BASIC
Для BM93**
Серия PICmicro
2009

Содержание

ПРИМЕНЯЕМЫЕ СИМВОЛЫ

При записи программ на языке BASIC применяются:
– десятичные цифры;
– строчные и заглавные буквы латинского алфавита;
– строчные и заглавные буквы русского алфавита (русские буквы допускается использовать 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 I2C);
"@" – 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).
Рекомендация. Для увеличения скорости работы программы следует употреблять имена как можно короче.
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 : ± 2-126 / ± 2128. При арифметических вычислениях и вычислении значений стандартных функций используется библиотека math MPLAB C30 фирмы Microchip. В этой библиотеке все арифметические ошибки помечаются признаком NaN (00h,00h,80h,7Fh). Поэтому выдается только одно сообщение об ошибке: "Error: math. Error" при переполнении, потере значимости и делении на ноль.
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.
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". Шестнадцатеричные числа 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 <var1>(n11,n12...n1N), <var2>(n21,n22...n2M),...<varn>(nn1,nn2...nnZ) резервирует место в FRAM для массивов индексированных переменных <var1>,<var2>,...<varn>. 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».
Пример:

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

DIM$

Оператор DIM $(n),m резервирует место в FRAM 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 <var1>,<var2> ... <varn> объявляет переменные и резервирует для них место в FRAM, но не инициализирует оные переменные. Оператор позволяет использовать сохраненные значения переменных после перезапуска программы после программного останова или сброса питания. Оператор STATIC должен располагаться в начале программы, до объявления прочих переменных. Для определения первого запуска программы следует пользоваться флагом оператора NEW FLAG_NEW, расположенного в FRAM по адресу 0x000A. При подаче директивы NEW этот флаг сбрасывается, установка флага – задача пользователя.
Пример:

```
10 static A,B,C,D ;объявление переменных
20 if MEM(0AH)<>0 then goto 100 ;первый запуск программы?
30 mem(0AH)=1: A=2: B=3: C=4: D=5 ;да-> инициализация и установка флага
100 A=A+B: B=B+C: C=C+D: D=D+A ;некий рабочий цикл
110 print A,B,C,D
```

ОПЕРАТОРЫ ОБЩЕГО НАЗНАЧЕНИЯ

ABS

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

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

AND

Оператор <iexpr1>.AND.<iexpr2> 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*

DATA

Оператор DATA <expr1>,<expr2>,...<exprn> 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
;Выводим на терминал
40 print v1,v2,v3,v4,v5
;Для повторного применения оператора READ
;необходимо использовать оператор RESTORE
50 restore
60 read x1,x2,x3,x4,x5
;Выводим на терминал
70 print x1,x2,x3,x4,x5
```

DO

Оператор DO организует циклические вычисления с выходом из цикла при выполнении определенного условия. Формат: *DO: <opr1>:<opr2>:... <oprn>: WHILE | UNTIL <cond>*
Пример:

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

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>=<expr1> TO <expr2> STEP <expr3> организует цикл. где:
<var> – имя переменной цикла;
<expr1> – начальное значение <var> – возможно применение HEX значений;
<expr2> – конечное значение <var> – возможно применение HEX значений;
<expr3> – 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 <opr1|ln1> [ELSE <opr2|ln2>] 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 <opr1> <:> <opr2> ...< :> <oprn> ;
– IF<cond> THEN <opr1> ELSE <opr2>< :> <opr3> ...< :> <oprn>.
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>",] <var1>,<var2>,...<varn> выводит приглашение (<text>) к вводу числовых или текстовых значений переменных <var1>...<varn> 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>|$(i)) возвращает адрес, по которому во FRAM располагается числовая или строчная переменная <var>|$(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 $(5)
40 print@ loc($(5),3)," " ;заменим 3-й символ на пробел
50 print $(5)
```

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 <ln0>,<ln1>,...<lnn> 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 <ln0>, если <expr>=1, то <ln1> 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 арифмети-ческой 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 Int1.
Пример:

```
10 onint1 100
20 print "WORK CIKLE"
30 time=0: do: while time<2
40 goto 20
;Подпрограмма обработки прерывания INT1
100 print "INTERRUPT INT1"
110 reti
```

OR

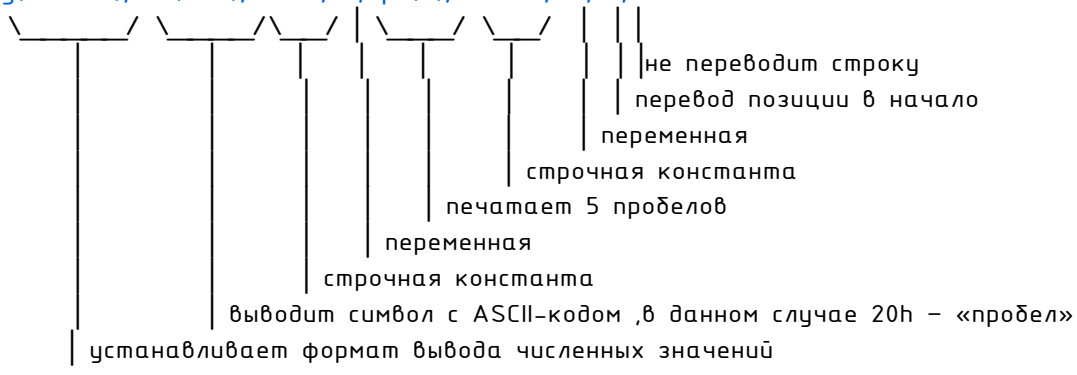
Оператор <iexpr1>.OR.<iexpr2> производит поразрядное логическое "ИЛИ" со словами или байтами.
Пример:

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

PHB

Оператор PHB [<s1>,<s2>,...<sn>] 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,
```



PHW

Оператор PHW [<s1>,<s2>,...<sn>] 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 <varn>,<varn-1>,...<var1> 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
```

PUSH

Оператор PUSH <expr1>,<expr2>,...<exprn> 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 <var1>,<var2>,...<varn> 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>. Число <n> может принимать значения от 1 до 8. Производится сдвиг 16-битного слова. Если <iexpr> больше 0xFFFF, то выдается сообщение об ошибке.
Пример:

```
10 B=rot8(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
```

SPC

Оператор SPC(<iexpr>) применяется в операторах печати для вывода пробелов в количестве <expr>.
Пример: *см. PHB*

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.

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 десятичнойдроби 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. Общее количество символов "#" не должно быть больше 8. Установленный формат

вывода действует до последующего применения оператора USING (в пределах оператора PRINT). Вместо ключевого слова "USING" можно употреблять сокращение – "U.".

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
```

VAL

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

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
```

ОПЕРАТОР ДЛЯ РАБОТЫ В ЛОКАЛЬНОЙ СЕТИ RS485

PUT$

Оператор PUT$ [<addr>,<len>,<ln>] посылает в последовательный канал RS485 <len> байтов из FRAM (начиная с адреса <addr>) по запросам прерывания от передатчика. Оператор работает с протоколом MODBUS в режиме "мастер". В FRAM по адресу начиная с <addr> необходимо разместить мастер-команду (без контрольной суммы – BASIC-PIC подсчитает его сам). В параметре <len> указывается длина команды в байтах без учета контрольной суммы. После передачи команды BM9300 исполняет следующие операторы, после прихода ответной телеграммы – происходит прерывание и переход на строку с номером <ln>. Переход происходит и в случае отсутствия ответной телеграммы – после истечения кванта ожидания ответа – 10 мс. Ответная телеграмма располагается в ОЗУ PIC24F по адресам 0x1000-0x1FFF. По адресу 0x1100 располагается флаг приема телеграммы:
0 – телеграмма принята
1 – истек квант ожидания.
Доступ к этим ячейкам возможен через команды REG и REGB.
При использовании команды PUT$ желательно не использовать режим "подслушивания" по протоколу MODBUS – команда использует те же физические линии связи и те же буферы памяти. 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.
Пример:

```
;В приведенном примере устройство  BM9300 связано индикатором по
;последовательному каналу RS485.
10 dim $(1),30 ;размер буфера команд в индикаторе =>25байт+5заголовок
20 ADR=loc($(0))
30 mem(ADR)=2,71h,0,0 ;узел 2,запись RAM,адрес 0x0000
40 ADR_MES=ADR+5 ;адрес начала данных
50 ADR_N=ADR+4 ;адрес числа байт данных
;пауза
70 time=0: do: while time<1
;выведем текстовое сообщение
200 print@ADR_MES,"QWERTY 1234567"
210 mem(ADR_N)=14 ;число байт данных
220 put$ ADR,5+MEM(ADR_N),2000
;пауза
230 time=0: do: while time<1
;поменяем цвет на красный
300 mem(ADR_MES)=0Ch,1
310 mem(ADR_N)=2 ;число байт данных
320 put$ ADR,5+MEM(ADR_N),2000
500 end
;напечатаем квитанцию
2000 phb reg(1000h), reg(1001h), reg(1002h), reg(1003h), reg(1004h)
2100 reti
```

ОПЕРАТОРЫ ДЛЯ РАБОТЫ С ШИНОЙ I2C

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 I2C отсутствовало подтверждение или был превышен лимит времени ожидания, "0" свидетель-ствует о нормально произведенном обмене.
Оператор I2C[W][#<SlaveAddress>,](<WordAddress>)=<iexpr1>,<iexpr2>,... <iexprn> производит вывод в канал I2C. Значение Slave Address может быть только четное в пределах 0...254. <WordAddress> – адрес слова в адресном пространстве устройства I2C – может принимать значения 0...65535; если значение <addr> меньше 256, то в канал I2C передается только 1 байт; <expr1>,<expr2>,... <exprn> – данные для записи в канал I2C. При наличии модификатора [W] выражения <iexpr1>,<iexpr2>,... <iexprn> рассматриваются как слова из 2-х байтов и могут принимать значение 0...65535. Если модификатор отсутствует, то когда значение <expr> меньше 256, в канал I2C 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>)=<iexpr1>,<iexpr2>,... <iexprn>.
Функция I2C[W][#<SlaveAddress>,(<WordAddress>)] возвращает численное значение байта или слова, считанного 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>)= <iexpr1>, <iexpr2>, ...<iexprn> 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".
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
30 if A>100 then i2cb3(2)=1 else i2cb3(2)=0
;проверим подтверждение последнего обмена
50 if i2ca=0 then print"OK" else print"ERROR"
```

CSR

Оператор CSR=<iexpr> устанавливает исходное положение действующей позиции печати при выводе в канал I2C. Нулевое значение соответствует крайней левой позиции. Значение <expr> может быть в пределах от 0 до 128 и перед выводом в канал I2C суммируется с константой 80h.

PRINT#

Оператор PRINT#[<iexpr>] , <s1>,<s2>,...<sn> выводит информацию в канал I2C. Объекты вывода как у оператора PRINT. Значение <iexpr> определяет адрес выходного устройства и может находиться в пределах 0...65535, передается в канал I2C в виде одного или двух байтов. Если значение <expr> находится в пределах 0...254, то оно воспринимается как Slave Address в канале I2C. В случае, когда <expr> занимает 2 байта, первый из них рассматривается как Slave Address, а второй – как Word Address в пространстве адресов выходного устройства. При зависании канала I2C вывод прекращается. Факт зависания можно проверить, прочитав значение, возвращенное функцией I2CA (если канал завис, I2CA=1). Следом за адресными байтами 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 запомненным значениям Slave Address и Word Address.

PHB#

Оператор PHB# [<iexpr>] , [<s1>,<s2>,...<sn>] аналогичен PHB, но выводит в канал I2C.

PHW#

Оператор PHW# [<iexpr>] , [<s1>,<s2>,...<sn>] аналогичен PHW, но выводит в канал I2C.

ОПЕРАТОРЫ ДЛЯ РАБОТЫ С ШИНОЙ SPI

SPI

Оператор SPI [W] [#<CS>,] [(<WordAddress>)] = <iexpr1>,<iexpr2>,... <iexprn> производит вывод в канал SPI.
Значение <CS> может быть в пределах 0...7. При работе оператора значение <CS> выводится на контакты 2, 3, 4 кросса. После отработки оператора исходное состояние линий не восстанавливается. До момента первого объявления <CS> эти линии находятся в третьем состоянии и могут быть использованы для других функций. <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; <expr1>,<expr2>,... <exprn> – данные для записи в канал SPI. При наличии модификатора [W] выражения <iexpr1>,<iexpr2>,... <iexprn>, 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. После исполнения оператора значение <CS> остается в системной переменной и, если нет необходимости в его замене, при следующем использовании оператора может быть не указано. Может отсутствовать и <WordAddress>, в этом случае размещение передаваемых данных будет определяться логикой работы устройства SPI.
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. В случае, когда #<CS> отсутствует, его значение берется из системной переменной. 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:

```
;Пример использования одновременно двух модулей АЦП с SPI интерфейсом
;для работы с  BM9300
;Используются модули ADC3-1.x с установленными адресными джамперами
; в положения соответствующие линиям №2 и №3 кросса.
;Комбинация <CS> для этих линий получается соответственно <6> и <5>.
;Измерим 1-й канал
100 ad=6 : gosub 400
; отобразим в HEX
110 print " ADC1(CS->P1.0) -> ", : phb xh, xm, xl, : print "h ",
; приведем к вольтам
120 u= (xh*65536+xm*256+xl)/3276800-2.56: print u, "V"
;Повторим для канала 2
200 ad=5 : gosub 400
; отобразим в HEX
210 print " ADC2(CS->P1.1) -> ", : phb xh, xm, xl, : print "h ",
; приведем к вольтам
220 u= (xh*65536+xm*256+xl)/3276800-2.56: print u, "V" : print
240 goto 100
```

```
;Подпрограмма запуска измерения и получения результата АЦП с адресом <ad>
;Сбросим AD7718 посылкой 32 единичных бит подряд и сразу запустим:
;FF FF FF FF 01 43
;Фильтр включен,небуферированный AINCOM,опора AD780,5/10 входов,
; осциллятор не выключен,режим циклического преобразования
;02 07
;IN+ => AIN1, IN- => AINCOM; биполярный режим; диапазон +-2.56В
400 spi#ad=0FFh,0FFh,0FFh,0FFh,1,43h,2,7
; ждем завершения преобразования
410 time=0 : do : while time<0.25
; запросим результат
420 spi#ad,(44h)
; произведем чтение SPI
430 xh=spi: xm=spi: xl=spi: return
```

ОПЕРАТОРЫ ДЛЯ РАБОТЫ С СЕТЬЮ MICROLAN

LAN

Оператор LAN <s1>,<s2>,...,<sn> выполняет последовательность команд <si> обмена с объектами сети MicroLAN. Команды могут быть записаны в любом порядке и выполняются последовательно, слева-направо. В операторе могут присутствовать следующие команды:
Z(<bit>,<iexpr>) - инициализирует сеть MicroLAN на входном дискретном канале с битовым адресом <bit>, 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>;
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 E_RAM начиная с адреса <addr>.
R(<addr>,<len>) - принимает из сети <len> байт и располагает их в E_RAM начиная с адреса <addr>; при приеме информации из сети вычисляются коды циклического контроля (CRC8, CRC16); вычисленные коды остаются в системных переменных CRC8, CRC16.
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 0 до 255, причем 0 соответствует задержке 256 мС.
S(<addr>,<ln>) задает начальные условия и производит один цикл поиска устройства в сети. Номер найденного устройства будет размещен в 8 байтах E_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: X8+X5+X4+1.
Пример простого поиска (команда 0F0h) в предположении, что в сети не более 3-х устройств:

```
lan  Z(90h,50),T#0F0h,S(8000h,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 8008h, 8010h, 8018h соответственно;
выполнение следующей строки, если в сети более 3-х устройств (номера первых 3-х будут определены).

ОПЕРАТОРЫ ДЛЯ РАБОТЫ С ПОРТАМИ ВВОДА/ВЫВОДА И ВНУТРЕННЕЙ ПАМЯТЬЮ (I_RAM)

REG

Оператор REG[W](<addr>)=<iexpr> устанавливает значение байта или слова в области регистров специальных функций или нижней половины внутренней памяти(I_RAM) по адресу <addr>. Записываемое значение задается <expr>. При записи слова оператор REGW размещает младший байт по адресу <addr>, а старший – по адресу <addr+1> (формат фирмы Intel).
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>)= <iexpr1>, <iexpr2>, …<iexprn> производит установку или сброс битов в байте, адресуемом выражением <addr>. 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 REGB<n>(<addr>) возвращает значение бита в байте по адресу <addr>. Функция возвращает значения "0" или "1".

IOU

Оператор IOU выдает "отмыкающую" последовательность ассемблерных команд. После этого можно изменять назначение выводов процессора PIC24FJ64GA004. Для изменения назначения выводов следует пользоваться операторами REG и REGB. Адреса соответствующих регистров и их назначение приведены в описании процессора PIC24FJ64GA004.

IOL

Оператор IOL выдает "запирающую" последовательность ассемблерных команд. Между операторами IOU и IOL предпочтительно _____только изменять назначение выводов, а не относящиеся к этой процедуре операторы размещать до оператора IOU и после оператора IOL.
Пример:

```
;Чтение состояния порта RB12 и запись в порт RB10
;эти порты выведены на разъем X1: RB10 – 14 вывод, RB12 – 12 вывод
;если эти выводы соединить, то значение, записанное в RB10, будет
;считываться в RB12
;назначение имен регистров
10 RPINR=680H: RPOR5=6CAH: TRISB=2C8H: PORTB=2CAH
;проверяем, не назначен ли RB12 выводом специальных функций
20 FOR RPINR=680H TO 6AFH
;назначен? да -> переназначим на RB15
30 IF REG(RPINR)=12 THEN IOU: REG(RPINR)=15: IOL
40 NEXT RPINR
;RB10 – обычный цифровой вход (снимаем назначения специальных функций)
50 IOU: REG(RPOR5)=0: IOL
;RB10 – вход, RB12 – выход
```

```
60 REGB2(TRISB+1)=0: REGB4(TRISB+1)=1: RB10=0
;выбор значения RB10
70 INPUT "RB10=",RB10
;ввод выбранного значения
80 REGB2(PORTB+1)=RB10
;чтение RB12
90 P. "RB12=",REGB4(PORTB+1)
100 GOTO 70
110 END
```

ОПЕРАТОРЫ И ФУНКЦИИ ДЛЯ РАБОТЫ С ВНЕШНЕЙ ПАМЯТЬЮ (FRAM)

PRINT@

Оператор PRINT@<addr>,<s1>,<s2>,...<sn> производит "печать" в память. Выводимые коды располагаются начиная с адреса <addr>.

MOVE

Оператор MOVE <addr1>,<addr2>,<len> копирует <len> байтов начиная с адреса <addr1> в область памяти с начальным адресом <addr2>. Оператор может быть применен для выделения подстроки из строчной переменной,
например:
`move loc($(0))+3,loc($(1)),8 : mem(loc($(1))+8)=0Dh` – в строчную переменную $(1) скопированы 8 символов (начиная с 3-го, если считать от нуля) из строчной переменной $(0). Код "0Dh" необходим для формирования строчной переменной.

CMP

Функция CMP(<addr1>,<addr2>,<len>) сравнивает байтовые последовательности с начальными адресами <addr1>,<addr2> и длиной <len> байтов в E_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.
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>)=<iexpr1>,<iexpr2>,...<iexprN> записывает значения байтов или слов в FRAM начиная с адреса <addr>. Записываемые значения задаются <iexpr1>,<iexpr2>,...<iexprN>. Значения <addr> и <iexpr> могут быть в диапазоне 0...65535.
Функция MEM[W][(<addr>)] возвращает значения байтов или слов FRAM. 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>)= <iexpr1>, <iexpr2>, ...<iexprn> производит установку или сброс битов в байте FRAM, адресуемом выражением <addr>. 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 MEMB<n>(<addr>) возвращает значение бита в байте FRAM по адресу <addr>. Функция возвращает значения "0" или "1".
Пример:

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

CHSMR

Функция CHSMR[%] <addr>,<len> возвращает контрольную сумму или контрольный циклический код области FRAM с начальным адресом <addr> из <len> байт. При наличии модификатора [%] вычисляется контрольный циклический код (см. оператор LAN).
Оператор CHSM=<iexpr> устанавливает значение системной переменной CH_SUM. Адрес CH_SUM – 0003 hex.

ОПЕРАТОРЫ РАБОТЫ С ВНЕШНЕЙ FLASH-ПАМЯТЬЮ.

FLS

Оператор FLS[W][#<FlashPage>,](<WordAddress>)=<iexpr1>,<iexpr1>...<iexprn> производит запись в FLASH. Размер FLASH – 2(4) Мбайта, соответственно адрес 21(22)-ми разрядный. Для адресации используется FlashPage – 5(6) старших разряда адреса и WordAddress – младшие 16 разрядов. После исполнения оператора значение FlashPage остается в системной переменной и, если нет необходимости в его замене, при следующем использовании оператора может быть опущено. При наличии модификатора [W] выражения <iexpr1>,<iexpr1>...<iexprn> рассматриваются как слова из 2-х байт и могут принимать значение 0...65535. Следует заметить, что при отработке оператора не проверяется был ли предварительно стерт записываемый байт/слово – пользователю необходимо самому следить за этим. Стирание в устройстве возможно только блоками не менее 4 Кбайт. Для стирания блока надо подать команду записи, установив 1 в старшем разряде FlashPage. Функция FLS[W][#<FlashPage>,](<WordAddress>) возвращает численное значение байта или слова, считанного из Flash по адресу <FlashPage><WordAddress>.

FLSB

Оператор
FLSB<n>[#<FlashPage>,](<WordAddress>)=<iexpr1>,<iexpr1>...<iexprn>производит сброс битов в байте, адресуемом <FlashPage><WordAddress>. Установить сброшенный байт можно только стиранием (оператор FLS с 1 в старшем разряде FlashPage). Минимальный размер стираемого блока – 4 Кбайт.
Функция FLSB<n>[#<FlashPage>,](<WordAddress>) возвращает значение бита в байте, адресуемом <FlashPage><WordAddress>. Функция возвращает значение "0" или "1".
Пример:

```
10 FL_PAGE=20
20 FLS#(FL_PAGE+80H),(0)=0; Стирание блока данных
30 FLS#(FL_PAGE),(10)=1,2,3
40 print "Содержимое Flash-памяти по адресу 20000B hex : ",FLS#(FL_PAGE),(11)
50 end
```

CHSM

Функция CHSM[%] <addr>,<len> возвращает контрольную сумму или контрольный циклический код области FLASH с начальным адресом <addr> из <len> байт. При наличии модификатора [%] вычисляется контрольный циклический код (см. оператор LAN).
Оператор CHSM=<iexpr> устанавливает значение системной переменной CH_SUM. Адрес CH_SUM – 0003 hex.

ОПЕРАТОРЫ ДЛЯ РАБОТЫ СО СЧЕТЧИКОМ РЕАЛЬНОГО ВРЕМЕНИ

TIME

Оператор TIME=<iexpr> устанавливает начальное значение счетчика реального времени. Значение <iexpr> (целочисленное) может быть в диапазоне (0...65535)сек. Функция TIME возвращает значение счетчика реального времени. Дискретность счета времени – 0.005 сек. Максимальное значение счетчика – 65535.995 сек.
Пример:

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

ONTIME

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

RLDT

!!! Оператор работает только в версиях модулей с функцией часов реального времени.
Оператор RLDT предназначен для установки часов реального времени. Формат оператора: RLDT=<year>,<month>,<day>,<hour>,<min>,<sec> , где
<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: 10 часов 51 минута 20 секунд 13 января 2009 года надо подать команду: RLDT=9,1,13,10,51,20 В качестве часов реального времени используется блок RTCC (REAL-TIME CLOCK AND CALENDAR) PIC24FJ64GA004. Этот блок позволяет проводить точную регулировку часов. В блоке используется внешний генератор с частотой 32 768 Гц. Блок позволяет вводить регулировку от +508 до -512 импульсов этого генератора в минуту. Регулировку можно проводить с точностью до 4 импульсов генератора в минуту. Величину регулировки записывается в регистр с адресом 626 hex:
0x7F – 508 импульсов в минуту
0x01 – 4 импульса в минуту
0x00 – отсутствие регулировки (установка по умолчанию)
0x80 – -512 импульсов в минуту
0xFF – -4 импульса в минуту
Например, чтобы ввести регулировку 12 импульсов в минуту, следует подать команду:
REG(626H)=3
Функция RLDT выдает значение часов реального времени в формате

<day>-<month>-<year> <hour>:<min>:<sec>
Вывести на печать это значение можно либо напрямую:
PRINT RLDT
либо через строчную переменную, например

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

ДИРЕКТИВЫ (ОПЕРАТОРЫ ВЫПОЛНЯЕМЫЕ ТОЛЬКО ИЗ КОМАНДНОЙ СТРОКИ)

Директива *RUN* очищает память от всех переменных и запускает BASIC-программу.
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* <ln1>-<ln2> 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.

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 отсутсвует оператор DO, соответствующий данному оператору WHILE/UNTIL (встретился оператор завершения цикла при отсутствии оператора начала оного).
"Error: it must be THEN" – цикле IF отсутствует в 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 делние 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.

ПРИЛОЖЕНИЕ 2 ВСТРОЕННЫЙ ПРОТОКОЛ MODBUS

В устройстве BM9300 реализован встроенный протокол MODBUS. Устройство использует физически те же линии связи, что и для ввода данных через BASIC-Terminal, поэтому не могут работать одновременно. Для перехода в режим поддержки протокола MODBUS необходимо в ячейку FRAM с адресом 0 занести байт 0ABh, а в следущий байт записать адрес данного устройства на шине MODBUS. После этого следует перезапустить устройство командой RST BASIC'а или сбросом питания. Для выхода из данного режима надо занести в ячейку FRAM с адресом 0 любое значение, отличное от ABh и подать команду перезапуска.

Команды MODBUS 74h и 75h работают не с EEPROM, как было во всех версиях базовой прошивки для модулей на базе PIC18Fxxxx, а с памятью FRAM (в PIC24Fxxx нет EEPROM).

Команды MODBUS 76h и 77h работают не с внутренней памятью микроконтроллера, а с внешней FLASH- памятью (AT26DF161A) – памятью программ. Объем этой памяти – 2M байт, поэтому для доступа ко всему адресному пространству FLASH введен параметр FLASH_PAGE. В нем содержатся 5 старших разрядов адреса. Для стирания соответствующей страницы памяти старший разряд этого параметра должен быть взведен в "1".

Команда 7Dh – обмен со смежным каналом UART – не реализована.

ПРИЛОЖЕНИЕ 3 РАСПРЕДЕЛЕНИЕ ПАМЯТИ FRAM

```
*****************************************************************
* Распределение памяти FRAM                                     *
* 0x0000-0x00FF - параметры BASIC-PIC                           *
* 0x0100-0x1FFF - имена переменных                              *
* 0x2000-0x3FFF - значения переменных                           *
* 0x4000-0x40FF - имена индексированных переменных (массивов)   *
* 0x4100-0x41FF - размерность + адрес начала массивов           *
* 0x4200-0x5FFF - значения индексированных переменных (массивов)*
* 0x6000-0x7FFF - значения строчных переменных                  *
*****************************************************************
```

ПРИЛОЖЕНИЕ 4 ПАРАМЕТРЫ BASIC-PIC

```
FLAG_MODBUS    0x0000    //флаг работы с MODBUS
ADR_MODBUS     0x0001    //адрес на шине MODBUS
FLASH_PAGE     0x0002    //адрес текущей страницы FLASH при работе с MODBUS
CIK_CRC        0x0003    //начальное значение циклического кода CRC16 - 2 байта
NUM_ERROR      0x0005    //номер ошибки
ADR_ERROR      0x0006    //адрес (номер строки) ошибки - 2 байта
UART_RATE      0x0008    //скорость UART
FLAG_NEW       0x000A    //флаг оператора NEW
```

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 224. 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 = 210 + 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 000 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	