Система управления «Megapirate X»

Командный интерпретатор

Редакция F
27.10.2014г.

СОДЕРЖАНИЕ

1. Введение 3
2. Работа с интерпретатором 4
3. Описание языка BASIC 5
3.1 Алфавит языка 5
3.2 Величины 5
3.3 Выражения 7
3.4 Операторы 8
3.5 Команды 9
3.6 Функции 11
3.7 Команды системы управления 12
3.8 Функции системы управления 15
4. Создание программ 18
4.1 Общие сведения 18
4.2 Нумерация и метки 19
4.3 Ветвления 20
4.4 Циклы 21
4.5 Подпрограммы, процедуры и функции 23
4.6 Ввод и вывод данных 24
4.7 Графика 26
4.8 Блокнот НСУ 28

1. Введение

Командный интерпретатор (КИ) системы управления Megapirate X служит для задания алгоритмов работы СУ, обработки ее текущих параметров, выполнения произвольных вычислительных задач, организации вывода данных через текстовый терминал и приема текстовых команд пользователя.

КИ имеет синтаксис BASIC и основан на диалекте *MMBasic (c)Geoff Graham*.

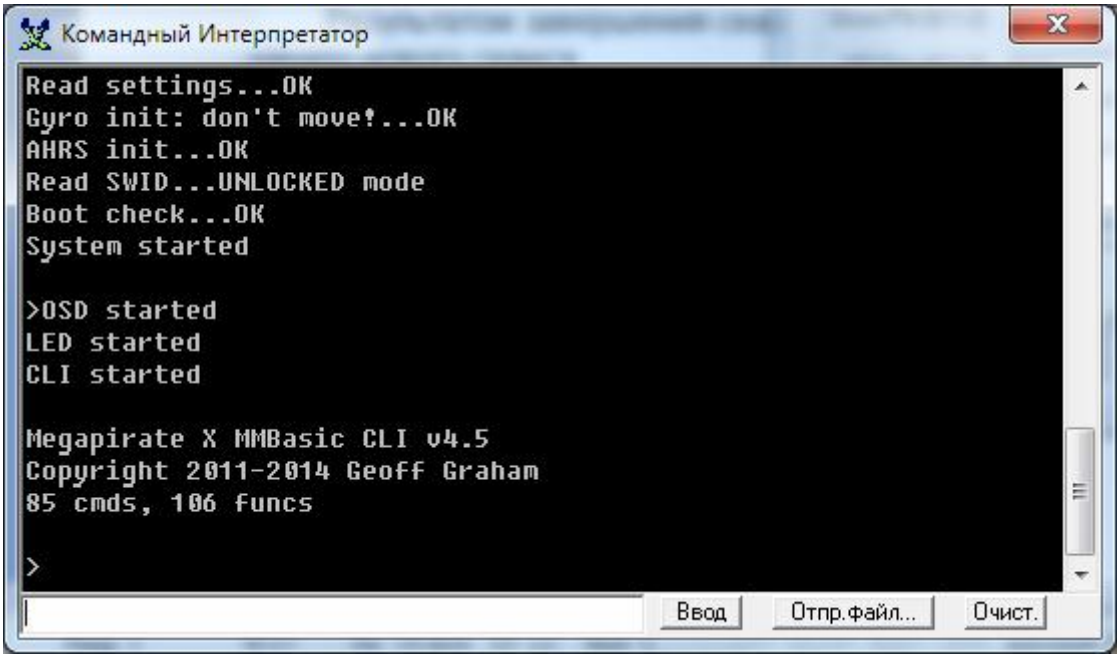
Реализована поддержка кириллицы при вводе-выводе текста и в комментариях.

Помимо терминала, вывод КИ дублируется на третьем экране ИЛС.

Для работы командного интерпретатора нужен ключ SWID.

2. Работа с интерпретатором

Для начала работы с КИ нужно нажать кнопку Terminal в основном окне НСУ. Откроется окно терминала:



Наличие приглашения **>** означает готовность интерпретатора к работе.

Ввод команд производится в белое текстовое поле и сопровождается нажатием кнопки "**ввод**" на клавиатуре или **send** в окне терминала.

Вывод данных, сообщений и листинга программы происходит в черном поле, также в нем дублируются команды, введенные с клавиатуры.

3. Описание языка BASIC

3.1 Алфавит языка

1) **Латинские буквы a-z, A-Z** — используются для набора команд языка и идентификаторов.

ВНИМАНИЕ! Команды и имена величин записываются **только латинскими буквами** и, хотя некоторые буквы обоих алфавитов схожи по написанию (A,O,K,M), они имеют разный числовой код.

2) **Буквы русского алфавита а-я, А-Я** — используются для вывода сообщений и комментариев.

3) **цифры** от 0 до 9.

4) Знаки арифметических действий:

сложение	+	умножение	*
вычитание	-	деление	/

5) Знаки отношений:

меньше	<	меньше или равно	<=
больше	>	больше или равно	>=
равно	=	не равно	< >

6) Специальные знаки:

точка	.	запятая	,	двоеточие	:
точка с запятой	;	скобки	( )	кавычки	“
восклицательный знак	!	вопросительный знак	?	знак доллара	$
процент	%	коммерческое “И”	&	апостроф	‘

3.2 Величины

1) **Числовые константы**.

Числовые величины в языке BASIC делятся на целые и вещественные. Способ изображения десятичных дробей отличается от традиционного тем, что вместо знака запятой, разделяющей целую и дробную части числа, используется точка.

Например: 1,45 $\rightarrow$ 1.45

0,5 $\rightarrow$ .5

Вещественное число можно записать двумя способами:

а) Число записывается с десятичной точкой, за которой должна следовать хоть одна цифра. Такая запись числа называется представлением числа с фиксированной точкой.

б) Десятичная дробь записывается как целое или вещественное число, умноженное на целую степень числа десять. Это показательная форма записи с плавающей точкой.

Например: 5e+2 $\rightarrow 5*10^2 \rightarrow 5*100 = 500$

3e-3 $\rightarrow 3*10^{-3} \rightarrow 3*0{,}001=0{,}003$

25e+3 $\rightarrow 0{,}25*10^3 \rightarrow 0{,}25*1000=250$

.5e+2 $\rightarrow 0{,}5*10^2 \rightarrow 0{,}5*100=50$

Целые величины можно записывать четырьмя способами:

а) в десятичном счислении - 12345, 10000, -25700
б) в шестнадцатеричном - **&h**3A, **&h**FD7EAC88
в) в восьмеричном - **&o**227503
г) в двоичном - **&b**1000101001101

В случае записи с модификатором **&** число всегда считается положительным.

2) **Символьные константы**.

Символьные константы записываются в виде последовательности символов, заключенных в кавычки. Длина символьной величины (т.е. количество символов внутри кавычек) может составлять максимум 255 символов в данной реализации языка.

Например: "это символьная константа"

3) **Переменные**.

Переменной называется величина, значение которой изменяется в ходе работы программы. Переменная характеризуется именем, типом и значением.

Имя переменной (идентификатор) — это последовательность из произвольного количества латинских букв, арабских цифр и, в отдельных случаях, специальных знаков. Однако при выборе имени следует помнить, что первый символ обязан быть буквой!

Например: B, A1, CDX, WORD$, X, ac — верно.
1B, $ABC, !X — неверно.

BASIC не различает написание строчных и прописных букв в идентификаторах, таким образом имена Abc и aBc будут считаться именем одной переменной. Служебные слова нельзя использовать в качестве имен переменных, меток и функций.

Тип переменной (вещественный или символьный) определяется множеством значений, которые может принимать данная переменная.

После имени символьной переменной ставят знак $. Если имя не содержит знака $, она считается вещественной.

Объявление переменных не требуется: область памяти для хранения переменной отводится при первом обращении к ней в программе. До присвоения переменной значения, вещественные переменные будут равны нулю, а символьные будут содержать пустую строку нулевой длины.

Все переменные в программе можно разделить на три группы — аргументы, результаты и промежуточные. Аргументы — это исходные данные программы, результатами являются полученные после выполнения программы данные, а промежуточные величины используют во вспомогательных целях или для обмена данными.

4) **Массивы**

Массив — это набор упорядоченных по номерам переменных одного типа, с общем именем. Массив характеризуется именем, типом и размерностью.

Имя массива образуется по общему правилу образования имен в BASIC’е, но его имя не должно совпадать ни с одним другим именем простых переменных, используемых в программе.

Тип данных массива может быть символьным или вещественным - это определяет наличие знака $ после имени массива.

Массивы нужно объявлять перед использованием.

Например:

DIM massiv(4,5) - объявление массива из 4 строк и 5 столбцов
DIM a$(2) - линейный массив из двух символьных строк

Размерность массива задается при объявлении и в дальнейшем изменению не подлежит. Однако, можно удалить массив целиком командой ERASE и объявить новый.

После объявления массива обращаться можно только к его отдельным элементам: a(1,2), a(1,3), a(2,1) и т.д. Использовать в качестве аргумента массив целиком нельзя.

5) **Данные**

Для изначального заполнения массивов и задания значений переменных в программе могут храниться данные. Данные представляют собой произвольный набор вещественных и символьных констант, разделенных запятыми и предваряемых оператором **DATA**.

Например:

```
DATA 1.2, 100,1500,"петя", 8, "маша"
DATA 30, 87,21
```

Данные могут быть разбросаны по всей программе, но читаются подряд от первого до последнего значения.

Присвоение переменной текущего значения из блока данных производится командой **READ**.

Например, после выполнения чтения данных абзацем выше:

```
READ a,b,d(3),c$
```

переменной c$ будет присвоено значение "петя", а переменная b = 100

При каждой операции чтения счетчик данных сдвигается на количество переменных, получивших данные. То есть две одинаковых команды READ a присвоят этой переменной два последовательных значения из блока данных.

Необходимо следить, чтобы символьная переменная не попала на чтение вещественных данных и наоборот.

Сброс счетчика чтения данных выполняется командой **RESTORE**.

3.3 Выражения

Выражение - это набор действий (**операторов**) над константами, переменными и результатами функций.

Выражение всегда имеет результат в виде вещественного числа или символьной строки.

Результат выражения служит аргументом для функции или команды, присваивается переменной или оценивается на истинность (ложь (0) или истина (не-ноль)) в ветвлении.

Выражения могут содержать только один тип данных: символьный или вещественный.

Все выражения в программе должны быть записаны по правилам языка:

а) Выражения должны быть записаны в виде линейной цепочки символов.

Например: $a_0 \rightarrow$a0

$x_{10} \rightarrow$x10

б) Нельзя опускать знаки операций.

Например: 3a$\rightarrow$3*a

a(в+с) $\rightarrow$a*(в+с)

Примеры выражений:

(a+b)*2 : результат вещественный
4*SQR(2)+d : результат вещественный
(a+b)>=(c+d) : результат "ложь" или "истина" (0 или 1)
"abc"+"DEF" : результат "abcDEF"

Функции и команды также могут в качестве аргументов содержать выражения с различным типом результатов.

3.4 Операторы

Оператор - это инструкция интерпретатору выполнить математическое действие с двумя **величинами** (операндами), слева и справа от оператора, дающее **результат**.

Совокупность операторов и операндов составляет **выражение**.

Если в выражении несколько операторов, они выполняются согласно приоритету. Если рядом стоят несколько операторов с одним приоритетом, они выполняются последовательно, слева направо, и результат правого оператора становится операндом для левого.

Очередность выполнения операторов можно изменить заключением частей выражения в скобки.

3 * 2 + 4 = 10
3 * (2 + 4) = 18 - сложение в скобках имеет приоритет выше умножения

Таблица операторов BASIC (Л - левый операнд, П - правый)

Оператор	Пр-т	Числа	Символы	Описание
^	0	+		Возведение Л в степень П
*	1	+		Умножение Л на П
/	1	+		Деление Л на П
\	1	+		Целочисленное деление Л на П
MOD	1	+		Остаток деления Л на П
+	2	+	+	Сложение Л и П
-	2	+		Вычитание
NOT	3	+		Если П=0, результат = 1, иначе 0
<>	4	+	+	если Л не равно П, рез-т =1, иначе 0
>=	4	+	+	если Л больше или равно П, рез-т =1
<=	4	+	+	если Л меньше или равно П, рез-т =1
<	4	+	+	если Л меньше П, рез-т =1
>	4	+	+	если Л больше П, рез-т =1
=	5	+	+	если Л равно П, рез-т =1
AND	6	+		логическое И для Л и П
OR	6	+		логическое ИЛИ для Л и П
XOR	6	+		исключающее ИЛИ для Л и П

Оператор **=** используется не только в выражениях, но и для присвоения значений. В этом случае его результат не используется.

Например:

a = b - операция присвоения
IF a = b THEN <действие> - выражение "если А равно Б"

Оператор **NOT** является безусловным отрицанием правого операнда и поэтому может использоваться сразу в качестве правой части другого оператора.

Например:

```
b=0 : a=3
c = a * NOT b          - результат равен 3, т.к. NOT 0 = 1
```

3.5 Команды

Команда - это инструкция интерпретатору выполнить определенное действие, не выдающая результата в виде числа или строки символов.

Команды всегда записываются или в начале строки, или после двоеточия. Если нужно выполнить несколько команд в одной строке, они отделяются двоеточием **:**

Команды, в зависимости от назначения, могут требовать **аргументы** , которые можно задавать в виде **выражений**. Если параметров несколько, они задаются через запятую.

Примеры команд:

```
PRINT "привет",a       - команда печати на экран (в терминал)
RUN   20               - запуск программы с 20 строки
Delay_ms SQR(400)      - задержка на 20мс (корень из 400)
```

Список команд BASIC, в <скобках> показаны необязательные аргументы:

Команда	Описание
? <строка>, <переменная>, <число>, ...	вывод на печать всех аргументов последовательно. Запятая - через табуляцию, точка с запятой - слитно
AUTO первый номер, <шаг>	автонумерация строк при вводе программы. Прерывание - отправить символ "~"
CLEAR	очистить все переменные в памяти
CONTINUE	продолжить программу после останова
DATA <константы через запятую>	хранилище данных в программе
DELETE номер или от-до или -до или от-	стирает строку или диапазон строк программы
DIM имя (размер1, <размер2>, ...)	задает массив переменных
DO	начало цикла DO..WHILE
ELSE	ветвление
ELSEIF	ветвление
END FUNCTION	конец описания функции
END SUB	конец описания процедуры
END	конец выполнения программы
ENDIF	ветвление
ERASE имя	удалить переменную из памяти

ERROR <"текст">	прервать программу с ошибкой
EXIT FOR	прерывание цикла FOR... NEXT
EXIT SUB	прерывание процедуры
EXIT FUNCTION	прерывание функции
EXIT DO	прервать цикл DO...WHILE
EXIT	выход в командную строку (с продолжением)
FOR	цикл FOR..NEXT
FUNCTION имя< (аргументы)>	описание функции
GOSUB число	вызов подпрограммы
GOTO число	переход на номер строки
IF	ветвление
INPUT <"приглашение">, имя, ...	приглашение ввода данных. Если переменных много - ввод через запятую
LET имя = значение	присвоение, можно опустить
LIST <номер или от-до или -до или от->	вывод листинга программы (задается диапазон строк)
LOAD	загрузить программу из флеш-памяти
LOCAL имя,<имя>	объявляет местную переменную в процедуре или функции
LOOP	цикл DO..LOOP
NEW	стереть все переменные и удалить программу
NEXT	цикл FOR...NEXT
ON	ветвление
OPTION	не применять
PRINT	вывод текста, см. ?
RANDOMIZE	сброс генератора случайных чисел
READ имя, ...	чтение данных из блока DATA в переменную
REM или '	начало комментария
RESTORE	сброс счетчика данных
RETURN	возврат из процедуры/функции
RUN <число>	запуск программы (с номера строки)
SAVE	запись программы во флеш-память
SUB имя (<аргументы>)	описание процедуры
TRON	включить трассировку (печать номера исполняемой строки) - много текста!
TROFF	выключить трассировку
WEND	конец цикла WHILE...WEND
WHILE	цикл WHILE
WRITE <строка>, <переменная>, <число>, ...	печать, эквивалент PRINT

3.6 Функции

Функция - это инструкция интерпретатору выполнить определенное действие и вернуть **результат** в виде числа или строки символов.

Функции могут быть вызваны как самостоятельно, так и участвовать в выражениях, быть аргументами для других функций и команд и т.д.

Результат функции нужно обязательно присваивать какой-либо переменной или использовать в качестве аргумента, в противном случае функция будет считаться **командой** и будет выдано сообщение об ошибке.

Функции, в зависимости от назначения, могут требовать **аргументы** , которые можно задавать в виде **выражений**.

Если функция требует аргументы, они указываются в скобках (арг1, арг2, ...) сразу после имени функции. Если аргументов нет, вызов функции не отличается от запроса обычной переменной.

Примеры функций:

```
a = CINT(2.54)              - функция округления
a$ = FORMAT$( b, "%e")  - а содержит строку - вывод b в
                            экспоненциальном виде: "1.22E-10"
b$ = LCase$("ПрИвЕт")   - результат: "привет"
```

Ниже приведен список стандартных функций BASIC. Если за именем функции стоит знак $, результат функции - строка, если скобка (- при вызове нужно задать аргументы.

Функция	Описание
Abs(число)	модуль числа (отбрасывает минус)
Asc("символ")	ASCII-код символа
Atn(число)	арктангенс
Bin$(число)	строка - двоичное представление
Chr$(число)	символ из ASCII-кода
Cint(число)	Округление с порогом 0.5
Cos(число)	косинус
Deg(число)	градусы из радиан
Exp(число)	экспонента - E в степени (число)
Fix(число)	отбрасывает дробную часть
FP$(число)	печать числа с полной разрядностью после запятой (6 разрядов)
Format$(число, "%формат")	печать числа в строке согласно формату см. гл. "Ввод и вывод данных"
Hex$(число)	шестнадцатеричное представление
Inkey$	Возвращает код нажатой кнопки или -1
Instr(число, "строка1", "строка2")	Ищет строку2 в строке1 с позиции "число", возвращает смещение или 0
Int("строка")	Преобразует строку в целое число
LCase$("строка")	Преобразует строку в нижний регистр

Left$("строка", число)	Возвращает часть строки слева длиной "число"
Len("строка")	Определяет длину строки
Log(число)	Возвращает десятичный логарифм числа
Mid$("строка", число1 <, число2 >)	Возвращает часть строки по смещению "число1" длиной "число2" или до конца
Oct$(число)	восьмеричное представление
Pi	число Пи - 3.14.....
Pos	Позиция каретки в строке вывода
Rad(число)	перевод градусов в радианы
Right$(строка, число)	Возвращает часть строки справа длиной "число2" (от конца)
Rnd	Возвращает случайное число от 0 до 0.999999(9)
Sgn(число)	Возвращает знак числа (-1, 0, 1)
Sin(число)	синус
Space$(число)	Возвращает строку из заданного кол-ва пробелов
Spc(	Аналогично Space$
Sqr(число)	квадратный корень
Str$(число)	печать числа в строку
String$(число, "символ")	Возвращает строку из заданного кол-ва символов
Tab(число)	При печати строки выводит следующий параметр со смещением "число" символов с начала строки. Если позиция занята предыдущим символом, печатает с новой строки с тем же отступом
Tan(число)	тангенс
UCase$(строка)	Переводит строку в верхний регистр
Val(строка)	Преобразует строку в число с точкой

3.7 Команды системы управления

Команда системы управления - это инструкция интерпретатору выполнить определенное действие, связанное с воздействием на систему управления Megapirate X. Как и обычные команды BASIC, команда СУ не возвращает результата и не может использоваться в выражениях.

Имена команд системы управления обычно начинаются символами **AP**_.

К оформлению команд СУ применяются те же требования, что и к обычным командам BASIC.

Примеры команд СУ:

```
AP_SetFlightMode "FM_AUTO"  - команда включить автопилот
AP_OSD 3                    - сменить экран ИЛС на вывод терминала
AP_WriteRC 8,1000           - подать в 8-й канал микшера сигнал +500мкс
AP_Land                     - выполнить автопосадку
```

Список команд СУ, в <скобках> показаны необязательные аргументы:

Команда	Описание
CLS	очистить экран терминала в ИЛС путем вывода пустой строки
RENUMBER старт, шаг	перенумеровать строки программы
VarList	вывести список текущих переменных
StampRun	запуск профилировщика (измерение времени выполнения), результат - по вызову функции StampDiff
Delay_ms число	задержка на кол-во миллисекунд
Sys_Trace	вывести системную информацию RTOS
MEMORY	вывести статистику КИ
AP_Arm	Разблокировать двигатель
AP_Disarm	Заблокировать двигатель
AP_PinMode 1..4, "тип"	Установить тип вывода Д01..Д04.Варианты: "out" - выход "od" - открытый коллектор "pu" - вход, подтяжка к питанию 10к "pd" - вход, подтяжка к земле 10к
AP_PinOut 1..4, 0..1	задать значение вывода Д01..Д04
AP_SetFlightMode "режим"	Установить режим СУ. Варианты: FM_MANUAL - Ручной FM_ACRO - Акро FM_STAB - Стаб FM_ALT - Высота FM_HOLD - Круг FM_RTH - Возврат FM_AUTO - Авто
AP_SetScriptControl	Разрешить программе ставить задачи для автопилота. Сброс - путем установки любого режима командой AP_SetFlightMode или с пульта пилота
AP_WaitForCompletion 0..1	Включить ожидание выполнения команд АП
AP_GotoLatLonAlt широта, долгота, высота	Команда АП: лететь в точку с координатами в градусах и высотой относительно базы
AP_GotoHdgDisAltB курс, расст, высота	Команда АП: лететь в точку по направлению в градусах, на расстояние и высоту в метрах относительно базы
AP_GotoNorthEastAlt север, восток, высота	Команда АП: лететь в точку со смещением в метрах на

	север и восток, на высоте в метрах относительно базы. Отрицательные значения - на юг и запад соответственно.
AP_GotoHdgDisAltC курс, расст, высота	Команда АП: лететь в точку по направлению в градусах, на расстояние и высоту в метрах относительно последней точки
AP_GotoNorthEastAltC север, восток, высота	Команда АП: лететь в точку со смещением в метрах на север и восток, на высоте в метрах относительно последней точки. Отрицательные значения - на юг и запад соответственно.
AP_Takeoff	Команда АП: Выполнить автоматический взлет согласно текущей схеме взлета. См. гл. "АСВП" Руководства по СУ
AP_Land	Команда АП: Выполнить автоматическую посадку на базу согласно текущей схеме посадки. См. гл. "АСВП" Руководства по СУ
AP_LandAt широта, долгота, взлетный курс ВПП, разница высот с базой	Команда АП: Выполнить автоматическую посадку на стороннюю ВПП согласно текущей схеме посадки, но с указанием взлетного курса ВПП и разницы высот. См. гл. "АСВП" Руководства по СУ
AP_I2C_Write16 устройство, адрес, число	Записать по шине I2C в устройство (8бит) по заданному адресу (16бит) заданное число (0-255). Подходит для записи EEPROM
AP_I2C_Write устройство, адрес, число	Записать по шине I2C в устройство (8бит) по заданному адресу (8бит) заданное число (0-255).
AP_Led число	Выбрать номер программы контроллера БАНО (1-4) или выключить его (0)
AP_OSD число	Переключить экран ИЛС (1-3) или выключить совсем (0). 1 - авиагоризонт 2 - карта 3 - терминал
AP_WriteRC 5..8, число	Записать на вход микшера 5..8 значение в диапазоне -1000..1000. Единица соответствует 0.5 мкс. Для включения в микширование альтернативная функция канала должна быть установлена "выход КИ"
AP_PTR_home	Команда подвесу камеры отслеживать базу
AP_PTR_target	Команда подвесу камеры отслеживать текущую цель
AP_PTR_targetG	Команда подвесу камеры отслеживать

	точку на земле под текущей целью
AP_PTR_mode "режим"	Установить режим работы подвеса: PTR_OFF - выключен PTR_HEAD - режим хедтрекера PTR_FWD - режим "вперед-прямо" PTR_CTRL - пропорциональный режим PTR_DIFF - дифференциальный режим PTR_LOOK - режим слежения за точкой
AP_WritePan число	Записать на выход PAN платы значение в диапазоне -1000..1000 к нейтрали 3000. Единица соответствует 0.5 мкс. Для работы команды режим подвеса должен быть установлен в "PTR_OFF"
AP_WriteTilt число	Записать на выход TILT платы значение в диапазоне -1000..1000 к нейтрали 3000. Единица соответствует 0.5 мкс. Для работы команды режим подвеса должен быть установлен в "PTR_OFF"

3.8 Функции системы управления

Функция СУ - это инструкция интерпретатору выполнить определенное взаимодействие с системой управления Megapirate X и выдать **результат** в виде числа или строки символов.

Функции СУ могут быть вызваны как самостоятельно, так и участвовать в выражениях, быть аргументами для других функций и команд и т.д.

Имена функций системы управления обычно начинаются символами **X**_ или **AP**_.

К оформлению функций СУ применяются те же требования, что и к обычным функциям BASIC.

Примеры функций:

a = X_baro_alt - a содержит текущую высоту
a$ = X_FlightMode$ - a$ содержит строку - текущий режим работы СУ, к примеру, "FM_ACRO"
b = AP_GetDistance(XPoint("target","lat"), XPoint("target", "lon"), XPoint("home", "lat"), XPoint("home", "lon"))
- расстояние между текущей целью и базой

Ниже приведен список функций СУ. Если за именем функции стоит знак $, результат функции - строка, если скобка (- при вызове нужно задать аргументы.

Функция	Описание
TimeStr$(секунды)	Возвращает форматированное время: ЧЧ:ММ:СС
StampDiff	Время, прошедшее после запуска профилировщика StampRun, микросекунды
TimeSec	Время в секундах с момента подачи питания на СУ
Round(	Округление числа с порогом 0.5
AP_PinIn(1..4)	Читает лог. состояние вывода Д01..Д04, (0 или 1)
X_FlightMode$	Возвращает строку-название режима работы СУ: FM_MANUAL - Ручной FM_ACRO - Акро FM_STAB - Стаб FM_ALT - Высота FM_HOLD - Круг FM_RTH - Возврат FM_AUTO - Авто
X_Armed	Состояние блокировки двигателя: 1 - работа 2 - блокировка
AP_GetDistance(широта1, долгота1, широта2, долгота2)	Расстояние в метрах между двумя геогр. точками
AP_GetHeading(широта1, долгота1, широта2, долгота2)	Курс в градусах из точки 1 в точку 2
AP_I2C_Test(	Проверка устройства на шине I2C. Возвращает 1, если устройство готово принимать команды.
AP_I2C_Read16(устройство, адрес)	Чтение 1 байта из устройства (8бит) по адресу (16 бит). Подходит для чтения из EEPROM
AP_I2C_Read(устройство, адрес)	Чтение 1 байта из устройства (8бит) по адресу (8 бит).
AP_ReadRC(1..8)	Возвращает значение одного из каналов радиоуправления. 1 единица - 0.5 мкс
X_roll	Крен в градусах, положительный - направо
X_pitch	Тангаж в градусах, положительный - носом вверх
X_yaw	Рыскание в градусах (направление носа)
X_target_reached	1 - текущая цель достигнута
X_gps_fix	1 - есть захват координат GPS
X_gps_sats	кол-во спутников GPS
X_gps_home_set	1 - координаты базы сохранены

X_gps_lat	текущая широта, град
X_gps_lon	текущая долгота, град
X_gps_alt	высота по GPS, м. над уровнем моря
X_gps_spd	скорость GPS, км/ч
X_gps_hdg	курс GPS, град
X_baro_alt	баровысота, м. над базой
X_baro_temp	температура в районе датчиков ИНС
X_vert_spd	вертикальная скорость, м/с
X_pitot_spd	воздушная скорость, км/ч
X_vbat1	напряжение БАТ1, В
X_vbat2	напряжение БАТ2, В
X_current	ток, А
X_mah	счетчик расхода батареи, мАч
X_rssi	датчик УВС, 0-100%
X_failsafe	1 - сигнал аварийного режима (потеря управления)
X_PTR_mode$	Режим работы подвеса камеры: PTR_OFF - выключен PTR_HEAD - режим хедтрекера PTR_FWD - режим "вперед-прямо" PTR_CTRL - пропорциональный режим PTR_DIFF - дифференциальный режим PTR_LOOK - режим слежения за точкой
Xpoint("точка", "параметр")	Возвращает один из параметров указанной точки АП. Значения "точки": "local" - текущая позиция "target" - текущая цель "home" - база Возможные параметры: "lat" - широта, град "lon" - долгота, град "alt" - высота отн. базы, м "hdg" - курс на точку, град "dis" - расстояние до точки "name" - ASCII-код символа имени точки

4. Создание программ

4.1 Общие сведения

Программа для интерпретатора - это структурированный и **пронумерованный** набор инструкций для КИ, который находится в памяти (постоянной или оперативной) и может быть запущен на выполнение в нужный момент.

Для примера, если пользователь наберет в командной строке КИ следующий текст:

PRINT " Я - программа! "

и нажмет кнопку "ввод", в результате получится не программа, а единичная команда, которая немедленно будет выполнена интерпретатором и сразу же стерта.

Но если немного изменить нашу строчку, добавив **номер строки**:

10 PRINT " Я - программа! "

- то это будет уже полноценная программа, которая будет сохранена в оперативной памяти и запустится только по команде RUN.

Запустить набранную программу с первой строки можно командой **RUN**. Также можно выбрать конкретную строку для запуска: **RUN 390**

Просмотреть имеющуюся в памяти программу можно командой **LIST**.

Программу можно сохранить для дальнейшего использования во флеш-памяти СУ командой **SAVE** и загружать в любой удобный момент по команде **LOAD**

Чтобы стереть программу из оперативной памяти целиком есть команда **NEW**, а чтобы затереть программу во флеш-памяти - нужно стереть программу в оперативной и дать команду **SAVE**.

В программу можно добавлять новые строки, увеличивая их номера. Они будут добавлены к текущей программе и упорядочены по возрастанию номеров.

Если номер строки совпадает с уже имеющимся в программе, то имеющаяся строка будет заменена на новую без запроса.

Строки из программы можно произвольно удалять командой **DELETE** <номер строки>

Если нужно вставить новую строку, а номера строк отличаются на 1, можно перенумеровать строки в программе командой RENUMBER:

RENUMBER 10,10

После этого просмотрите программу, найдите искомые строки - между ними можно будет вставить еще 10 новых строк. По старой традиции строки при вводе программы нумеруют с шагом 10, но выбор за вами: хоть подряд, хоть через 100 - учитывайте вышеописанную ситуацию.

Для помощи в нумерации строк программы при вводе служит команда **AUTO <начало, шаг>**.

AUTO 10,10

После команды выше приглашение терминала будет сразу содержать номер строки 10. После отправки строки номер сменится на 20 и т.д.

Если строка с таким номером уже есть, около номера строки будет стоять звездочка:

10* >

Автонумерацию можно прервать в любой момент, как и любые другие действия: вывод листинга, выполнение самой программы, наконец - отправкой символа прерывания.

ВНИМАНИЕ! ВАЖНО!
Найдите на клавиатуре и запомните этот символ:

~

Он называется "тильда", расположен слева от цифры 1, нажимается Shift+Ё в английской раскладке.

Этот символ - та самая "волшебная кнопка", которая поможет выбраться из, казалось бы, зависшей в КИ программы, прекратить листинг, остановить автонумерацию и вернуть себе старую добрую командную строку.

Достаточно ввести в строку этот символ и нажать "ввод", независимо от наличия приглашения на экране терминала.

Автонумерацию можно будет запустить заново, программу продолжить командой **CONTINUE**, а листинг запустить заново с нужной строки с прочерком:

LIST 290-

Для улучшения читаемости программы в нее можно добавлять **комментарии**.

AP_Land ' пора на посадку

Началом комментария является символ **'** - апостроф, на букве Э в английской раскладке клавиатуры. Все символы после апострофа до конца строки считаются комментарием, и можно писать по-русски.

Хотя при помощи КИ невозможно "повесить" систему управления и получить безответный кусок железа, при составлении программ помните, что они работают на реальном летательном аппарате. Отладку программ проводите с отключенными регулятором оборотов двигателя и сервомашинками. Для отладки управления полетом желательно пользоваться симулятором.

4.2 Нумерация и метки

В предыдущей главе мы установили, что важной частью программы является нумерация строк и научились ей пользоваться.

При работе программы, согласно задуманному алгоритму, можно переходить на произвольные строки командой GOTO:

10 PRINT "some text" : ***GOTO 30***
20 PRINT "2nd line" ' эта строка не напечатается
30 *PRINT "3rd line" ' эта - напечатается*

Однако, в большой программе, а также при перенумерации строк, структура переходов по номерам может стать неактуальной. Поэтому в программе можно устанавливать метки.

Метка - это произвольный буквенно-цифровой идентификатор, стоящий сразу за номером строки и имеющий двоеточие на конце. Переделаем наш пример:

10 PRINT "some text" : ***GOTO stroka3***
20 PRINT "2nd line" ' эта строка не напечатается
30 ***stroka3:*** *PRINT "3rd line" ' эта - напечатается*

Теперь переход происходит по метке, а не по номеру строки, и поэтому замена номера строки 30 на 500 не приведет к ошибке "строка не найдена".

4.3 Ветвления

Ветвления используются, если от программы требуется разное поведение при разных условиях. Ветвления бывают однострочные и многострочные, одно- и многопозиционные.

Внутри ветвлений, за исключением однострочных, можно вкладывать дополнительные ветвления

Условие - **выражение** сравнения или логической оценки, дающее результатом "истину" или "ложь".

Оператор присваивания "=" в условиях становится оператором проверки равенства. Т.е. 1=2 даст результат "ложь".

В качестве выражения или его части можно использовать также переменную или функцию. Нулевое значение результата функции или переменной даст результат "ложь". Пример: условие **(a) AND (b=0)** даст "истину", если А=1 и Б=0

Рассмотрим разные типы ветвлений:

IF <условие> THEN <действие -истина> ELSE <действие -ложь>

- это **Однопозиционное однострочное** ветвление Действие может быть только одно, разделители-двоеточия запрещены. Ветка ELSE может отсутствовать

```
IF <условие> THEN
        <действие -истина1>:< еще одно действие>
        <действие -истина2>
       ....
        <действие -истинаN>
ELSE
        <действие -ложь1>
        <действие -ложь2>
       ....
        <действие -ложь3>
ENDIF
```

- это **Однопозиционное многострочное** ветвление. Ветка ELSE может отсутствовать

```
IF <условие1> THEN
        <1истина>
       ...
ELSEIF <условие2> THEN
        <1ложь 2истина>
       ...
ELSEIF <условие3> THEN
        <1ложь 2ложь 3истина>
.............
ELSEIF <условиеN> THEN
        <1ложь 2ложь 3ложь.......Nистина>
ELSE
        <ВСЕ ложь>
ENDIF
```

- это **Многопозиционное многострочное** ветвление. Ветка ELSE и произвольное число ELSEIF могут отсутствовать.

В конце каждого многострочного ветвления должна стоять команда **ENDIF**.

4.4 Циклы

Циклы предназначены для многократного повторения определенных действий до наступления требуемого условия.

В КИ есть 4 вида циклов, отличающихся по способу проверки условий.

а) **FOR...NEXT**

Цикл привязан к переменной цикла. Повторяет заключенные команды до тех пор, пока переменная не достигнет значения TO. Каждый цикл увеличивает значение переменной на STEP. Если STEP не указывать, шаг по умолчанию равен 1. Пример:

```
FOR <имя>=0 TO 100 STEP 10
        <действие>
NEXT <имя>
```

б) **DO....LOOP WHILE**

В основе цикла лежит проверка какого-либо условия. Условие может проверяться в начале или в конце каждого цикла... или в обоих местах - это определяет положение команды WHILE. Цикл продолжается, пока условие после WHILE является **истиной**. Пример:

```
DO
        <действие>
LOOP WHILE <условие>
```

=====альтернатива======

```
DO WHILE <условие>
        <действие>
LOOP
```

*=====**извратl**======*

```
DO WHILE <условие1>
        <действие>
LOOP WHILE <условие2>
```

в) **DO...LOOP UNTIL**

В основе цикла также лежит проверка какого-либо условия Цикл продолжается, пока условие после UNTIL является **ложью**. Пример:

```
DO
        <действие>
LOOP UNTIL <условие>
```

=====альтернатива======

```
DO UNTIL <условие>
        <действие>
LOOP
```

г) **WHILE...WEND**

Цикл аналогичен **DO....LOOP WHILE**, но проверка условия проводится всегда вначале. Пример:

```
WHILE  <условие>
        <действие>
WEND
```

4.5 Подпрограммы, процедуры и функции

Для уменьшения размера программы часто используемые участки кода объединяют в группы, которые можно вызвать из разных мест программы одной-единственной командой, а затем вернуться к месту вызова и продолжить выполнение.

Самый простой способ - организация подпрограммы.

Подпрограмма - это любой участок программы, имеющий последней командой **RETURN**. Обращение к подпрограмме происходит командой **GOSUB** <имя/номер>, возврат из подпрограммы выполняется по команде RETURN к месту вызова.

Подпрограмма может иметь метку-имя, но может вызываться и по номеру строки. Если программа в процессе обычного исполнения попадает на код процедуры, то команда RETURN в ее конце приведет к ошибке **return without gosub**, поэтому основную программу нужно "отгородить" от подпрограммы командой **END** или "перепрыгнуть" командой **GOTO**.

Объявление подпрограммы: **<метка>**

Завершение подпрограммы: **RETURN**

Экстренный выход: **RETURN**

Все переменные основной программы доступны в подпрограмме. Созданные в подпрограмме переменные автоматически считаются **глобальными**, если не объявлены с модификатором **LOCAL**.

Пример подпрограммы:

```
240 GOSUB 280 : PRINT i      - здесь вызов подпрограммы по номеру строки
250 GOTO 300                 - перепрыгиваем подпрограмму
260 '=======
280 i=i+1                    - сама подпрограмма
290 RETURN
```

Процедура - объявленный участок программы, который "невидим" при работе основной программы. Процедура всегда имеет имя. Также в процедуру можно передавать параметры, количество и тип которых задается при объявлении процедуры.

Объявление процедуры: **SUB** имя<(параметры)>

Завершение процедуры: **END SUB**

Экстренный выход: **RETURN**

Заявленные при объявлении процедуры входные переменные создаются как локальные и сохраняются до момента выхода из процедуры.

Процедура не может возвращать данные, но может менять глобальные переменные. Любые созданные внутри процедуры переменные считаются также локальными и стираются после выхода из процедуры.

Пример процедуры:

```
240 procedura(2,4) : PRINT i    - здесь вызов процедуры по имени
250 .....                       - программа сама перепрыгнет на строку 300
260 '=======
270 SUB procedura(a,b)  - объявление процедуры с двумя входными переменными
280 i=a+b               - присвоение глобальной переменной i входных данных
290 END SUB             - конец процедуры - автоматический возврат к стр 240

300..... - продолжение основной программы
```

Функция - объявленный участок программы, по составу и "поведению" аналогичен процедуре, но при этом имеет возвращаемое значение.

Объявление функции: **FUNCTION** имя<(параметры)>

Завершение функции: **END FUNCTION**

Экстренный выход: **RETURN**

Функция имеет внутри одноименную **переменную-результат**, которая служит для возвращения значения вызывающему. Переменная-результат служит **только для записи**! Попытка чтения этой переменной приведет к вызову самой функции (рекурсия) .

Пример функции:

```
240 i=funkcia(2,4) : PRINT i      - здесь вызов функции по имени
250 .....                         - программа сама перепрыгнет на строку 300
260 '=======
270 FUNCTION funkcia(a,b)  - объявление функции с двумя входными переменными
280  funkcia=a+b                  - присвоение результату входных данных
290 END FUNCTION                  - конец функции - возврат к стр 240

300..... - продолжение основной программы
```

4.6 Ввод и вывод данных

Командный интерпретатор может выводить данные единственным способом: на экран терминала в виде текста. Для печати текста служит команда PRINT.

Для краткости вместо **PRINT** можно писать **?** - разницы нет.

```
? "hello world"    'это вопрос или утверждение? Проверьте!
```

Вкратце, чтобы сообщить что-либо оператору, нужно выполнить команду PRINT, снабдив ее некоторым количеством переменных и текста в кавычках.

Текст в кавычках выведется как есть, строковые переменные и просто числа - тоже, числовые переменные будут преобразованы в текстовый вид.

Способ отделения друг от друга нескольких аргументов определяет способ вывода их на терминал. Если разделены знаком **;** или пробелом - будут напечатаны слитно, запятая означает табуляцию в 8 символов, также см. описание функции **TAB**.

Поскольку число можно напечатать по-разному, есть множество функций, для разных форматов чисел. См. функции **TimeStr$,Oct$,Hex$,Bin$,FP$**.

Отдельно стоит функция **FORMAT$.** В качестве формата можно указывать точность до и после запятой, выравнивание, расположение, вид числа (экспоненциальный, целое, полная точность) и т.д.

Команда PRINT подразумевает перевод строки при печати. Чтобы этого не произошло, между последним аргументом и концом строки (двоеточием) нужно поставить точку с запятой :

```
? "*" ; : ? "*"    - результат : ** в одной строке
```

Пример сложной команды PRINT с различными форматами чисел:

```
280   PRINT "img" Format$(nomer,"%05d") ".jpg;" FP$(X_gps_lat) ";" FP$(X_gps_lon) ";";
290   PRINT Round(X_baro_alt) ";" X_roll ";" X_pitch ";" X_yaw
```

Результат:

img00025.jpg;23.342764;-120.827431; 17;-22; 13.3; 201.2

Ввод данных в процессе работы программы происходит через командную строку терминала. Для этого существует команда **INPUT**.

Формат: **INPUT** <приглашение>,<переменная1>,<переменная2>,.....

Вначале выводится на экран приглашение, которое оператор увидит, и затем система ожидает от оператора строку текста и нажатие кнопки "ввод". При получении строка разбивается на фрагменты, разделителем служит запятая. Переменным в списке последовательно назначаются фрагменты. Для числовых текст преобразуется в число, строковым присваивается текст между очередными запятыми. Текст, заключенный в кавычки, является одним параметром, с любым количеством запятых внутри.

Пример:

```
 INPUT "введите а,б,в", a$,b$,c
введите а,б,в> "превед,,,медвед",hello,12 <<< ввод пользователя

PRINT a$,b$,c
превед,,,медвед    hello   12
```

Если строка пустая или содержит абракадабру, программа не выпадет с ошибкой: просто будет нулевая числовая переменная или пустая строковая. Таким образом, программа должна самостоятельно контролировать верность вводимых пользователем данных и "переспрашивать" при неправильном вводе:

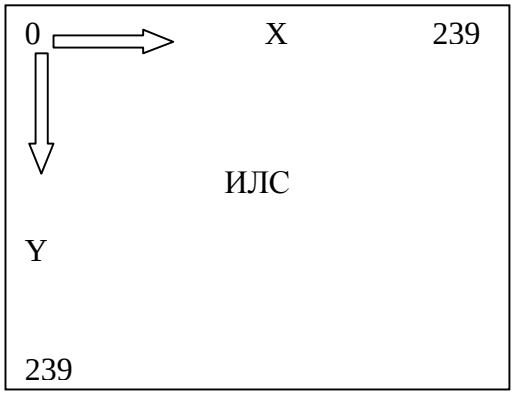
```
10 DO
20 INPUT "Введите число от 1 до 10 >",a
30 LOOP WHILE (a<1 OR a>10) ' переспрашиваем....
40 ? "Правильно! a=";a
```

4.7 Графика

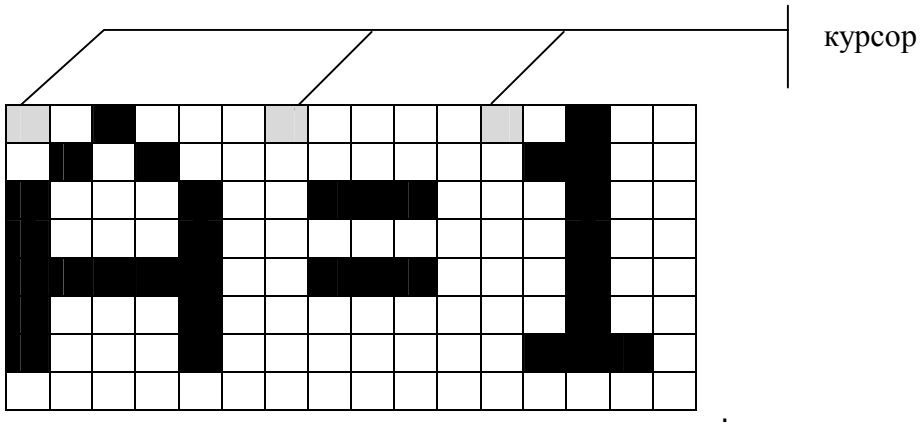
Для вывода информации на ИЛС в КИ есть два режима: текстовый и графический. В графическом режиме КИ может рисовать произвольную графику и выводить текст в любом месте экрана. В текстовом экран ИЛС копирует вывод на терминал. Переключение режимов производится при помощи команды SCREEN :

SCREEEN 0 - текстовый режим
SCREEN 1 - графический режим

Размер экрана в графическом режиме 240 на 240 точек.
Координаты каждой точки (X и Y) определяются из рисунка:



При включении графического режима вывод команды **PRINT** переводится на графический экран. Печать выполняется, начиная с позиции графического курсора. Позиция курсора не отображается, но задать ее можно командой **LOCATE x,y**. После печати каждого символа курсор перемещается на его ширину. Перевода строки не происходит. Порядок вывода текста указан на рисунке:



.

Для изменения цвета рисуемых линий и текста предназначена команда COLOR:

COLOR 1	(по умолчанию) Линии и окружности "белые", т.е. включают точки, текст - белый на прозрачном/черном
COLOR 0	Линии и окружности "прозрачные", т.е. стирают точки, текст - белое знакоместо и черный текст внутри

Имеется возможность включать-выключать отдельные точки командами PSET и PRESET независимо от установки COLOR.

Список команд для графического режима представлен в таблице:

Команда	Описание
SCREEN 0..1	Переключение режима КИ: 0 - текстовый 1 - графический
CLS	очистка текущего экрана
COLOR 0..1	Инверсия линий и текста: 1 - норма 1 - инверсия
PSET x,y	включить точку с координатами X и Y
PRESET x,y	выключить точку с координатами X и Y
LINE x1,y1,x2,y2	провести линию между двумя точками
CIRCLE x,y,R	нарисовать окружность с центром в x,y и радиусом R
LOCATE x,y	установить графический курсор на точку x,y
PRINT <строка>, <переменная>, <число>, ...	вывод на печать с позиции графического курсора. Перевод строки и табуляция не поддерживаются.

При помощи программы в КИ можно создавать анимированные графические изображения: **индикаторы, графики, маркеры** и т.д. Ниже даны некоторые советы для того, чтобы изображение выглядело "красивым" и не мерцающим.

Содержимое графического экрана каждые 20мс копируется в область отображения ИЛС. Синхронизация с процедурами рисования отсутствует, поэтому если обновление наступает посреди отрисовки линии, на ИЛС будет выведена часть графики, которая уже отрисована, а остальное изображение "присоединится" в следующем кадре, через 20мс.

Сразу определите желаемую частоту обновления изображения. Чаще 10 раз в секунду - большая вероятность мерцания.

Общая структура цикла анимации такова:

- подготовка данных для изображения
- стирание предыдущего изображения
- отрисовка нового
- экспозиция несколько циклов обновления

Такая компоновка связана с тем, что стирание и отрисовка нового изображения должны происходить как можно быстрее одно за другим, чтобы вероятность обновления ИЛС за это время была как можно ниже, во избежание мерцания. По той же причине подготовка данных (координаты точек, строки текста и т.д.) вынесена вперед: чтобы сократить время отрисовки.

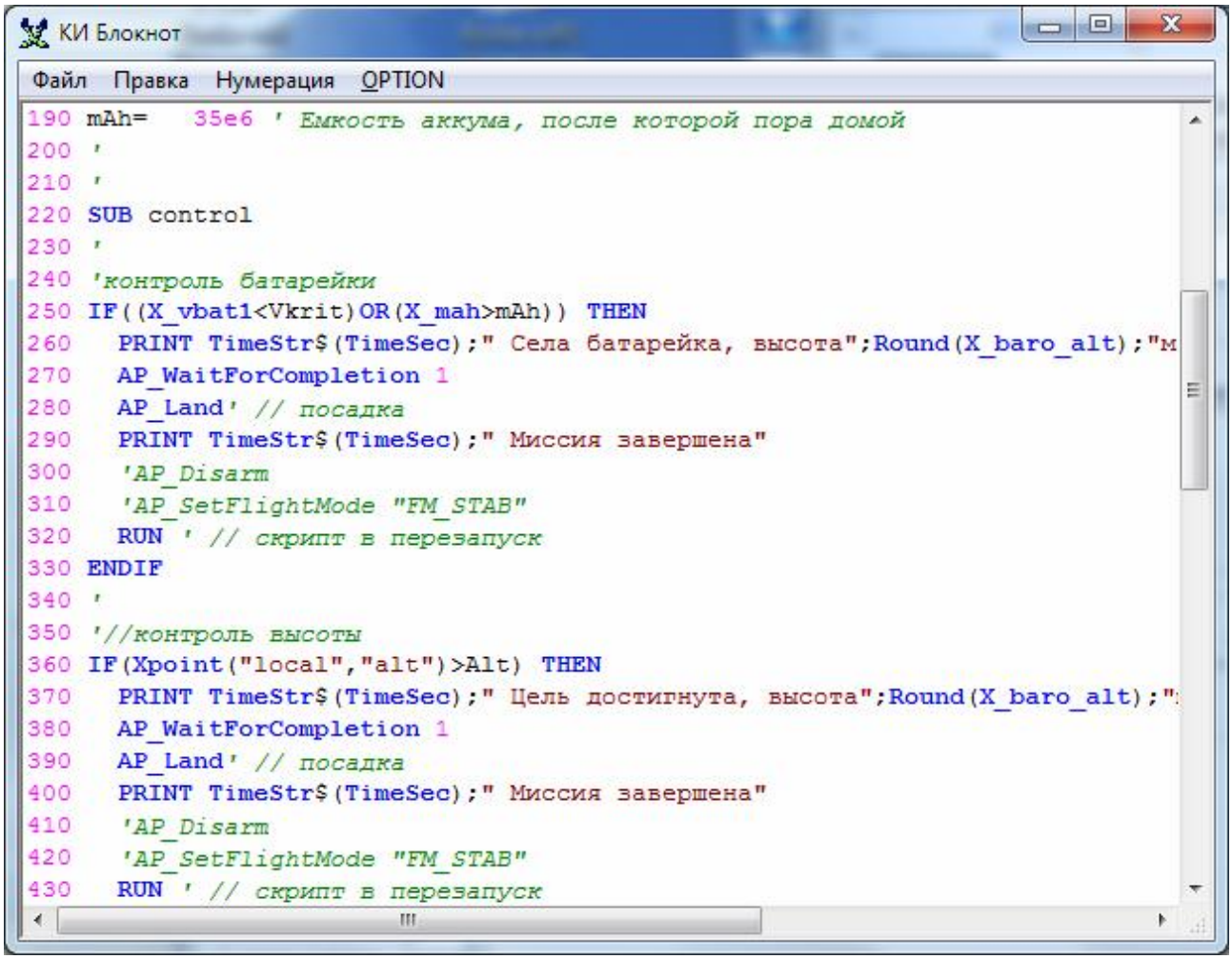
После отрисовки изображение необходимо экспонировать некоторое время (несколько циклов обновления), чтобы оно гарантированно попало на ИЛС и продержалось на глазах у пилота, затем стирать и перерисовывать с новыми данными. Если подготовка данных занимает много времени (замеряется StampDiff) - его можно "позаимствовать" у экспозиции, уменьшив задержку.

Для примера анимации загрузите в КИ программу **graphics_demo.bas**.

Действующий индикатор авиагоризонта: **ahi_demo.bas**

4.8 Блокнот НСУ

НСУ имеет встроенный текстовый редактор (блокнот) для редактирования программ КИ. В редакторе есть подсветка синтаксиса, нумерация, возможность загружать и сохранять программы на компьютер, в плату и через терминал.



Чтобы вызвать редактор, нужно нажать кнопку **Настройка АП - Пакеты - Блокнот**. Сохраненная программа будет скачана из постоянной памяти подключенного СУ и откроется окно редактора.

По окончании редактирования можно выбрать в меню "**Файл - Система - Запись во флеш**" - и отредактированная программа будет сохранена во флеш-память СУ.

Редактор имеет возможность добавлять и удалять нумерацию строк в программе - в меню **Нумерация**. Если выделить текст - нумерация меняется только в выделенном фрагменте, если выделения нет - применяется ко всему тексту.

Помимо изменения самой программы, в редакторе (и нигде более) можно задавать поведение КИ при включении СУ в меню **OPTION**.

КИ может автоматически загрузить программу в оперативную память (**автозагрузка**), и сразу ее запустить (**автозапуск**).

Работу запущенной программы все равно можно прервать "волшебной кнопкой". Однако, в некоторых случаях (помехи по телеметрии и т.д.) целесообразно запретить прерывание программы (**Нельзя прервать**).

Опции КИ можно сменить в любой момент, открыв редактор, после чего нажать "**Файл - Система - Запись во флеш**".