 TIM 8 на выдачу ШИМ скважностью 0.01%
```

Соответствия каналов конкретного таймера ножкам разъемов и количество доступных линий см. в описании на конкретный модуль.

Доработан оператор **BEEP**. Теперь его можно использовать для выдачи звукового сигнала в линию во всех модулях, даже если в них не предусмотрена установка динамика. Для начала работы необходима инициализация. Для выбора линии доступны до 12 линий(определяется конкретным типом модуля см. описания модулей). Для работы оператор использует один из трех(2,3,8) таймеров.
!Обратите внимание - не предусмотрена одновременная работа конкретного таймера одновременно с BEEP и PWM. Т.е. если конкретный таймер инициализируется для работы с BEEP, то не получится использовать свободные каналы этого таймера для работы с ШИМ.
Оператор работает похожим образом с PWM, доступны одни и те же таймеры и их каналы.
При инициализации в параметре необходимо указать номер таймера и канала - таймер в десятках, канал в единицах.
!Обратите внимание при инициализации параметр ОДИН!

Примеры:

```
beep 21 ;перевод соотв.линии канала 1 TIM 2 на выдачу звукового сигнала
beep 32 ;перевод соотв.линии канала 2 TIM 3 на выдачу звукового сигнала
beep 84 ;перевод соотв.линии канала 4 TIM 8 на выдачу звукового сигнала
```

После проведенной инициализации можно работать с BEEP как и раньше, указывая частоту и длительность сигнала.

Пример:

```
beep 1000,0.3 ;выдача звука частотой 1000 Гц длительностью 0.3с
```

!!!В модулях в которых предусмотрен динамик, проводить инициализацию канала звука не надо!

В операторе LAN для модулей со сложными выходами сделана автоматическая инициализация линий (подтяжек) для адресов линий < 12.

Оператор LAN оптимизирован и для работы с приборами совместимыми с оригинальными приборами DALLAS.

Исправлена проблема с выдачей русских букв на ЖКИ.

Максимальное количество переходов GOTO/GOSUB... увеличено с 200 до 400.

Исправлена неправильная работа GOSUB в теле ON.

Исправлен CHR($(),i).

Доработан RS485#.

Исправлен RS485A.

Исправлен MOVE.

Исправлен INPUT.

Исправлен MOVE.

29.09.2011 v2.02

Для модулей с графическим встроенным ЖКИ с контроллером PCF8531 подключенным по I2C добавлены операторы управления режимами:

Управление режимами вывода:
Для управления режимом вывода применяется оператор **LCD**(111) = …
0x01 -> установить режим вывода с забоем поверх, в случае вывода символов рисуется и символ и его фон поверх бывшего изображения;
0x02 -> режим вывода наложением. т.е. фон остается прежним, рисует только перо;
0x03 -> режим вывода «исключающий или»;
0x04 -> режим вывода инверсный (перо белое);
0x05 -> режим вывода прямой (перо черное);
0x06 -> режим вывода спецсимволов нормального размера, символы как при обычном выводе, но координата задается графическим курсором;
0x07 -> режим вывода спецсимволов широкий -:- ;
0x08 -> режим вывода спецсимволов высокий -:- ;
0x09 -> режим вывода спецсимволов широкий + высокий -:- ;
0x0A -> режим вывода обычных символов по знакоместам;
0x0C -> курсор выключить;
0x0E -> курсор включить мигающее подчеркивание;
0x0F -> курсор включить мигающее знакоместо;
0x00 -> ничего не изменять.
После включения питания или сброса, модуль устанавливает следующие параметры:
вывод обычных символов по знакоместам;
режим вывода с забоем поверх;
режим вывода прямой (перо черное);
координаты символьного и графического курсоров обнуляются;
курсор – выключен.

Команда очистки экрана
Для очистки/заполнения/инверсии применяется оператор **LCD**(130) = …
0x00 -> производит заполнение экрана нулями и сбрасывает символьный курсор;
0x01 -> производит заполнение экрана единицами и сбрасывает символьный курсор;
0x02 -> инвертирует экран и сбрасывает символьный курсор;

Команда установка курсора, его типа и режима вывода
Для управления курсором применяется оператор **LCD**(120) = x , y, <rej>
При работе в режиме эмуляции алфавитно-цифрового режима(далее по тексту – символьного) это номер знакоместа соответственно в строке и ряду. Счет ведется начиная с 0-ой позиции 0-ой строки(верхней).
При работе в графическом режиме(далее по тексту – спецсимволов) это координаты левого нижнего угла выводимого сообщения.
Оба режима(символьный и спецсимвольный) работают одновременно и прозрачно, регистры курсора у каждого свои. Просто для правильной работы этой команды есть программный переключатель на нужный комплект регистров в зависимости от контекста параметра <режим> или, в случае его отсутствия, прошлого значения этого параметра.
Необязательный параметр <режим> устанавливает режимы вывода и вид курсора.
Ниже описана отдельная команда управления режимом. Действие этого параметра там полностью идентичное. Здесь опишем команды управления курсором, остальные ниже.
0x0C -> курсор выключить;
0x0E -> курсор мигающее подчеркивание;
0x0F -> курсор мигающее знакоместо.
При выводе спецсимволов отрисовка курсора не поддерживается, координата графического курсора сдвигается только вправо на новую позицию вплоть до конца экрана.
!Включение курсора значительно замедляет скорость вывода символов.

Для совместимости с модулями производства Фирмы Фрактал с базовой прошивкой для PIC18, добавлены команды MODBUS для работы с битами -> команды 0x72(чтение бита) и 0x73(запись бита). Эти комбинации в Fractal-

BASIC-Cortex уже используются под команды чтение и запись регистров специального назначения.
Выбор между этими группами осуществляется записью в ячейку EEPROM(0x205) значения 0 или 1.
Новые модули поставляются с предустановленным значением 0, что соответствует принятым до этого в Fractal-BASIC-Cortex командам чтение/запись регистров. При занесении в эту ячейку единицы, модуль начинает интерпретировать эти комбинации как чтение / запись бита. Поскольку нужное значение заносится в EEPROM, то достаточно один раз сделать эту запись, она будет сохраняться при отключении питания. При работе с битовыми командами теперь допустимо указывать номер бита от 0 до 15.

Исключены операторы и функции CHSM и CHSMR. Вместо них введены функции CRC16 и CRC8.
Новые функции вычисляют соответственно CRC16 для MODBUS RTU и CRC8 для MicroLan.
В качестве параметров указываются - начальная ячейка и число ячеек. Для CRC16 максимальное число ячеек 65535, для CRC8 - 256. Если адрес указан как положительное число, то идет работа с RAM, если отрицательный, то с FLASH.

При использовании режима входов PT1000(реализованы не во всех модификациях), теперь доступны все форматы преобразования - вольты, единицы АЦП и проценты.

Оператор LAN теперь поддерживает режим паразитного питания датчика.

18.07.2011 v2.01

Приведены в соответствие оператор и функция MEMW. Младший по адресу в RAM байт соответствует младшему байту в операторе/функции.

Для модулей имеющих в своем составе графический ЖКИ, добавлены операторы POINT, LINE, RECT и COLOR.

Оператор **POINT** выводит точку с координатами X и Y.
```
Пример:     POINT 107, 23     ; вывод точки с координатами x=107, y=23
```

Оператор **LINE** выводит линию с координатами от прошлого значения POINT или LINE до X и Y.
```
Пример:     LINE 67, 14 ; вывод линии в точку x=67, y=14
```

Оператор **RECT** выводит прямоугольник с координатами от прошлого значения POINT или LINE до X и Y.
```
Пример:     RECT 49, 30 ; вывод прямоугольника в точку x=49, y=30
```

Оператор **COLOR** управляет яркостью и цветом подсветки ЖКИ. Если указан один параметр, то он расценивается как цвет из палитры 14 основных цветов. Если заданы три параметра, то они задают яркость красного, зеленого и синего соответственно. Диапазон 0…64.
```
Пример:     COLOR 8     ; включить ярко красную подсветку
            COLOR 32,32,32    ; включить подсветку белого на половинную яркость
```

Оператор **BEEP** обеспечивает подачу звукового сигнала на модулях где установлен динамик.
Первый параметр – частота в герцах, второй – длительность в секундах 0…9.999с. После обработки этого оператора интерпретатор сразу переходит к обработке следующего, т.е. воспроизведение звука не приостанавливает работу программы. Для случаев, когда требуется ожидание завершения звука, например проигрывание мелодии, длительность необходимо указывать на 10 больше. Если в качестве частоты указан 0, то будет проиграна пауза нужной длительности.
```
Пример «Былененький он зел»:
10 beep 1335,10.5
20 beep 1000,10.5
30 beep 1335,10.5
40 beep 1000,10.5
50 beep 1335,10.5
60 beep 1260,10.25
70 beep 0,10.25
80 beep 1260,11
```

19.06.2011 v2.00

Архив изменений перенесен в приложение №7 в конец документа.

Расширена функциональность интерпретатора для раскрытия в полной мере особенностей новых модулей MCX53-32.x и MCX53-20.x. Добавлены:

- поддержка одновременно двух каналов RS485;
- поддержка многофункциональных линий ввода-вывода с автоматической конфигурацией внешних входных цепей;
- поддержка работы с графическим ЖКИ TIC32 (драйвер PCF8531) через дополнительный канал I2C и управление яркостью RGB подсветки этого индикатора;
- поддержка работы со звуковым выходом.

Для доступа к еще одному каналу RS485 в операторе / функции RS485 необходимо при указании адреса устройства на шине указать адрес на 1000 больше. Для канала 1 (UART3) применяются адреса без изменений, для канала 0 (UART1) указывается адрес на 1000 больше. Для изменения параметров канала 0 (UART1) нужно указывать адрес не 400 , а 1400 : `RS485F#1400,(0) = ...`

Для оператора PUT и функции GET тоже добавлена возможность работы со вторым каналом RS485. При сохранении уже имеющегося формата команд , добавлен упрощенный вариант обращения.
Перед работой канала требуется провести инициализацию – указать начало буфера и режим приема(пакетный/ побайтный). После этого, при работе просто указывается номер канала и число байт. Примеры:

```
PUT$ 0,ADR1; Инициализация канала 0(UART1) с указателем ADR1
PUT$ 0,0x100; Инициализация канала 0 с таймаутом
PUT$ 1,0x101; Инициализация канала 1(UART3) без таймаута
PUT$ 0,5; канал 0, послать 5 байт из предварительно указанного буфера
X = GET$ 1,7; канал 1, проверить сколько пришло и передать в буфер 7 байт
```

Введена поддержка многофункциональных линий ввода-вывода, работа которых обеспечивается несколькими линиями микроконтроллера. Такие линии есть в серии MCX53-32.x. В таких линиях одна линия (основная)выполняет функцию ввода(вывода), а другая в паре с соответствующими джамперами конфигурирует режим работы основной линии. Для работы и конфигурирования режимов используется оператор / функция PIN. Для задания режима линии необходимо вызвать PIN с указанием номера линии и номера режима в котором эта линия будет работать далее и установить соответствующий джампер в нужное положение.
!Обратите внимание , конфигурирование линий допустимо для упрощенных(косвенных) адресов линии(соответствуют расположению на разъемах и имеют адреса меньше чем 32) и для модулей у которых есть вспомогательные конфигурирующие линии.
Для MCX53-32.x теперь доступны следующие режимы:

Параметры инициализации сигнальных линий для оператора PIN MCX53-32.x			
Номер режима	**Описание**	**джампер**	**Результат**
0	Дискретный выход 0В без ограничительного резистора	2-3	Выход = 0
1	Дискретный выход +3.3В без ограничительного резистора	2-3	Выход = 1
0x10	Дискретный выход 0В с ограничительным резистором 1К	2-3	Выход = 0 через 1К
0x11	Дискретный выход +3.3В с ограничительным резистором 1К	2-3	Выход = 1 через 1К
0x80	Режим дискретного входа с подтяжкой 0В / ~ 40 К	2-3	X = PIN(N) ; X = 0/1
0x81	Режим дискретного входа с подтяжкой +3.3В / ~ 40 К	2-3	X = PIN(N) ; X = 0/1
0x82	Режим плавающего дискретного входа 0 / +3.3В	снят	X = PIN(N) ; X = 0/1
0x90	Режим дискретного входа с подтяжкой 0В / 1 К (делитель 0 / +24В)	снят	X = PIN(N) ; X = 0/1
0x91	Режим дискретного входа с подтяжкой +3.3В / 1 К («сухой контакт»)	2-3	X = PIN(N) ; X = 0/1
0xC0	Аналоговый режим потенциального входа 0…+3.3В	снят	X = ADC(N) ; X = 0 … 3.3
0xD0	Аналоговый режим потенциального входа 0…+10В(+24В)	снят	X = ADC(N) ; X = 0 … 24
0xD1	Режим работы с термодатчиками PT1000	2-3	X = ADC(N) ; X = 0 … 3.3
0xE0	Аналоговый режим токового входа 0…20мА	1-2	X = ADC(N) ; X = 0 … 20

При обращении по абсолютному адресу линии микроконтроллера (например,0xA6, 0xC3 и т.д.) доступны режимы: 0, 1, 0x80, 0x81, 0x82, 0xC0.
Для аналоговых режимов, после конфигурирования применяется функция ADC. При этом в зависимости от выбранного режима меняется формат результата: для режима 0…20мА будет возвращено значение в диапазоне 0…20, для 0…+24В значения 0…24 и для 0…+3.3В значения 0…3.3.
Для режима PT1000 возвращается напряжение измеренное на полумосте PT1000 - 1К в вольтах(из расчета питания моста 3.3В) и при этом на четырех указанных линиях(см. описание MCX53-32) производится автоматическая коррекция результата, учитывающая ошибки вызванные падениями на ключах.
Для дискретного режима 0…+24В после инициализации в этот режим функция PIN возвращает 0 или 1, порог выбран в районе середины диапазона.

Добавлена поддержка ЖКИ TIC32 с драйвером PCF8531. Такая конфигурация предусмотрена в модуле MCX53-20.x. Обмен с ЖКИ производится по второму каналу I2C с использованием встроенного контроллера прямого доступа (ЦПУ не участвует в процессе передачи массива данных в ЖКИ). Благодаря этому регенерация экрана не отражается на производительности контроллера.
Поддерживается и вывод на ЖКИ прямо из интерпретатора (например оператором PRINT), и обработка команд приходящих по интерфейсам RS485 и I2C (полностью аналогично нашему модулю MCX53-19.x).
Обеспечивается вывод символов в символьном режиме(по знакоместам), в графическом режиме(в любой точке экрана).

Доступны символы двойной ширины / высоты (+).
Можно выводить точки, линии, прямоугольники, заранее сохраненные графические фрагменты.
Все это можно делать при полном наборе режимов вывода: перо белое / черное , забой, наложение, инверсия и т.д.
Для символьного вывода из PRINT нужно указать адрес #16. Для доступа у графическим функциям будут в следующей версии реализованы операторы, сейчас они доступны пока по командам поступающим по внешним интерфейсам.

Для модуля MCX53-20.x доступно управление RGB подсветкой и контрастностью ЖКИ.
Для основных цветов используется оператор LCD(140) = 0…14, для раздельного задания R, G, B - LCD(141…143) соответственно.
Для управления контрастностью применяется LCD(110) = …

Для модуля MCX53-20.x доступно управление звуком. Пока доступ через оператор LCD(150) = тон , LCD(151) = длительность. Тон задается из расчета длительности периода сигнала в микросекундах, длительность в миллисекундах.

Для модуля MCX53-20.x восемь кнопок доступны по функциям PIN(0 …7).

9.05.2011 v1.15

Добавлены оператор / функция **COUNT**. Теперь можно использовать встроенные аппаратные счетчики для счета импульсов от внешних источников и работы с энкодерами. Подсчет происходит аппаратно, независимо от работы BASIC- программы.
Доступно до трех(в зависимости от типа модуля) независимых 16 битных счетчиков. Для запуска счета необходимо вызвать оператор COUNT с параметрами инициализации. В параметрах указывается:
- номер аппаратного таймера (2, 3, 8) -> третья тетрада считая от младшей,
- режим счета(счет с/без обнулением при прочтении, энкодер, энкодер с обнулением) -> вторая тетрада от младшей,
- комбинация входа(ов)(для счетчика до 4 линий, для энкодера до 2 пар линий) -> младшая тетрада,
- начальное состояние счетчика (0…0xFFFF ⇔ 0…65535).

При инициализации выбранная(ые) линии автоматически инициализируются как входы с подтяжкой.
В режиме энкодера подсчет ведется при каждом изменении любого из сигналов, т.е. на период получается 4 .
Т.к. используются аппаратные таймеры, то для подачи входного сигнала доступны не все линии, а только конкретные комбинации для конкретного типа модуля. Доступные комбинации приведены в описаниях на модули.
Функция COUNT возвращает число накопленных импульсов(шагов для энкодера). Есть 2 группы режимов с обнулением и без обнуления при прочтении.
Оператор COUNT заносит в счетчик требуемую величину.

Примеры:

```
count(213h)=7 ;инициализация TIM 2 в счетчик без обн. на 3 вход с предустановкой 7
count(821h)=5 ;инициализация TIM 8 в счетчик с обн. на 1 вход с предустановкой 5
count(831h)=4 ;инициализация TIM 8 в энкодер без обн. на 1 комб. с предустановкой 4
count(342h)=0 ;инициализация TIM 3 в энкодер с обн. на 2 комб. с предустановкой 0
count(3)=5678 ;занесение в TIM 3 значения 5678
x = count(8)  ;чтение TIM 8
```

Для функции **CMP** добавлена возможность сравнения данных и во Flash. При указании отрицательного адреса, он рассматривается как адрес байта во Flash памяти.

Добавлен оператор **WDOG**. Он управляет работой встроенного в микроконтроллер узла Independent watchdog. После сброса микроконтроллера работа этого узла запрещена. Пользователь, при необходимости, может разрешить его работу, указав период в секундах срабатывания watchdog. При этом в программе необходимо будет предусмотреть периодический сброс watchdog гарантировано чаще чем было указано при инициализации оператора. Если произойдет переполнение watchdog , то модуль будет сброшен и программа запустится сначала. Диапазон задаваемых времен 0.01 … 26с. По умолчанию принято значение =1с. Узел watchdog имеет собственный тактовый генератор. У этого генератора довольно большой разброс по опорной частоте – приблизительно до 1.5 раз в обе стороны, это надо учитывать при выборе значения оператора WDOG. Пример использования:

```
17 WDOG = 2.5     ;зададим критическое время =2.5с
…
1458 WDOG   ;сброс, должен выполняться заведомо чаще чем 2.5с
…
```

11.04.2011 v1.14

Значительно расширена функциональность директивы **INFO**. Теперь директива выводит:
- текущее значение даты – времени энергонезависимых часов реального времени;
- тип модуля;

- тип микроконтроллера;
- серийный номер контроллера;
- версию интерпретатора;
- версию Boot-loader;
- текущий адрес MODBUS для USB канала;
- текущий адрес MODBUS для RS485 канала;
- текущий slave - адрес I2C области управляющих регистров;
- текущий slave - адрес I2C области данных;
- количество строк BASIC – программы зарегистрированных директивой MTOP;
- размер буфера для сохранения переменных во Flash зарегистрированных директивой MTOP;
- максимальное количество переменных в BASIC;
- максимальное количество индексированных переменных(элементов массивов);
- максимальное количество символов в символьных массивах;
- количество строк в текущей BASIC – программе;
- температура кристалла в градусах;
- текущая разница опорных напряжений Vdd и Vint_ref (в качестве опорного напряжения для встроенного АЦП используется основное напряжение питания микроконтроллера 3.3В, оно задается отдельным внешним для микроконтроллера стабилизатором. Кроме этого в самом микроконтроллере имеется собственный источник опорного напряжения номиналом 1.2В. Величина отношения в процентах показывает текущую разницу между ними. Как правило, при нормальном питании, типичное значение этой разницы в пределах одного процента);
- адрес технической поддержки;
- сайт разработчика.

Добавлена возможность измерения температуры кристалла. В микроконтроллере имеется встроенный температурный датчик. Функция ADC(200) возвращает значение в вольтах, измеренное на этом датчике. Температуру в Цельсиях можно рассчитать по формуле T (in °C) = (1.43 - ADC (200)) / 0.0043 + 25.

Добавлена возможность измерения напряжения на встроенном источнике опорного напряжения. Функция ADC(201) возвращает значение в вольтах, измеренное на нем. В качестве опорного напряжения для встроенного АЦП используется основное напряжение питания микроконтроллера 3.3В, оно задается отдельным внешним для микроконтроллера стабилизатором. Встроенный источник имеет типичное значение = 1.2В. При необходимости, пользователь может использовать его измерение для дополнительной калибровки точности своих измерений.

Добавлена обработка удобной формы записи шестнадцатеричных чисел : 0xF1B5. Это довольно распространенная форма записи. В ней всегда первые 2 символа “0x” , а далее следует шестнадцатеричное число нужной длинны. При этом не нужно добавлять ноль перед числом начинающимся с буквы и символ “h” в конце. Теперь можно в программах применять обе формы – старый вариант с “h” на конце, или новый.

Добавлена возможность простого изменения параметров канала RS485. Для изменения надо применить оператор RS485F#400,(0) = . После знака равенства перечислить скорость(в бодах), битность(8), количество стоп бит(1 или 2), четность(0-отсутствует, 1-нечетность, 2-четность). При вызове необходимо перечислить обязательно все 4 параметра. Новые значения вступают в силу сразу и действуют до нового изменения или сброса питания. По включению питания всегда устанавливаются: 115200 бод, 8 бит, 1 стоп, без четности.
Так например, для варианта скорости 57600, 8-ми бит, одного стопового, без четности надо вызвать:

```
RS485F#400,(0) = 57600, 8, 1, 0
```

Исправлены проблемы с каналом RS485, операторами/функциями RS485w / RS485f, USB каналом в режиме MODBUS.

25.03.2011 v1.13

Обновлено приложение «Сообщения BASIC системы».

Удалены оператор / функция FLS. Новые варианты оператора / функции FLASH полностью перекрывают FLS.

Обновлено приложение «Параметры Fractal-BASIC-Cortex».

Исправлена ошибка при обработке функции PIN допущенная в версии 1-12. В 1-12 была оптимизирована работа с оператором / функцией PIN – обработка образующей их внутренней функции ускорена в 3-4 раза. Одновременно улучшены диаграммы при работе с оператором LAN.

Добавлена директива **PROTECT**. Теперь стало возможным защитить пользовательскую программу от считывания и копирования. В случае подачи директивы LIST после применения директивы PROTECT, будет выдано сообщение о

том что включена защита от считывания. Отключить защиту можно только директивой NEW, которая автоматически сотрет текст программы.
Добавление такой директивы стало возможным благодаря одному из важных свойств Fractal-BASIC-Cortex – его однокристальности. В прежних версиях - Fractal-BASIC и Fractal-BASIC-PIC эту директиву не имело смысла вводить из за того что BASIC - программа хранилась во внешнем чипе памяти. И при очень большом желании можно было просто считать программу оттуда. Теперь злоумышленники останутся наедине со своим желанием. Все что можно сделать с микроконтроллером в котором расположены и интерпретатор Fractal-BASIC-Cortex и защищенная BASIC программа - это стереть либо BASIC – программу и записать новую, либо при использовании доступа через средства низкоуровневого программирования и отладки - стереть кристалл полностью, вместе с интерпретатором Fractal-BASIC-Cortex и Fractal-Boot-Loader.

Добавлен оператор **LANI**. Он является полным аналогом оператора LAN. Разница в том, что этот вариант оператора работает с Lan-устройствами не непосредственно через линии микроконтроллера, а через специальный чип DS2482 , подключаемый по каналу I2C. Этот чип разработан и изготавливается «авторами» MicroLan 1-wire технологии – фирмой DALLAS(MAXIM). Он наилучшим образом адаптирован к работе с линиями MicroLan, обеспечивая формирование нужных диаграмм в «тонкостях». Использование этого чипа в качестве дополнительного моста для подключения MicroLan устройств позволяет создавать более протяженные и защищенные линии по сравнению с подключением приборов непосредственно к линиям ввода/вывода основного микроконтроллера.
В перечне выпускаемых нами модулей есть модуль IO4-1.1 с одним из таких чипов DS2482-800. Он обеспечивает взаимодействие с 8-ю лучами MicroLan, дополнительно обеспечивая гальваническую развязку линий от системной I2C шины. Интерпретатор выполняя оператор самостоятельно ведет обмен с DS2482, скрывая всю рутину множества необходимых действий. Новый оператор является очень простым, удобным и мощным средством взаимодействия с MicroLan приборами.
Синтаксис оператора полностью повторяет синтаксис LAN. Разница при задании параметра **<Line >.** В нем указывается I2C адрес модуля и номер выбранной линии . **<Line >** может принимать значения от 0 до 255(0FFh).
В старшей тетраде указывается младшая тетрада slave-адреса I2C выбранного модуля(в DS2482 старшая тетрада I2C slave-адреса всегда равна 3), а в младшей тетраде указывается номер линии 0…7.

20.03.2011 v1.12

Добавлен оператор **LCD**. Он обеспечивает поддержку работы модулей с Fractal-BASIC-Cortex с символьными ЖКИ и OLED индикаторами совместимыми с HD44780 (подавляющее большинство символьных ЖКИ и OLED). Наличие этой возможности максимально упрощает конечное изделие и написание программы для него, еще больше увеличивая функциональность наших модулей. После применения LCD, вывод сообщений производится через обычный оператор PRINT#. В адресе PRINT# для вывода на LCD надо указывать 16. При этом функциональность PRINT# по отношению к другим каналам вывода (I2C и SPI) сохранена в полном объеме.
Вызов оператора инициализирует работу с ЖКИ по конкретным линям. Для подключения используется тетрадный режим обмена, что позволяет подключать индикаторы всего по 6 линиям : 4 данные и 2 управляющие. Пользователь сам выбирает удобные ему линии. Для этого доступны все линии присутствующие на разъемах модуля. При этом не имеет никакого значения последовательность или очередность линии, любая из 6 линий может быть назначена на любой вывод разъема. Это дает возможность пользователям максимально оптимизировать свой проект.
Для инициализации параметр оператора должен указываться 100 и после обязательно перечисляются все 6 линий.
Выбранные пользователем линии просто перечисляются в последовательности: RS, E, D4, D5, D6, D7.
Линия ЖКИ “R/W” не используется, т.к. в индикатор производится только запись, то на этом можно сэкономить. Вывод “R/W” на индикаторе надо замкнуть на землю. Адреса линий задаются как в операторе PIN. Таблицу соответствия см. описание соответствующего модуля.

```
Пример инициализации:          LCD(100) = 0A3h,0B7h,0A1h,0A5h,0C6,0A2h
еще вариант       LCD(100) = 1,0,6,8,2,15
```

Для управления позицией вывода – курсором, нужно использовать оператор CSR.
Для двух строчных ЖКИ как правило вторая строка начинается с 40 –го знакоместа.
Оператор LCD(16) может задавать вид курсора:

```
LCD(16) = 12      ;курсор выключить
LCD(16) = 14      ;курсор включить
LCD(16) = 15      ;курсор мигающий
```

!!!Рекомендации по подключению ЖКИ:
Программно, как уже упоминалось, можно использовать все доступные пользователю линии.
НО, поскольку микроконтроллер использует для своего питания напряжение 3.3В, его *ВСЕ* выходные каскады могут либо выдавать уровни соответствующие 0 и 1(0 и 3.3В), либо *ЧАСТЬ* линий толерантных к +5В могут работать по схеме с открытым стоком, что в паре с внешним подтягивающим резистором дает размах +5В для единичного состояния. Следовательно, выбор конкретного типа ЖКИ может накладывать ограничения на часть линий модуля.
Самый простой вариант(но не самый распространенный) – ЖКИ для которых потенциал 3.3В уже гарантировано является логической единицей(см.спецификации ЖКИ). Как правило, такие ЖКИ сами могут питаться от 3.3В.

Такие индикаторы можно подключать к любым доступным линиям микроконтроллера. При этом варианте нужно указать интерпретатору, что состояние логической единицы будет кодироваться активным состоянием выхода – логической единицей(3.3В). Это делается вызовом LED(106) = 1. Для этого случая внешние резисторы не нужны.
Второй вариант это ЖКИ у которых входной уровень логической единицы больше 3.3В. Для подключения таких индикаторов можно использовать только линии толерантные к +5В. Для MCU32-1.x это все 16 линий попадающие на кроссовый разъем. Эти линии могут работать по схеме с «открытым стоком». Для получения на них размаха логического сигнала близкого к +5В нужно выбранные линии «подтянуть» внешними резисторами 1.5К…10К к +5В. При использовании MCU32-1.x совместно с некоторыми модификациями кросс - модулей CRS10-1.x, линии подаваемые на ЖКИ можно не подтягивать, т.к. подтяжки уже стоят на модулях CRS10-1.x(см. соотв. схемы).
По умолчанию, после сброса для оператора LCD выбран режим LED(106) = 3, при котором состояние логической единицы кодируется состоянием выхода «вход», что эквивалентно обрыву и следовательно при наличии подтягивающего резистора там будет +5В.

Добавлены операторы / функции **MEMB#** и **REGB#**. Они введены вместо прежних своих аналогов. Назначения новых версий полностью совпадают с прежними. Главное и единственное отличие в том, что теперь номер бита “n” можно задавать константой, переменной или выражением. Раньше номер бита был «телом» оператора / функции, т.е. фактически было «n» копий. Операторы / функции работают с 16-битными словами.

Добавлена функция **CHSMR[%] <addr>,<len>** возвращает контрольную сумму или контрольный циклический код области RAM с начальным адресом <addr> из <len> байт. При наличии модификатора [%] вычисляется контрольный циклический код (см. оператор LAN).

11.03.2011 v1.11

Добавлена функция **ROT#**. Функция введена вместо прежней функции ROT. Назначение новой версии функции полностью совпадает со старой. Главное и единственное отличие в том, что теперь число сдвигов “n” можно задавать константой, переменной или выражением. Раньше число сдвигов было «телом» функции, т.е. фактически было n функций. Функция работает с 16-битными словами. Синтаксис: `ROT#<n>,( X )`

Добавлена функция **TSTFL**. Она предназначена для быстрой проверки доступности для записи нового значения переменной по конкретному адресу массива Flash данных пользователя. Другими словами она проверяет на «стертость» комплекта из четырех байт по указанному адресу. В случае если ячейки «чистые» возвращается 1, если нет, то 0. Небольшая сложность в определении этого факта без этой функции состоит в том, что функция FLASH возвращает переменную в формате «плавающая». И значение чистой переменной, когда все 4 байта равны 0xFF, для формата **IEEE754** не соответствует никакому числу.

Добавлен модификатор “W” в оператор/функцию FLASH. Это сделано для удобства работы с бинарными величинами (в смысле не плавающими - > целыми-16 бит). Параметром является адрес ячейки. Адрес указывается с точностью до байта, т.е. он соотносится с адресацией оператора/функции FLASH без модификатора как 4:1. Адресация сделана байтовой, чтобы для бинарных данных она совпала с адресацией используемой при доступе через MODBUS(там идет тоже адресация байта). Обращение допускается только по четным адресам, т.к. из за особенностей применяемых чипов , обеспечивается доступ только к 16-битным словам.
FLASH и FLASHW оперируют одним и тем же массивом Flash памяти и начинаются с одной и той же ячейки 0.

Теперь и обмен по каналу RS485 тоже сопровождается светодиодной индикацией. Любая передача из модуля в каналы USB или RS485 сопровождается красным светодиодом, а прием – желтым.

Добавлена возможность обмена по протоколу MODBUS в USB - терминальном канале модуля. Это дает широкие возможности по защищенному и быстрому обмену через USB с внутренними ресурсами модуля и его локальными шинами I2C и SPI. Появилась возможность работы через PIC18Terminal по каналу USB и следовательно теперь можно осуществлять загрузку драйверов во внешние модули не прибегая к услугам канала RS485.
Терминальный канал теперь может находиться в двух устойчивых состояниях: посимвольном для работы с BASIC-Terminal и пакетном для работы PIC18Terminal (MODBUS). Переключение между каналами осуществляется записью в ячейку EEPROM(202h) значения 0 для побайтного режима или значения 1…255 для пакетного режима. Это значение одновременно является адресом MODBUS для канала USB. Записанные изменения вступят в силу только после сброса питания. Новые модули поставляются с предустановленным побайтным режимом.
!Обратите внимание: поскольку оператор / функция EEPROM оперирует с 16-битными словами(2 байта), то при адресации через MODBUS, где используется побайтовая адресация, нужно указывать вдвое больший адрес и при записи нового значения прописывать незначащий старший байт = 0.

```
Пример изменения режима из BASIC на пакетный, с присвоением адреса MODBUS = 0x12:
   EEPROM(202h) = 12h
Пример изменения режима из пакетного на побайтный:
   12 75 04 04 02 00 00 CRC16
```

7.03.2011 v1.10

Добавлен оператор **PUT$**. Он предназначен для передачи в канал RS485 пакетов данных «как есть», без CRC. Оператор имеет два параметра: первый – адрес ячейки с которой начинаются подготовленные к передаче данные, второй – число передаваемых байт. Применение оператора запрещает работу канала в режиме slave MODBUS до применения оператора / функции RS485.

```
Пример посылки сообщения длинной N расположенного начиная с адреса ADR:
   PUT$ ADR, N
```

Добавлена функция **GET$**. Она позволяет получить доступ к пришедшим по RS485 данным. Данные принимаются в пакетном режиме – ожидается слитный пакет, принимается целиком и передается в RAM начиная с указанного адреса. После приема пакета(вызова GET$), функция готова к приему следующего пакета.
Параметры функции : первый - адрес ячейки с которой будут размещены данные, второй – число байт которое нужно разместить. GET$ возвращает число равное реально пришедшему числу байт.
!Обратите внимание, если заданное ожидаемое число байт меньше реально пришедшего количества, то часть данных будет утеряна, скопируется только запрошенное количество.
Для правильной работы GET$ необходимо переключиться в режим непротокольного обмена, для этого достаточно хотя бы раз использовать PUT$ или GET$.
Применение функции запрещает работу канала в режиме slave MODBUS до применения оператора / функции RS485.

```
Пример запроса о наличии сообщения длинной N, которое будет расположено с адреса ADR,
количество байт в пришедшем пакете буде помещено в X:
   X = GET$ ADR, N
```

Добавлены оператор / функция **RS485**. С их введением работа с удаленными узлами сети RS485 по протоколу MODBUS стала очень простой и удобной. Оператор/функция RS485 обеспечивают соответственно запись и чтение памяти удаленных узлов. Обеспечивается протокольный обмен с подсчетом CRC16 и проверкой всех возникающих ошибок при обмене. При всей своей внешней простоте они осуществляют все процедуры в соответствии с протоколом MODBUS. Вся «протокольная рутина» спрятана от пользователя и на поверхности только результат.
Формат самоочевиден и копирует формат принятый в I2C. Первый параметр это адрес узла, второй – адрес байта в памяти удаленного узла. По аналогии с I2C доступны модификаторы “W” и “F”, что добавляет гибкости.

```
Пример записи слова из 2 байт в ячейку 0 узла MODBUS с адресом 2 с последующим
прочтением и печатью одного старшего байта:
   RS485W#2,(0) = 1234h
   PHB RS485(1)
```

Добавлена функция **RS485A**. Она возвращает диагностику последнего обмена оператора / функции RS485.
Если обмен прошел штатно (не было ошибок, совпал CRC16 и все параметры), будет возвращен 0.
В противном случае будет возвращен код ошибки.
Функцию необходимо использовать там где нужна уверенность, что slave устройство ответило, и ответило правильно.

Приведена в соответствие адресация команд MODBUS 0x76 и 0x77(чтение/запись Flash) с адресацией оператора/функции FLASH. Теперь начала областей совпадают. Адресация начинается с «0».
!Обратите внимание, что адресация через канал MODBUS – байтовая, а в операторе/функции FLASH адресуется переменная (состоящая из 4-х байт).
Поэтому адрес MODBUS соотносится к адресу в операторе/функции FLASH как 1:4.
В случае, если пользователь не использовал ни разу процедуру MTOP, или ее значение не оставляет места по буфер данных пользователя, ответная квитанция на команды MODBUS 0x76 и 0x77 придет с кодом ошибки 7, что означает обращение к закрытому или несуществующему ресурсу.

Введена возможность обращения к Flash-памяти данных пользователя через команду MODBUS 0x76 к адресам выходящим за пределы двух байтовой адресации (65536 байт). Введен страничный регистр в который заносится номер требуемой страницы по 65536 байт. После сброса он устанавливается в 0, обеспечивая доступ к начальным 65536 байтам. Занести новое значение в него можно обратившись с этой же командой 0x76, указав число байт к чтению = 0. При этом младший байт адреса указанного в этой команде будет занесен в адресный регистр. Таким образом вызванная команда вернет в ответной квитанции 0 байт, т.е. свою вызывающую копию.

Добавлены команды MODBUS 0x72 ->«чтение REG», 0x73 -> «запись REG», 0x7C -> .
Команды работают аналогично командам «чтение/запись RAM», только с адресным пространством регистров специального назначения. Принятая адресация такая же, как и в операторе/функции REG:
- диапазон 0… 7FFFh соответствует адресам Cortex-M3 0x4000 0000…0x4000 7FFF;
- диапазон 8000h…0FFFFh соответствует адресам Cortex-M3 0x4001 0000…0x4001 7FFF.
В эти области попадают основные регистры внутренних узлов.

Добавлена команда MODBUS 0x7C -> «обмен по SPI». Поскольку шина SPI полнодуплексная, обмен ведется одновременно в обе стороны. В случае если требуется только чтение, в теле запроса команды все равно посылаются любые данные в количестве равном запрашиваемому. В случае только записи, вернется такое же количество прочтенных данных, которые можно просто игнорировать.
Если значение параметра CS не больше 7, он передается на соответствующие сопровождающие обмен адресные линии. При этом линии автоматически инициализируются на выход.
Если не требуется сопровождение обмена адресными линиями и они должны оставаться в исходном состоянии, нужно задавать CS равным 0xFF.

Добавлены варианты значений для оператора PIN.
Присваивание значения 2 переводит линию в состояние аналогового входа, значения 3 в состояние входа с подтяжкой.

Добавлен раздел «общие замечания».

Поправлена функция ROT. Теперь число сдвигов до 15.

13.02.2011 v1.09
Исправлены оператор / функция CHSM.

Для оператора / функции I2C добавлен модификатор “F” для работы с переменными(переменная занимает 4 байта). Это значительно упрощает работу по каналу I2C, т.к. теперь нет необходимости передавать значения переменных побайтно и потом «склеивать» байты в переменную. Объектом обмена является именно переменная.
Кроме этого в режиме slave-I2C обеспечивается целостность значения переменной. Пользователь может быть уверенным в том, что несмотря на распределенный во времени обмен, все 4 байта гарантировано из одного комплекта, как при записи , так и при чтении. Обязательным условием соблюдения целостности является размещение переменных по адресам кратным 4-м байтам, т.е. адрес переменной обязательно должен содержать в двух младших битах «0». Интерпретатор размещает свои переменные именно так, поэтому специально об этом заботится не нужно.

Добавлена директива INFO. При ее вызове в командной строке выдаются сведения о размере кольцевого буфера для хранения текста программы и количестве данных, которые пользователь может сохранить во Flash. Значения выдаются с учетом типа модуля и примененных до этого вызовов директивы MTOP.
Содержание этой директивы будет расширено в ближайших версиях.

7.02.2011 v1.08
Для режима I2C-slave теперь можно воспользоваться функциями I2C#256,(0) или I2C#258,(0) для считывания текущих значений соответствующих slave-адресов. Кроме этого при определении новых адресов теперь их значения сохраняются в EEPROM .

Внесены изменения в оператор / функцию PIN и аналогичные ему по способу указания номера линии.
Теперь, если при указании номера линии, значение параметра в шестнадцатеричном выражении начинается с буквы соответствующей выбранному порту микроконтроллера, адресуется именно выбранная линия соответствующего порта. Т.е. в этом случае все – как и раньше.
Если значение параметра указывающего на номер линии находится в пределах 0…0x1F (0…31), то адресуется линия модуля с учетом разводки на разъемы. Так, на модуле MCU32-1.x пользователю доступны 32 линии ввода вывода, попадающие на разъемы модуля группами по 8. Соответственно при такой адресации не нужно заботиться о том какие именно линии каких портов попадают на конкретный разъем, а просто адресовать их подряд.
Такой двойной подход позволяет максимально упростить пользователю работу с линиями, не теряя при этом гибкость для более тонкого использования всех возможностей конкретного модуля, конкретного микроконтроллера.

Введена директива MTOP. Она введена для разделения пользовательской Flash-памяти контроллера на две части: кольцевой буфер текста BASIC-программы и буфер для долговременного сохранения пользовательских данных. Введение MTOP позволяет не задумываться о расчете адреса начала расположения пользовательских данных во Flash и не беспокоиться о проблеме наложения данных на текст BASIC-программы и наоборот.
Директива вызывается в режиме командной строки. *Ее применение вызывает стирание BASIC- программы* и размещение вновь заносимой программы с начала кольцевого буфера. Параметром директивы является число равное максимальному числу строк BASIC-программы помещающемуся в буфере. Причем речь идет не о максимальном номере строки, а именно о количестве строк.
Директиву необходимо применять только в случае если пользователь собирается пользоваться сохранением данных во Flash, в остальных случаях она не имеет смысла, и даже вредна, в том смысле , что уменьшает размер кольцевого буфера BASIC-программы и следовательно уменьшает ресурс циклов перепрограммирования этого участка.
Параметр устанавливаемый директивой(число строк) запоминается в энергонезависимой памяти, поэтому нет

необходимости лишний раз ее вызывать. Общая рекомендация – пользоваться ей не чрезмерно часто. Так, не рекомендуется на этапе отладки вносить ее в текст загружаемой программы по аналогии с NEW (тем более это не имеет смысла, т.к. однажды указанный размер буфера запоминается и нет необходимости его повторять).
Общая рекомендация: на этапе отладки всегда указывайте число строк кратно превышающее размер Вашей программы(чем больше тем медленнее будет расходоваться ресурс перепрограммирования Flash-памяти), а после того как программа отлажена – примените директиву с реально получившимся размером программы и при этом будет максимально возможный размер буфера для сохранения данных.
После применения директивы интерпретатор возвращает число равное максимальному числу переменных которое можно сохранить в пользовательском буфере данных. Эта величина зависит от типа микроконтроллера(размера Flash) и объема зарезервированного буфера BASIC-программы. Причем обратите внимание, возвращаемое число это именно количество переменных, а не байт(одна переменная занимает 4 байта).
В новом модуле изначальное значение MTOP сохранено максимально возможным для этого типа микроконтроллера, это соответствует максимально возможному размеру кольцевого буфера под текст программы и отсутствию буфера под долговременно сохраняемые пользовательские данные.
При необходимости вернуть уже инициализированную директиву в состояние эквивалентное новому модулю (вся память используется под текст программы, место под данные не отведено), нужно применить MTOP = 0 .
Обратите внимание – работа с оператором/функцией EEPROM не имеет отношения к этим областям Flash-памяти и следовательно значение MTOP не оказывает никакого влияния на их работу.
Пример установки MTOP на буфер размером в 500 строк: `MTOP = 500`

Добавлены оператор / функция FLASH. Они предназначены для работы с Flash –памятью данных пользователя. В сравнении с уже имеющимися оператором/функцией FLS, они значительно упрощают работу с данными сохраняемыми в пользовательской области Flash-памяти данных. Теперь единицей при работе с Flash является не байт, а переменная(она состоит из 4 байт). Адресация тоже упрощена – элементом является переменная, нумерация начинается с 0-го элемента буфера и т.д. до конца буфера. Максимальное количество переменных которые пользователь может сохранить в буфере выдает директива MTOP, при разделении Flash памяти на части – память текста BASIC-программы и память долговременно сохраняемых пользовательских данных. Новый модуль содержит указатель в значении которое предоставляет весь объем тексту программы и нет буфера данных. Пред первым использованием FLASH нужно обязательно воспользоваться директивой MTOP исходя из объема программы и ее статуса(отладка или законченный проект).
Формат оператора FLASH[(<Address>)]=<expr0>,<expr1>...<exprn> где Address - индекс в массиве сохраняемых переменных (4-хбайтовых float значений), который образуется в пользовательской части FLASH выше области отведенной под BASIC-программу. Одна единица индекса – одна переменная(4 байта Flash). При отсутствии Address в операторе запись производится в ячейку, следующую за последней записанной. Следует помнить, что запись возможна только в предварительно стертую ячейку, иначе будет выдано сообщение об ошибке. Стереть ячейку можно оператором ERASE.
Формат функции FLASH[(<Address>)] . Возвращает значение сохраненной переменной (4-х байтовое float-значение) из ячейки с индексом Address. Если индекс в функции не указан, то читается ячейка с индексом, следующим за последним прочитанным. То есть указатели чтения и записи независимы.

Добавлен оператор ERASE. Формат: ERASE(<Address>) . Стирает ячейку с индексом Address. Следует учесть, что поскольку блоки FLASH стираются по 2 Кбайта, то будут стерты все ячейки, попавшие в данный блок – всего 512 4-хбайтовых float-ячеек.

Мультимастер I2C работает в штатном режиме.

Добавлена встроенная самодиагностика.

24.01.2011 v1.07

Появилась возможность менять скорость обмена по I2C. Эта возможность позволяет подключать медленные устройства со скоростью обмена 100 КГц. Причем это можно делать динамически, что позволяет использовать на одной шине одновременно устройства со скоростью 400 КГц и 100 КГц.
Для установления скорости 100 КГц нужно вызвать оператор I2C#400,(0) = 0
Для установления скорости 400 КГц нужно вызвать оператор I2C#400,(0) = 1
После сброса устанавливается максимальная скорость обмена – 400 КГц.

В тестовом режиме введен режим мультимастер I2C.

Введен режим I2C- slave. Теперь пользовательская область RAM вся доступна через I2C, что позволяет получить полный и прозрачный доступ к данным для других модулей. Так к примеру, чтобы прочитать первую объявленную переменную бейсика, надо прочитать первые 4 ячейки по I2C.
Доступ к RAM осуществляется аналогично принятому в комплекте MCU4:

- первый байт содержит slave-адрес и признак записи / чтения;
- второй байт при записи расценивается как адрес байта в RAM(адрес слова – word – адрес);
- чтение производится применением совмещенного цикла запись-чтение, где сначала указывается адрес.

Поскольку word-адрес имеет размер всего один байт, для доступа к большому объему RAM сделана возможность записи смещения к этому адресу путем записи величины смещения в специальный страничный регистр(указатель). Причем используется два разных указателя – отдельно для записи и отдельно для чтения. Это позволяет при необходимости «развести» буфера чтения и записи в разные области.
После сброса оба указателя равны нулю, что означает - выбранные страницы записи и чтения совпадают и расположены в самом начале пользовательской области.
Word-адрес указателя для буфера записи = 0.
Word-адрес указателя для буфера чтения = 4.
Для доступа к управляющим регистрам(указателям) используется отдельный slave-адрес. Другими словами модуль отвечает на два разных slave - адреса.
Заводские установки – для доступа к данным slave - адрес 0x02, для доступа к управляющим регистрам 0x04.
Пользователь может изменить эти адреса (адреса могут быть только четные!) :
Для установления адреса доступа к данным нужно вызвать оператор I2C#256,(0) = NewAdr
Для установления адреса доступа к регистрам нужно вызвать оператор I2C#258,(0) = NewAdr

19.01.2011 v1.06

TIME не тормозится при выполнении DELAY.

Исправлены ошибки в RLDT.

Добавлен раздел «Работа с RS485 / MODBUS».
Добавлены команды MODBUS.

Упорядочено расположение массива переменных, оно приведено к началу пользовательской памяти (см.прил.3).

Исправлены ошибки при работе мастера I2C.

Введена директива REBOOT. Используется для перехода в режим ***Boot-loader-Fractal версии не ниже 1-07*** для обновления интерпретатора. Позволяет делать обновление без сброса питания и изменения состояния джамперов.

7.01.2011 v1.05

Приведены в соответствие форматы оператора и функции RLDT : в операторе теперь последовательность параметров – день, месяц, год-2000, часы, минуты, секунды. Отредактировано описание RLDT.

Работает LAN. Разница с Fractal-BASIC-PIC в параметре определяющим линию ввода-вывода. Формат параметра полностью соответствует применяемому теперь в PIN.

Уточнение работы с функциями/операторами FLS :
В связи со специфическими аппаратными особенностями допускается работа только с 16 – битными словами и по четным адресам. Адресация остается побайтовая. Кроме этого не допускается повторная запись уже записанной ячейки до ее стирания. В противном случае интерпретатор выдаст ошибку. Стирание производится блоками по 2К.
В ближайших версиях FLS будет модифицирован под сохранение переменных интерпретатора (они занимают 4 байта). Процедура сохранения переменных будет максимально упрощена. Такие же модификации ожидают и EEPROM.
Удалены оператор/функция FLSB.

Оператор DELAY теперь прерывается событиями Ctrl-C, ONINT1, ONTIME.

23.12.2010 v1.04

Введены функция/оператор EEPROM. Они предназначены для долговременного сохранения ограниченного количества данных пользователя и конфигурации системы. Поскольку в применяемых кристаллах STM32F103 EEPROM отсутствует, его работа сэмулирована на базе основной Flash.
Прямое сохранение пользовательских данных во Flash имеет два недостатка:
- необходимость стирания блока перед записью, если записываемая ячейка была использована до этого(не 0xFF);
- ограниченный ресурс перезаписей Flash (~10 000).

Оператор/функция EEPROM, как и внутренний алгоритм сохранения текста BASIC – программы лишены этих недостатков. Хотя, конечно, ресурс все равно не бесконечен. Реально ресурс повышается примерно на порядок при использовании средней загруженности занимаемых объемов. Так, если Вы используете обновление ячейки с одним и тем же адресом N раз, это эквивалентно использованию N ячеек один раз. Т.е. речь идет не о ресурсе каждой ячейки, а общем количестве перезаписей.

Пользователю предоставлено 256 слов по 16 бит, кроме этого есть системные ячейки для хранения параметров системы.
Пользовательские адреса 0…255. Кроме пользовательской области , существует область системных установок. В этой области допустимо использовать только приведенные в описании адреса.

```
Пример записи ячейки с адресом 0  :       eeprom(0) = 55AAh
Пример чтения ячейки с адресом 5  :       X = eeprom(5)
```

Адрес MODBUS теперь доступен для редактирования пользователем. Он хранится в ячейке EEPROM(200h). Производственная установка = 2. Вновь занесенное значение вступает в силу после сброса питания.

```
Пример нанесения нового значения адреса MODBUS = 8 : eeprom(200h) = 8
```

17.12.2010 v1.03

Введен оператор DELAY. Он предназначен для простой организации временнЫх задержек в программе. Ему могут присваиваться значения от 0.001с до 65.535с.

```
Пример задержки на 3.5с  :          delay = 3.5
```

Для оператора DAC добавлена возможность выбора формата данных.
Возможны 3 варианта:

- формат 0 (по умолчанию) -> величина в вольтах 0…3,3 В;
- формат 1 -> величина соответствующая коду записываемому в ЦАП 0…4095;
- формат 2 -> величина в долях от целого 0…1.

Для выбора формата необходимо обратиться к несуществующим каналам 100, 101, 102 соответственно.
Формат один для всех каналов ЦАП.

```
Пример установки формата «доля от целого»  :   dac(102) = 0
```

Для функции ADC добавлен оператор ADC для возможности выбора формата данных.
Возможны 3 варианта:

- формат 0 (по умолчанию) -> величина в вольтах 0…3,3 В;
- формат 1 -> величина соответствующая коду АЦП 0…4095;
- формат 2 -> величина в долях от целого 0…1.

Для выбора формата необходимо обратиться к несуществующим каналам 100, 101, 102 соответственно.
Формат один для всех каналов АЦП.

```
Пример установки формата «код АЦП»  :     X = adc(101)
```

16.12.2010 v1.02

Введена функция ADC. Функция предназначена для получения измерения выбранного канала АЦП.
Параметром функции является номер аналогового канала (0…15).
Линии имеющие функции аналогового входа перечислены в описании соответствующего модуля.
При вызове функции происходит автоматическая инициализация соответствующей линии в режим аналогового входа.
Функция возвращает измеренное напряжение в вольтах.
При первом обращении к каналу, который до этого момента не был аналоговым, возможен возврат неверного значения. Поэтому, в тех случаях где это важно, нужно одно холостое первое обращение.
!обратите внимание диапазон измеряемых напряжений 0…3,3В. Недопустим выход входного напряжения за эти пределы.

```
Пример измерения канала ADC4 :      X = adc(4)
```

Введен оператор DAC. Оператор предназначен для выдачи аналогового сигнала на выбранный канал ЦАП.
Параметром является номер аналогового канала (1…2).
Линии имеющие функции аналогового выхода перечислены в описании соответствующего модуля.
При вызове оператора происходит автоматическая инициализация соответствующей линии в режим аналогового выхода. Оператор устанавливает значение ЦАП в пределах 0…4095 единиц (0…3,3В)

```
Пример выдачи на DAC2 3.3В/2:       dac(2)=2048
```

13.12.2010 v1.01

Оператор PRINT# теперь имеет возможность вывода в канал SPI. Если указанный после # адрес больше 0FFh, то оператор выводит поток в канал I2C , как и раньше, без каких либо отличий.
Если адрес 0…0FFh то вывод идет в SPI. При адресе 0…7 он перед началом обмена выставляется на адресные линии SPI. и держится весь обмен до завершения оператора, после чего на линии адреса выдается состояние 7, что соответствует всем единицам.
Если адрес 0FFh, то обмен будет производиться без изменения состояния линий адреса.

Работает RLDT.

7.12.2010 v1.00 Создание документа.

Первая версия **Fractal-BASIC-Cortex** .

*Отличия от **Fractal-BASIC-PIC v1.47** :*

- Увеличено быстродействие порядка 4- 5 раз.
- Оператор / функция REG имеют в качестве параметра адрес в диапазоне 0…0FFFFh.

При этом диапазон 0… 7FFFh соответствует адресам Cortex-M3 0x4000 0000…0x4000 7FFF,
диапазон 8000h…0FFFFh соответствует адресам Cortex-M3 0x4001 0000…0x4001 7FFF.
В эти области попадают основные регистры внутренних узлов.

- Оператор / функция MEM имеет в качестве параметра адрес в диапазоне 0…0FFFFh.

Этот адрес не совпадает с абсолютными адресами Cortex-M3. Он покрывает область данных интерпретатора.

- Оператор / функция PIN - теперь параметр не номер контакта разъема, а номер порта-линии микроконтроллера. Это расширяет возможности, позволяя использовать все доступные выводы. Параметр имеет очевидный формат – например линия порта A №4 теперь может быть адресована как 0A4h или она же 04h, D №15 -> 0DFh или 3Fh.
- Оператор SPI теперь позволяет легко изменить номер режима по полярности и фазе тактового сигнала. Режим по умолчанию №2. Для смены режима нужно обратиться к несуществующим устройствам с адресами 100…103 соответственно.

```
Пример  - установить режим 1 :      spi#101 = 0
```

- Отсутствует операторы IOU и IOL.