Quark-Forth

1.0.10 build 28

1. Назначение и основные характеристики

Quark-Forth (quark) – 32-разрядный транслятор языка Форт (Forth), предназначенный для работы в ОС Windows. Преимущественное назначение – реализация форт-машины с помощью динамически загружаемой библиотеки. Минимальный вариант транслятора представляет собой единственный файл quark.dll.

Основные особенности:

· Машинный код

· Вычисления с плавающей точкой на базе сопроцессора

· Работа с двумерной и трехмерной графикой с помощью библиотеки OpenGL

· Возможность встраивания в программы или самостоятельной работы (в exe-варианте)

Quark написан на fasm 1.67 и реализует машинный код с раздельными областями кода и данных. Выделяемая при старте память составляет 1 Мб и 256 Мб соответственно и может быть дополнительно выделена средствами ОС (GlobalAlloc). Если при старте не удается выделить 256 Мб памяти данных, производится попытка выделения 4 Мб. Стековая машина реализована путем эмуляции стеков в памяти.

Стек данных – 2048 ячеек с указателем вершины в памяти.

Стек возвратов – системный стек, адресуемый регистром ESP

Стек чисел с плавающей точкой – аппаратный стек сопроцессора (8 ячеек).

Стек структур управления (control-flow) – максимум 1024 уровня.

· Адрес

· Идентификатор структуры управления

Стек циклов – максимум 256 уровней вложенности.

· Счетчик цикла

· Максимальное значение счетчика

· Адрес для перехода

Стек контекстов загрузки – максимум 32 уровня вложенности.

Форт-машина работает посредством трансляции строк, передаваемых вызывающей программой при помощи функции Evaluate, которая принимает в качестве аргумента указатель на ASCIIZ строку и возвращает 0 в случае успеха, или номер символа, при обработке которого возникла ошибка. Формат вызова – register, т.е. указатель передается в регистре eax, результат возвращается там же.

Quark реализует виртуальный экран в памяти – массив точек 2048x2048x32bpp, доступный для чтения внешней программой, которая ответственна за правильное отображение выведенной информации. В dll встроен знакогенератор формата 8x16 пикселов в кодировке Win1251.

Кроме dll, quark существует в формате exe-файла для работы в качестве оконного приложения с графической консолью. При этом рабочая область окна представляет собой поверхность OpenGL, которая воспроизводит внутренний виртуальный экран и позволяет наложить поверх него трехмерную сцену, задаваемую пользователем с помощью векторизованного слова 3D. В том же окне реализована текстовая консоль, которая также обрабатывает сообщения от функциональных клавиш и мыши, вызывая соответствующие векторизованные слова, что позволяет гибко настраивать реакцию системы на действия пользователя в процессе работы с консолью.

Quark-Forth распространяется бесплатно, без предоставления исходных текстов. Декомпиляция, дизассемблирование или внесение изменений в программу любыми способами запрещены и лишают пользователя права на использование продукта.

Установка

Quark.dll не требует установки или регистрации в системе. Для работы достаточно поместить библиотеку в папке проекта или в папке, доступной для загрузки.

Используются следующие файлы:

Kernel32.dll
User32.dll
Gdi32.dll
Opengl32.dll
Glu32.dll

Quark.exe также не требует установки и использует те же библиотеки, что и quark.dll. Наличие самой библиотеки quark.dll при этом не требуется.

2. Взаимодействие с программным окружением

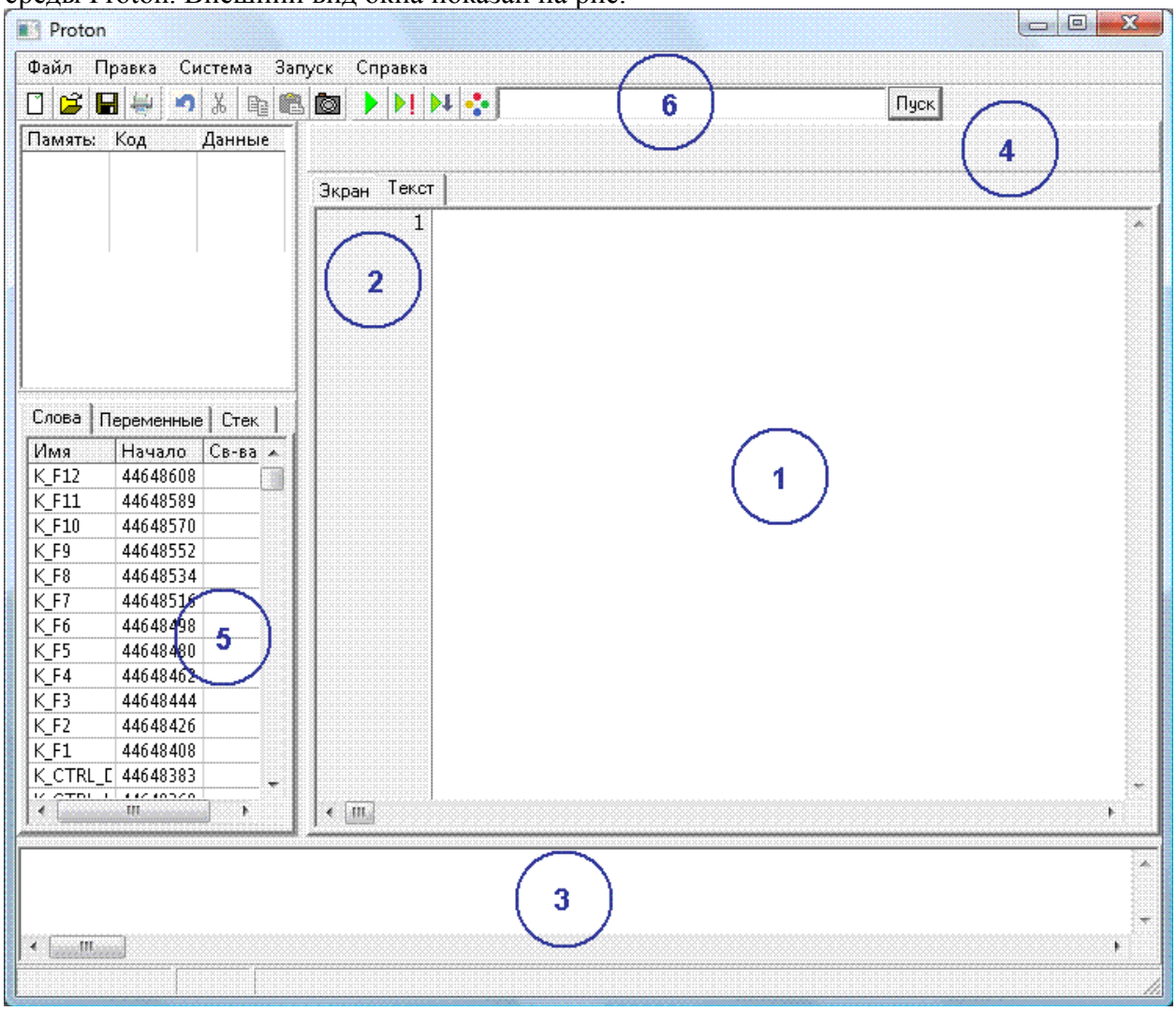
Quark.dll экспортирует следующие функции.

Имя	Описание
Evaluate(s : lpstr)	Транслирует строку, заданную указателем на первый символ и завершаемую нулем (ASCIIZ). Возвращает 0 при успешной трансляции или номер символа, при обработке которого обнаружена ошибка. Формат вызова – register (указатель на строку ожидается в регистре eax)
Init	Выделяет память кода и данных, инициализирует форт-машину
Done	Освобождает память кода и данных
GetCode	Возвращает указатель на область памяти кода
GetData	Возвращает указатель на область памяти данных
GetStack	Возвращает указатель на начало стека. Стек растет вверх
GetDepth	Возвращает текущую глубину стека (в ячейках). Верхнее число на стеке находится по адресу GetStack + (GetDepth - 1)*4
GetScreen	Возвращает указатель на область памяти виртуального экрана
SetHWindow(x : hwnd)	Устанавливает системную переменную hwnd, представляющую собой указатель на окно форт-машины. Используется, если программе-владельцу dll необходимо перенаправить поток вывода. *Примечание: зарезервировано для последующих версий, при работе с dll переменная hwnd не используется.*

Объявления функций:

```
function Evaluate(s : integer): integer; external 'quark.dll';
function Init(): integer; external 'quark.dll';
function Done(): integer; external 'quark.dll';
function GetCode(): integer; external 'quark.dll';
function GetData(): integer; external 'quark.dll';
function GetStack(): integer; external 'quark.dll';
function GetDepth(): integer; external 'quark.dll';
function GetScreen(): integer; external 'quark.dll';
function SetHWindow(hwnd : integer): integer; external 'quark.dll';
```

Все функции используют формат вызова register (параметр передается в eax, результат возвращается в eax). При работе регулярно вызывается векторное слово <TIMER>, интервал вызова которого задается системной переменной TIMER-INTERVAL (в миллисекундах). В качестве среды разработки используется приложение Proton, написанное в среде Lazarus. Оно реализует оконную IDE, позволяющую редактировать исходный текст программы на Форте, запускать dll или exe-версию Quark, просматривать содержимое виртуального экрана (только 2D) в отдельной закладке рабочего пространства среды Proton. Внешний вид окна показан на рис.



Цифрами обозначены:

1 – Редактор исходного текста

2 – Переключение закладок с редактором и содержимым виртуального экрана

3 – Консоль

4 – Панель настраиваемых инструментальных кнопок (настройка по правой кнопке мыши)

5 – Просмотр содержимого словаря и стека

6 – Кнопки быстрого запуска и консоль форт-машины

3. Описание слов

Соглашения

В стековой нотации используются следующие соглашения.

Стековая нотация представлена в виде:

<Состояние стека до выполнения слова> – <состояние стека после выполнения слова>

Вершина стека находится справа, т.е. запись 2 3 – 5 означает, что до выполнения слова на вершине стека находилось число 3, а под ним число 2; после выполнения эти числа оказались удалены, а на вершине вместо них оказалось число 5.

Если исполняемое слово влияет на стек возвратов, его состояние записывается после символов «R:». Если исполняемое слово влияет на стек чисел с плавающей точкой, его состояние записывается после символов «F:»

Данные на стеке представлены, как правило, буквами латинского алфавита. В отдельных случаях используются обозначения, отражающие характер представленных величин. Например:

Addr – адрес

Data – данные

T – логическое значение (истина/ложь)

Cnt, counter – значение счетчика

uA – беззнаковая интерпретация числа A

S, str - строка

hFile – индекс файла

Форматы данных

Основным форматом представления чисел является 32-разрядное целое число, занимающее одну ячейку (cell) стека данных. По умолчанию число рассматривается как представленное в дополнительном двоичном коде. При необходимости использовать числа меньшей разрядности активными являются младшие разряды числа. При записи числа меньшей разрядности прочие форматы игнорируются, при чтении числа на стек – заполняются нулями.

Основным форматом представления чисел с плавающей точкой является число двойной точности (double precision), занимающее 8 байт памяти. Возможны чтение и запись чисел одинарной точности (single precision, также short float), занимающих 4 байта. При этом необходимо иметь в виду, что все операции на стеке сопроцессора происходят во внутреннем представлении (comp), занимающем 80 бит, и преобразуются в числа двойной или одинарной точности только при записи в память.

Основным форматом представления строк является строка, завершаемая байтом 0 (ASCIIZ строка). Адресом строки считается адрес ее первого символа. При создании строки перед ней предварительно размещается 4-байтный счетчик символов.

При вычислении логических выражений любое ненулевое значение считается ИСТИНОЙ, нулевое – ЛОЖЬЮ.

Прочие форматы представления данных зависят от используемых библиотек и функций.

Кварк не использует методы статического или динамического контроля типов значений на стеках. Например, не контролируется «помещение на целочисленный стек вещественного числа, прочитанного как целое», или арифметические операции с указателями.

Для упрощения работы со словами, активно использующими стековые перестановки, введен локальный стек. Он представляет собой статически выделенный, глобальный массив памяти, способный хранить 8-байтные значения, что позволяет помещать на него как целые числа, так и числа в формате с плавающей точкой.

Слова для работы со стеком

Имя	**Стековая нотация**	**Описание**
DUP	A - A, A	Дублирование числа на вершине стека
DROP	A -	Удаление числа с вершины стека
SWAP	A, B – B, A	Обмен верхних чисел на вершине стека
OVER	A, B – A, B, A	Копирование на вершину стека второго сверху числа
ROT	A, B, C – B, C, A	Вращение трех верхних чисел в соответствии со стековой нотацией
NIP	A, B – B	Удаление второго сверху числа
PICK	An.. A1, A0, n – An.. A1, A0, An	Копирование на вершину стека числа, отстоящего на n от вершины стека
XCHG	An.. A1, B, n – B, An-1, A1, An	Обмен числа на вершине стека и числа, отстоящего на n от вершины стека
>R	A – R : A	Перемещение числа с вершины стека данных на вершину стека возвратов
R>	R: A – A	Перемещение числа с вершины стека возвратов на вершину стека данных
R@	R: A – A, R: A	Копирование числа с вершины стека возвратов на вершину стека данных
RDROP	R: A – R:	Удаление числа с вершины стека возвратов
DEPTH	-- D	Помещение на стек количества чисел до выполнения этого слова
LOCALDEPTH	-- D	Помещение на стек количества чисел на локальном стеке до выполнения этого слова
CLEARSTACK	An.. A0 --	Очистка стека
FDUP	F: A – F: A, A	Дублирование числа на вершине стека сопроцессора
F2DUP	F: A, B – F: A, B, A, B	Дублирование пары чисел на вершине стека сопроцессора
FDROP	F: A -	Удаление числа с вершины стека сопроцессора
FSWAP	F: A, B – F: B, A	Обмен верхних чисел на стеке сопроцессора
FOVER	F: A, B – F: A, B, A	Копирование на вершину стека сопроцессора второго сверху числа
FROT	F: A, B, C – F: B, C, A	Вращение трех верхних чисел на стеке сопроцессора в соответствии со стековой нотацией
>L	A – L: A	Переместить число со стека данных на локальный стек
L>	L: A -- A	Переместить число с локального стека на стек данных
L@	L: A – A, L: A	Скопировать число с локального стека на стек данных
LDROP	L: A --	Удалить число с локального стека
L>F	L: A – F: A	Переместить число с локального стека на стек сопроцессора в формате двойной точности
F>L	F: A – L: A	Переместить число со стека сопроцессора на локальный стек в формате двойной точности
L>SF	L: A – F: A	Переместить число с локального стека на стек сопроцессора в формате одинарной точности
SF>L	F: A – L: A	Переместить число со стека сопроцессора на локальный стек в формате одинарной точности
FRAME{	--	Отметить положение стекового кадра

}FRAME	--	Удалить запись о стековом кадре
ARG0.. ARG9		Положить на стек число, находящееся на 0..9 ячеек от вершины стека, как она была запомнена словом FRAME{

Арифметика и логика

Имя	**Стековая нотация**	**Описание**
+	A, B – A+B	Сложение верхних чисел на стеке
-	A, B – A-B	Вычитание верхних чисел на стеке
1+	A – A+1	Прибавление 1 (инкремент) к верхнему числу на стеке
1-	A – A-1	Вычитание 1 (декремент) из верхнего числа на стеке
CELL+	A – A+4	Прибавление 4 (размера ячейки в байтах) к верхнему числу на стеке
CELL-	A – A-4	Вычитание 4 (размера ячейки в байтах) их верхнего числа на стеке
FCELL+	A – A+8	Прибавление 4 (размера числа с плавающей точкой в байтах) к верхнему числу на стеке
FCELL-	A – A-8	Вычитание 4 (размера числа с плавающей точкой в байтах) их верхнего числа на стеке
*	A, B – A*B	Умножение верхних чисел на стеке
U*	A, B – uA*uB	Беззнаковое умножение верхних чисел на стеке
/	A, B – A/B	Деление верхних чисел на стеке
U/	A, B – uA/uB	Беззнаковое деление верхних чисел на стеке
2*	A – A*2	Быстрое умножение на 2 верхнего числа на стеке
2/	A – A/2	Быстрое деление на 2 верхнего числа на стеке
MOD	A, B – A mod B	Остаток от деления верхних чисел
UMOD	A, B – uA mod uB	Остаток от беззнакового деления верхних чисел
/MOD	A, B – A/B, A mod B	Частное и остаток от деления верхних чисел
U/MOD	A, B – uA/uB, uA mod uB	Частное и остаток от беззнакового деления верхних чисел
AND	A, B – A and B	Побитное И над верхними числами
OR	A, B – A or B	Побитное ИЛИ над верхними числами
XOR	A, B – A xor B	Побитное ИСКЛЮЧАЮЩЕЕ ИЛИ над верхними числами
SHL	A - B	Логический сдвиг верхнего числа влево на 1 разряд
SHLA	A - B	Арифметический сдвиг верхнего числа влево на 1 разряд
SHR	A - B	Логический сдвиг верхнего числа вправо на 1 разряд
SHRA	A - B	Арифметический сдвиг верхнего числа вправо на 1 разряд
LSHIFT	A, count – B	Логический сдвиг числа A на count разрядов влево
RSHIFT	A, count - B	Логический сдвиг числа A на count разрядов вправо

<<	A, count – B	Логический сдвиг числа A на count разрядов влево. То же, что LSHIFT
>>	A, count - B	Логический сдвиг числа A на count разрядов вправо. То же, что RSHIFT
NEGATE	A - -A	Изменение знака числа на вершине стека
ABS	A - |A|	Модуль числа на вершине стека
SGN	A - -1|0|1	Заменяет число на вершине стека значением -1, если число отрицательное, нулем, если число равно нулю, и 1, если число положительное
NOT	A - B	Логическое НЕ числа на вершине стека (0 заменяется на -1, остальные значения заменяются на 0)
WITHIN	A, MIN, MAX - T	Проверка попадания числа A в диапазон MIN, MAX (включительно), оставляет на стеке ИСТИНУ, если A находится в указанном диапазоне
=	A, B – T	Проверка равенства двух верхних чисел. Оставляет на стеке ИСТИНУ, если числа были равны
>	A, B – T	Проверка отношения «БОЛЬШЕ» двух верхних чисел. Оставляет на стеке ИСТИНУ, если A>B
<	A, B – T	Проверка отношения «МЕНЬШЕ» двух верхних чисел. Оставляет на стеке ИСТИНУ, если A<B
>=	A, B – T	Проверка отношения «БОЛЬШЕ ИЛИ РАВНО» двух верхних чисел. Оставляет на стеке ИСТИНУ, если A>=B
<=	A, B – T	Проверка отношения «МЕНЬШЕ ИЛИ РАВНО» двух верхних чисел. Оставляет на стеке ИСТИНУ, если A<=B
<>	A, B – T	Проверка неравенства двух верхних чисел. Оставляет на стеке ИСТИНУ, если числа не были равны
U>	A, B – T	Проверка отношения «БОЛЬШЕ» двух верхних чисел, рассматриваемых как числа без знака. Оставляет на стеке ИСТИНУ, если A>B
U<	A, B - T	Проверка отношения «МЕНЬШЕ» двух верхних чисел, рассматриваемых как числа без знака. Оставляет на стеке ИСТИНУ, если A<B
MIN	A, B – Min	Выбор минимального из двух верхних чисел на стеке.
MAX	A, B – Max	Выбор максимального из двух верхних чисел на стеке.
RANGE	A, MIN, MAX - B	Коррекция числа A таким образом, чтобы оно гарантированно находилось в интервале MIN..MAX (включительно).
UMIN	A, B – C	Выбор минимального из двух верхних чисел на стеке. Числа рассматриваются как беззнаковые.
UMAX	A, B – C	Выбор максимального из двух верхних чисел на стеке. Числа рассматриваются как беззнаковые.
URANGE	A, MIN, MAX - B	Коррекция числа A таким образом, чтобы оно гарантированно находилось в интервале MIN..MAX (включительно). Числа рассматриваются как беззнаковые.

RGB	R, G, B - color	Вычисление цвета по компонентам R, G, B
BYTES	A – A	Дать число байтов для хранения A (не меняет число на стеке)
WORDS	A – A*2	Дать число байтов для хранения A слов
CELLS	A – A*4	Дать число байтов для хранения A двойных слов
FLOATS	A – A*8	Дать число байтов для хранения A чисел с плавающей точкой двойной точности
W>D	wA - A	Преобразовать слово на стеке в двойное слово знакорасширением
-TH	A, INDEX – Adr	По начальному адресу массива A и индексу элемента INDEX вернуть адрес этого элемента (каждый элемент занимает 4 байта)
-FTH	A, INDEX - Adr	По начальному адресу массива A и индексу элемента INDEX вернуть адрес этого элемента (каждый элемент занимает 8 байт)
S>F	A – F: X	Число со стека данных перенести на стек сопроцессора
F>S	F: X – A	Число со стека сопроцессора перенести на стек данных
F+	F: A, B – F: A+B	Сложить два числа с плавающей точкой на стеке сопроцессора
F-	F: A, B – F: A-B	Вычесть два числа на стеке сопроцессора
F*	F: A, B – F: A*B	Перемножить два числа на стеке сопроцессора
F/	F: A, B – F: A/B	Дать частное от деления двух чисел на стеке сопроцессора
FSIN	F: X – F: sin(X)	Вычислить синус от аргумента на стеке сопроцессора
FCOS	F: X – F: cos(X)	Вычислить косинус от аргумента на стеке сопроцессора
FSINCOS	F: X – F: sin(X), cos(X)	Вычислить синус и косинус от аргумента на стеке сопроцессора (операция выполняется быстрее, чем раздельное вычисление синуса и косинуса)
FSQRT	F: X – F: sqrt(X)	Вычислить квадратный корень от числа на стеке сопроцессора
FATAN	F: X – F: arctan(X)	Вычислить арктангенс от числа на стеке сопроцессора
FPATAN	F: X, Y – F: arctan(X/Y)	Вычислить арктангенс от отношения двух чисел на стеке сопроцессора. Допускается задавать Y равным 0
FABS	F: X – F: |X|	Абсолютное значение (модуль) от числа на стеке сопроцессора.
FNEGATE	F: X – F: -X	Изменение знака числа на стеке сопроцессора
F0>	F: X – T	На стек данных кладется результат логического выражения «число на стеке сопроцессора больше нуля»
F0<	F: X – T	На стек данных кладется результат логического выражения «число на стеке сопроцессора меньше нуля»
F0=	F: X – T	На стек данных кладется результат логического выражения «число на стеке сопроцессора равно

		нулю»
F>	F: A, B – T	На стек данных кладется результат логического выражения «второе сверху число на стеке сопроцессора больше, чем первое»
F<	F: A, B – T	На стек данных кладется результат логического выражения «второе сверху число на стеке сопроцессора меньше, чем первое»
F=	F: A, B – T	На стек данных кладется результат логического выражения «два верхних числа на стеке сопроцессора равны»
F2^X	F: X – F: 2^X	Вычислить 2^X (учитывается и целая, и дробная части X)
FEXP	F: X – F: exp(X)	Вычислить exp(X) (учитывается и целая, и дробная части X)
FGAUSS	F: X, S – F: exp(X/S)^2	Вычислить функцию exp((x/s)^2)
FLOG2	F: X – F: log2(X)	Вычислить логарифм от X по основанию 2
FLN	F: X – F: ln(x)	Вычислить натуральный логарифм от X
FLG	F: X – F: log10(X)	Вычислить логарифм от X по основанию 10
X*X	A1, A2, N – F: X	Вычислить свертку (сумму произведений соответствующих элементов) массивов, заданных начальными адресами A1, A2, длиной N элементов. Элементы первого массива – слова (16-разрядные числа), элементы второго массива – числа с плавающей точкой в формате short float (4 байта)
D*D	A1, A2, N – F: X	Вычислить свертку (сумму произведений соответствующих элементов) массивов, заданных начальными адресами A1, A2, длиной N элементов. Элементы массивов – числа с плавающей точкой в формате double
SSE:F(T)*G(T)DT	A1, A2, N – xmm0: X	Вычислить свертку (сумму произведений соответствующих элементов) массивов, заданных начальными адресами A1, A2, длиной N*4 элементов. Элементы массивов - числа с плавающей точкой в формате short float (4 байта). Операция производится с использованием инструкций расширения SSE, работающих с четверками чисел short float. Рекомендуется выравнивать адреса массивов на границу 16 байт для получения максимальной производительности. Результат находится в регистре xmm0 (частичные суммы свертки).
SSE!	A -	Регистр xmm0 записать по адресу A. Записываются четыре числа в формате short float
SSE:[X]+=[Y]	A1, A2, N -	Добавляет элементы массива, начинающегося с адреса A2, к соответствующим элементам массива, начинающимся с адреса A1. Размеры массивов – N четверок (N*4 чисел) в формате short float. Операция производится с использованием инструкций расширения SSE.

Операции с памятью

Имя	**Стековая нотация**	**Описание**
@	A – D	Читать из памяти число D по адресу A
!	D, A -	Записать в память по адресу A число D
+!	D, A -	Прибавить к содержимому памяти по адресу A число D
ON	A -	Записать ИСТИНУ (-1) в ячейку памяти с адресом A
OFF	A -	Записать ЛОЖЬ (0) в ячейку памяти с адресом A
F@	A – F: X	Читать из памяти с адресом A число двойной точности и положить его на стек сопроцессора
F!	A, F: X -	Записать число со стека сопроцессора в память по адресу A в формате вещественного числа двойной точности
F+!	A, F: X -	Прибавить число со стека сопроцессора к числу в памяти по адресу A в формате вещественного числа двойной точности
SF@	A – F: X	Читать из памяти с адресом A число одинарной точности и положить его на стек сопроцессора
SF!	A, F: X -	Записать число со стека сопроцессора в память по адресу A в формате вещественного числа одинарной точности
SF+!	A, F: X -	Прибавить число со стека сопроцессора к числу в памяти по адресу A в формате вещественного числа одинарной точности
,	D -	Перенести число со стека данных в память данных
C,	bD -	Перенести один байт со стека данных в память данных
W,	wD -	Перенести одно слово со стека данных в память данных
F,	F: X -	Перенести число двойной точности со стека сопроцессора в память данных
SF,	F: X -	Перенести число одинарной точности со стека сопроцессора в память данных
C@	A – D	Читать из памяти байт D по адресу A
C!	D, A -	Записать в память по адресу A байт D
[C]@	A – D	Читать из памяти число D по адресу A
[C]!	D, A -	Записать в память по адресу A число D
[C],	D -	Перенести число со стека данных в память кода
[C]C@	A – D	Читать из памяти байт D по адресу A
[C]C!	D, A -	Записать в память по адресу A байт D
[C]C,	D -	Перенести байт со стека данных в память кода
W@	A – D	Читать из памяти слово D по адресу A
W!	D, A -	Записать в память по адресу A слово D
MOVE	Src, dest, len --	Скопировать len ячеек из области памяти с начальным адресом Src в область памяти с начальным адресом dest.

CMOVE	Src, dest, len --	Скопировать len байтов из области памяти с начальным адресом Src в область памяти с начальным адресом dest.
SMOVE	Src, dest --	Скопировать строку области памяти с начальным адресом Src в область памяти с начальным адресом dest. Признаком завершения копирования является байт 0.
MOVE<	Src, dest, len --	Скопировать len ячеек из области памяти с начальным адресом Src в область памяти с начальным адресом dest, начиная с конца области памяти.
CMOVE<	Src, dest, len --	Скопировать len байтов из области памяти с начальным адресом Src в область памяти с начальным адресом dest, начиная с конца области памяти.
SMOVE<	Src, dest --	Скопировать строку области памяти с начальным адресом Src в область памяти с начальным адресом dest. Признаком завершения копирования является байт 0.
CCOMPARE	A1, A2, count -- F	Сравнить области памяти, заданные адресами A1 и A2 и длиной count байт. Вернуть на стеке ИСТИНУ, если области идентичны и ЛОЖЬ в противном случае.
S=	A1, A2 - F	Сравнить строки, заданные адресами A1 и A2. Строка, заданная адресом A1, должна завершаться нулем. Вернуть на стеке ИСТИНУ, если строки, не включая ограничитель, идентичны.
FILL	D, A, len --	Заполнить область памяти с начальным адресом A и длиной len ячеек значением D
CFILL	bD, A, len --	Заполнить область памяти с начальным адресом A и длиной len байт значением bD
ALLOT	D -	Изменить указатель свободного байта памяти данных на D
[C]ALLOT	D -	Изменить указатель свободного байта памяти кода на D
INPORT	A – D	Положить на стек байт D, прочитанный из порта BB с адресом A. Порт должен быть доступен приложению для чтения.
INPORTW	A – D	Положить на стек слово D, прочитанное из порта BB с адресом A. Порт должен быть доступен приложению для чтения.
INPORTD	A – D	Положить на стек двойное слово D, прочитанное из порта BB с адресом A. Порт должен быть доступен приложению для чтения.
OUTPORT	D, A -	Записать в порт BB с адресом A байт D. Порт должен быть доступен приложению для записи.
OUTPORTW	D, A -	Записать в порт BB с адресом A слово D. Порт должен быть доступен приложению для записи.
OUTPORTD	D, A -	Записать в порт BB с адресом A двойное слово D. Порт должен быть доступен приложению для записи.

Системные переменные и предопределенные константы

Имя	**Размер, байт**	**Описание**
STATE	4	Состояние транслятора (0 - интерпретация/1 - компиляция)
BASE	4	Основание системы счисления
LIMIT	4	Размер области памяти данных
[C]LIMIT	4	Размер области памяти кода
GETSTACK		Начальный адрес стека данных (стек растет вверх)
GETCODE		Начальный адрес области памяти кода
GETDATA		Начальный адрес области памяти данных
CONTEXT	4	Указатель на контекстный словарь
CURRENT	4	Указатель на текущий словарь
SOURCE	4	Идентификатор транслируемого файла
SDEPTH	4	Глубина стека контекстов загрузки
DEBUG	4	Флаг, означающий, что после компиляции каждого слова требуется компиляция вызова векторного слова DEBUG. Служит для организации отладки программы.
PROCESSED	4	Флаг, означающий, что обработчик сообщений Windows, установленный пользователем, обработал сообщение, и не требуется вызов DefWindowProc
LASTKEY	4	Код последнего введенного с клавиатуры символа
CONSOLEX	4	Координата X консоли
CONSOLEY	4	Координата Y консоли
CONSOLEL	4	Максимальная длина текста, выводимого в консоли
MAXX	4	Максимальное количество пикселов в текстовой части экрана (используется при автоматическом переводе строки)
OPAQUE	4	Логическая переменная, управляющая прозрачностью фона при выводе текста встроенными средствами
MOUSE-X	4	Координата X мыши
MOUSE-Y	4	Координата Y мыши
MOUSE-Z	4	Координата Z мыши (при наличии колеса прокрутки). При сообщении от колеса прокрутки в переменную записывается 120 или -120, являющиеся результатами, возвращаемыми функцией, определяющей направление смещения колеса прокрутки.
TIMER_INTERVAL	4	Интервал вызова векторного слова <TIMER>
SYMBOLS	4096	Начальный адрес встроенного знакогенератора
CAN-DISPATCH	4	Логическая переменная, разрешает обработку введенных чисел словом DISPATCH-NUMBER
ROTATEANGLE	4	Системная переменная для вызова функций OpenGL

glX	4	Системная переменная для вызова функций OpenGL
glY	4	Системная переменная для вызова функций OpenGL
glZ	4	Системная переменная для вызова функций OpenGL
INPUTBUF	258	Буфер для ввода данных словом INPUTDIALOG
MAJOR-VERSION	4	Константа – номер версии
MINOR-VERSION	4	Константа – номер версии
SUBVERSION	4	Константа – номер версии
BUILD-VERSION	4	Константа – номер версии

Константы

TRUE
FALSE
RED
GREEN
BLUE
BLACK
WHITE
1.0
10.0
PI
FPI

Структуры управления и объявление объектов

Имя	**Стековая нотация**	**Описание**
:	--	Начать определение нового слова с именем, взятым из входного потока.
;	--	Завершить определение слова.
PROC	--	Начать определение нового слова с именем, взятым из входного потока. То же, что :
ENDPROC	--	Завершить определение слова. То же, что ;
[	--	Перевести транслятор в режим интерпретации. Слово немедленного исполнения.
]	--	Перевести транслятор в режим компиляции.
CONSTANT	C --	Создать слово, кладущее на стек константу. Значение константы снимается со стека в момент исполнения слова CONSTANT. Имя слова выбирается из входного потока.
VARIABLE	--	Создать слово, кладущее на стек начальный адрес области памяти. Имя слова выбирается из входного потока. Зарезервировать в памяти данных место для одной ячейки.
FLOAT	--	Создать слово, кладущее на стек начальный адрес

		области памяти. Имя слова выбирается из входного потока. Зарезервировать в памяти данных место для одного числа в формате double float (8 байт).
ARRAY	Size --	Создать слово, кладущее на стек начальный адрес области памяти. Имя слова выбирается из входного потока. Зарезервировать в памяти данных место для Size ячеек.
"	-- Str	Выбрать из входного потока строку, завершенную символом кавычки ("). Положить на стек адрес первого символа строки, завершенной символом с кодом 0
QUAN	--	Создать слово, кладущее на стек значение переменной. Переменной присваивается начальное значение 0. Имя слова выбирается из входного потока. Запись в переменную осуществляется словом TO
VALUE	Value --	Создать слово, кладущее на стек значение переменной. Переменной присваивается начальное значение, снимаемое со стека. Имя слова выбирается из входного потока. Запись в переменную осуществляется словом TO
VECT	--	Создать слово-переменную, передающее управление на адрес, записанный в нее. Переменной присваивается начальное значение, указывающее на NOOP. Имя слова выбирается из входного потока. Запись в переменную осуществляется словом TO. Чтение записанного адреса без его исполнения осуществляется словом FROM
CREATE	--	Создать слово, имя которого выбирается из входного потока, с семантикой «положить на стек данных адрес первого свободного байта в области данных, как он был известен на момент создания слова»
DOES>	--	Добавить в семантику определения переход на текущий адрес области кода.

Пример:

VARIABLE A
0 **CONSTANT** НОЛЬ
FLOAT X
100 **ARRAY** TABLE[]
0 **VALUE** Y
QUAN Z
VECT MyFunction
: DOUBLECONSTANT **CREATE** , , **DOES>** DUP @ SWAP 4 + @ ;
2 3 DOUBLECONSTANT CC

Имя	**Стековая**	**Описание**

	нотация	
VOCABULARY	--	Создать новый словарь в текущем словаре, выбрав его имя из входного потока. Семантикой создаваемого слова является «установить данный словарь контекстным».
DEFINITIONS	--	Сделать контекстный словарь текущим.
FORTH	--	Установить FORTH контекстным словарем.

Пример:
VOCABULARY MyVoc
MyVoc **DEFINITIONS** // сделать MyVoc контекстным и текущим
: WordInMyVoc ; // добавляется слово в MyVoc
FORTH // сделать FORTH контекстным
WordFromForth // для поиска доступны только слова из FORTH
: WordStillInMyVoc ; // добавление слов происходит в MyVoc
DEFINITIONS // теперь и поиск, и добавление происходят только для FORTH

Имя	**Стековая нотация**	**Описание**
IF	T --	Слово немедленного исполнения. Компилирует операцию «проверить число на стеке, выполнить переход вперед, если условие не выполняется».
THEN	--	Слово немедленного исполнения. Компилирует код для завершения оператора условного выполнения
ELSE	--	Слово немедленного исполнения. Компилирует код для завершения ветки оператора условного выполнения, отмечает начало ветки, выполняющейся при условии ЛОЖЬ на стеке.

Пример:
: NEWWORD
2 2 + 4 = **IF** “ 2 + 2 = 4” PRINT **THEN**
;:
NEWWORD
2 2 + 4 = **IF** “ 2 + 2 = 4” PRINT **ELSE** “ 2 + 2 <> 4” PRINT **THEN**
;

Имя	**Стековая нотация**	**Описание**
BEGIN	--	Слово немедленного исполнения. Компилирует операцию «отметить начало цикла с проверкой условия выхода»
AGAIN	--	Слово немедленного исполнения. Компилирует безусловный переход к началу цикла
UNTIL	--	Слово немедленного исполнения. Компилирует переход к началу цикла, если на стеке ЛОЖЬ
WHILE	--	Слово немедленного исполнения. Компилирует операцию «выполнять тело цикла, если на стеке ИСТИНА»
REPEAT	--	Слово немедленного исполнения. Компилирует

		переход к началу цикла

Пример:
BEGIN
“ Бесконечный цикл“ PRINT
AGAIN
0
BEGIN
“ Выполняется 10 раз“ PRINT
1 + DUP 10 =
UNTIL

Пример:
0
BEGIN
DUP 10 <
WHILE
1 +
REPEAT

Имя	**Стековая нотация**	**Описание**
DO	end start --	Слово немедленного исполнения. Сформировать код начала цикла со счетчиком, берущего со стека начальное и предельное значения. Цикл НЕ выполняется для предельного значения.
FOR	start --	Слово немедленного исполнения. Начать цикл со счетчиком, сняв со стека начальное значение и установив предельное в 0. Цикл выполняется для предельного значения счетчика.
LOOP	--	Слово немедленного исполнения. Завершить итерацию цикла со счетчиком, увеличив счетчик на 1.
+LOOP	N --	Слово немедленного исполнения. Завершить итерацию цикла со счетчиком, увеличив счетчик на N
NEXT	--	Слово немедленного исполнения. Завершить итерацию цикла со счетчиком, уменьшив счетчик на 1.
I	-- I	Положить на стек текущее значение счетчика.
J	-- J	Положить на стек текущее значение счетчика объемлющего цикла
K	-- K	Положить на стек текущее значение счетчика второго объемлющего цикла
IJ	-- J*Imax + I	Положить на стек текущее значение выражения J*Imax + I
IMAX	-- Imax	Положить на стек предельное значение счетчика цикла
IADDR	-- Iaddr	Положить на стек адрес начала итерации цикла

Пример:
“ Печатаем числа 0 – 9” PRINT
10 0 **DO**
I .
LOOP
“ Обратный отсчет“ PRINT
10 **FOR**
I .
NEXT
“ Печатаем четные числа” PRINT
20 0 **DO**
I .
2 +**LOOP**

Имя	**Стековая нотация**	**Описание**
TO	x --	Снять число со стека и записать его в QUAN, VALUE или VECT- переменную, имя которой выбирается из входного потока. Слово работает как в режиме интерпретации, выполняя запись, так и в режиме компиляции, компилируя код для выполнения этой операции.
AS	x --	То же, что TO
+TO	x --	Снять число со стека и увеличить на эту величину значение QUAN, VALUE или VECT- переменной, имя которой выбирается из входного потока. Слово работает как в режиме интерпретации, выполняя запись, так и в режиме компиляции, компилируя код для выполнения этой операции.
FROM	--	Положить на стек содержимое QUAN, VALUE или VECT-переменной, имя которой выбирается из входного потока. Слово немедленного исполнения. Слово предназначено для чтения текущего значения VECT-переменной, так как упоминание имени такой переменной без слова FROM приводит к исполнению кода с адреса, хранящегося в этой переменной.
'	--	Выбрать имя слова из входного потока. Положить на стек адрес начала кода этого слова.
[']	--	Выбрать имя слова из входного потока. Положить на стек адрес начала кода этого слова. Слово немедленного исполнения.
USE	--	То же, что [']
CASE	N --	Скомпилировать код «начать оператор выбора по селектору». Величина-селектор снимается со стека. Слово немедленного исполнения.
ENDCASE	--	Скомпилировать код, завершающий управляющую конструкцию выбора по селектору. Слово немедленного исполнения.
OF	x --	Скомпилировать код, проверяющий число на

		вершине стека, и исполняющий код до слова ENDOF или BREAK, если это число равно значению-селектору. Слово немедленного исполнения.
ENDOF	--	Скомпилировать код, завершающий текущую ветвь управляющей конструкции выбора по селектору. Слово немедленного исполнения.
BREAK	--	Скомпилировать код, завершающий текущую ветвь управляющей конструкции выбора по селектору и прекращающий проверки на текущем уровне вложенности CASE. Слово немедленного исполнения.
>OF	N --	Скомпилировать код, проверяющий число на вершине стека, и исполняющий код до слова ENDOF или BREAK, если это число больше значения-селектора. Слово немедленного исполнения.
<OF	N --	Скомпилировать код, проверяющий число на вершине стека, и исполняющий код до слова ENDOF или BREAK, если это число меньше значения-селектора. Слово немедленного исполнения.
<OF>	Nmin, Nmax --	Скомпилировать код, проверяющий два числа на вершине стека и исполняющий код до слова ENDOF или BREAK, если значение-селектор находится в диапазоне Nmin..Nmax включительно. Слово немедленного исполнения.

Пример.

```
VARIABLE X

: TEST-X
  X @ CASE
    0 OF " X равен нулю" PRINT BREAK
    "X не равен нулю" PRINT
    10 99 <OF> "X – двузначное число" PRINT ENDOF
    50 OF "X равен 50" PRINT ENDOF
  END-CASE
;
```

Файловые операции

Имя	**Стековая нотация**	**Описание**
L	Str --	Загрузить и транслировать файл, заданныйименем.
OPEN	Str -- hFile	Открыть существующий файл, заданный именем, для чтения и вернуть на стек идентификатор файла.
OPENRW	Str -- hFile	Открыть существующий файл, заданный именем, для чтения и записи и вернуть на стек идентификатор файла.

NEWFILE	Str -- hFile	Создать файл, заданный именем, для чтения и вернуть на стек идентификатор файла. Если файл с таким именем существует, уничтожить его и создать файл нулевой длины.
CLOSE	hFile --	Закрыть файл, заданный идентификатором.
FILESIZE	hFile -- size	Вернуть размер файла, заданного идентификатором, в байтах
READFILE	hFile, Buf, count --	Читать count байт из файла, заданного идентификатором hFile, в память с начальным адресом Buf
READCHAR	hFile -- Char	Прочитать один символ из файла, заданного идентификатором hFile и положить его код на стек данных.
READLINE	hFile, Buf --	Читать одну строку из файла, заданного идентификатором hFile, в память с начальным адресом Buf.
WRITEFILE	hFile, Buf, count --	Записатьь count байт из файла, заданного идентификатором hFile, из памяти с начальным адресом Buf
WRITECHAR	hFile, char --	Записать символ char в файл, заданный идентификатором hFile.
HF-OUT	-- A	Положить на стек VALUE-переменную – идентификатор файла для быстрого вывода.
EMITF	C --	Записать символ, кодируемый значением со стека данных, в файл с идентификатором HFOUT
PRINTF	Str --	Печатать строку, заданную адресом первого символа и завершаемую нулем, в файл с идентификатором HF-OUT
CRF	--	Вывести символы перевода строки в файл с идентификатором HF-OUT
.F	X --	Печатать число с вершины стека в файл с идентификатором HF-OUT
U.F	uA --	Печатать число с вершины стека в беззнаковом формате в файл с идентификатором HF-OUT
F.F	F: X --	Печатать число с вершины стека сопроцессора в файл с идентификатором HF-OUT

Управление программой

Имя	**Стековая нотация**	**Описание**
NOOP	--	Нет операции
\		Игнорировать текст до конца строки
//		Игнорировать текст до конца строки
EXECUTE	A --	Выполнить слово, заданное адресом на стеке
BRANCH,	A --	Скомпилировать переход к адресу A
CALL,	A --	Скомпилировать вызов по адресу A
LIT,	D --	Скомпилировать литерал D
[COMPILE]	--	Выбрать из входного потока слово и скомпилировать его вызов, независимо от наличия у слова признака немедленного

		исполнения
BYE	--	Выйти из программы. Векторное слово. При выходе освобождаются области памяти кода и данных.
BYE1	--	Вариант слова BYE, установленный при старте программы.
HERE	-- Adr	Положить на стек адрес первого свободного байта в области памяти данных.
[C]HERE	-- Adr	Положить на стек адрес первого свободного байта в области памяти кода.
PARSE	-- Str	Удалить ведущие разделители. Выделить из входного потока очередной токен, ограниченный пробелом или символом перевода строки.
INPUT	Adr --	Вызвать диалоговое окно для ввода значения переменной. Если в окне нажата кнопка OK, проверить возможность перевода строки в целое число и записать его по адресу Adr. Если форматное преобразование невозможно, повторить запрос. Если нажата кнопка Cancel, выйти из диалога и не производить перезапись.
FINPUT	Adr --	Вызвать диалоговое окно для ввода значения вещественной переменной. Если в окне нажата кнопка OK, проверить возможность перевода строки в вещественное число и записать его по адресу Adr. Если форматное преобразование невозможно, повторить запрос. Если нажата кнопка Cancel, выйти из диалога и не производить перезапись.
SINPUT	Adr --	Вызвать диалоговое окно для ввода текста. Если в окне нажата кнопка OK, записать полученную строку по адресу Adr в формате ASCIIZ. Если нажата кнопка Cancel, выйти из диалога и не производить перезапись.
INPUTDIALOG	-- Btn	Вызывает диалоговое окно ввода текстовой строки. Возвращает на стеке 0, если нажата кнопка «Отмена», и 1, если нажата кнопка «ОК». Размещает полученную строку по адресу INPUTBUF
LOADLIBRARY	Str – hDll	Загрузить динамическую библиотеку, заданную адресом первого символа, вернув на стеке идентификатор библиотеки.
GETPROCADDRESS	hDll, Str - Adr	По идентификатору библиотеки и имени функции, заданной адресом первого символа, вернуть исполняемый адрес этой функции.
API0	Addr – X	Вызвать функцию API, заданную ее адресом, без параметров. Результат выполнения функции вернуть на стек данных.
API1	A, Addr – X	Вызвать функцию API, заданную ее адресом, с одним параметром A, находящимся на стеке данных. Результат выполнения функции вернуть на стек данных.
API2	A, B, Addr – X	Вызвать функцию API, заданную ее адресом, с

		двумя параметрами, находящимися на стеке данных. Порядок записи параметров в стековой нотации слова соответствует порядку записи в прототипе вызова функции. Результат выполнения функции вернуть на стек данных.
API3	A, B, C, Addr – X	Вызвать функцию API, заданную ее адресом, с тремя параметрами, находящимися на стеке данных. Порядок записи параметров в стековой нотации слова соответствует порядку записи в прототипе вызова функции. Результат выполнения функции вернуть на стек данных.
API4	A, B, C, D, Addr -- X	Вызвать функцию API, заданную ее адресом, с четырьмя параметрами, находящимися на стеке данных. Порядок записи параметров в стековой нотации слова соответствует порядку записи в прототипе вызова функции. Результат выполнения функции вернуть на стек данных.
API	A, .. An, N, Addr -- X	Вызвать функцию API, заданную ее адресом, с N параметрами, находящимися на стеке данных. Порядок записи параметров в стековой нотации слова соответствует порядку записи в прототипе вызова функции. Допускается указывать 0 параметров. Результат выполнения функции вернуть на стек данных.
hInstance	-- hInstance	Идентификатор запущенного экземпляра приложения
hwnd	-- hwnd	Идентификатор окна приложения

Печать и графика

Имя	**Стековая нотация**	**Описание**
.	A --	Печать числа со стека
U.	uA --	Печать числа со стека данных как числа без знака.
F.	F: X --	Печать числа со стека сопроцессора в инженерном формате.
EMIT	C --	Печать символа, заданного его кодом; векторное слово.
EMIT1	C --	Вариант слова EMIT, установленный по умолчанию.
PRINT	Str --	Печать строки, заданной адресом первого символа и завершаемой кодом ASCII 0
COUNT	Str -- count	Определить количество символов в строке, заданной адресом первого символа. Конец строки определяется по байту-ограничителю 0.
TYPE	Str, count --	Печать строки, заданной адресом первого символа и длиной
.”	--	Печатать строку, завершаемую кавычкой. Строка выбирается из входного потока, т.е. следует за

		словом .”
CR	--	Выполняет перевод строки
PIXEL	X, Y, Color --	Установить пиксел с координатами X, Y цветом Color. Векторное слово.
PIXEL1	X, Y, Color --	Вариант слова PIXEL, установленный по умолчанию
GETPIXEL	X, Y -- Color	Вернуть цвет пиксела в координатах X, Y. Векторное слово.
GETPIXEL1	X, Y -- Color	Вариант слова GETPIXEL, установленный по молчанию
CLS	--	Очистить внутренний экранный буфер.
3D	--	Векторное слово, вызывается после копирования внутреннего буфера на экран, предназначено для наложения трехмерной сцены поверх двумерного буфера.
GOTOXY	X, Y --	Установить текстовый курсор в позицию Row, Col (позиция отсчитывается в знакоместах)
WHEREXY	– X, Y	Вернуть позицию текстового курсора в координатах Row, Col (позиция отсчитывается в знакоместах)
WHEREX	– X	Вернуть координату Row текстового курсор (позиция отсчитывается в знакоместах)
WHEREY	– Y	Вернуть координату Col текстового курсора (позиция отсчитывается в знакоместах)
TEXTXY	X, Y --	Установить текстовый курсор в позицию X, Y (позиция отсчитывается в пикселях)
WHERETEXTXY	– X, Y	Вернуть координаты текстового курсора X, Y (позиция отсчитывается в пикселях)
SETCOLOR	Color --	Установить цвет для отображения символов
SETBGCOLOR	BGColor --	Установить фон для отображения символов
GETCOLOR	-- Color	Вернуть на стек текущий цвет для отображения символов
GETBGCOLOR	-- BGColor	Вернуть на стек текущий цвет для отображения фона символов
HLINE	X, Y, len, Color --	Рисовать горизонтальную линию из позиции X, Y, длиной len и цветом Color
VLINE	X, Y, len, Color --	Рисовать вертикальную линию из позиции X, Y, длиной len и цветом Color
DECIMAL	--	Установка десятичной системы счисления
HEX	--	Установка шестнадцатеричной системы счисления
BIN	--	Установка двоичной системы счисления
3DROTATE	--	Выполнить вращение трехмерной сцены. Аргументы должны быть заданы в системных переменных ROTATEANGLE glX glY glZ в формате short float.
3DTRANSLATE	--	Выполнить перенос координат трехмерной сцены. Аргументы должны быть заданы в системных переменных glX glY glZ в формате short float.
REDRAW	--	Форсировать перерисовку экранного буфера и трехмерной сцены.
$	--	Обработать входные сообщения, затем

		перерисовать экранный буфер и трехмерную сцену.

Векторизованные слова

Quark-Forth широко использует векторизацию словарных статей. Слова, имеющие на практике альтернативные варианты реализации, объявлены в трансляторе как векторные (VECT), что позволяет в ходе выполнения программы изменять адрес кода, который будет выполнен при их упоминании. Важным свойством векторного слова является то, что код, вызывающий такое слово, обращается не к статически скомпилированному адресу, а к векторному слову, выполняющему переход к адресу, хранящемуся в переменной. Это позволяет, например, описать обработчик сообщений Windows таким образом, что он вызывает векторизованные словарные статьи, которые при старте Quark-Forth не выполняют никаких действий, но после записи в векторные слова адресов обработчиков эти обработчики будут автоматически вызываться без необходимости переписывания цикла обработки сообщений.

Список векторизованных слов

Сообщения при нажатии функциональных и специальных клавиш.
K_F12 K_SHIFT_F12 K_CTRL_F12 K_ALT_F12
K_F11 K_SHIFT_F11 K_CTRL_F11 K_ALT_F11
K_F10 K_SHIFT_F10 K_CTRL_F10 K_ALT_F10
K_F9 K_SHIFT_F9 K_CTRL_F9 K_ALT_F9
K_F8 K_SHIFT_F8 K_CTRL_F8 K_ALT_F8
K_F7 K_SHIFT_F7 K_CTRL_F7 K_ALT_F7
K_F6 K_SHIFT_F6 K_CTRL_F6 K_ALT_F6
K_F5 K_SHIFT_F5 K_CTRL_F5 K_ALT_F5
K_F4 K_SHIFT_F4 K_CTRL_F4
K_F3 K_SHIFT_F3 K_CTRL_F3 K_ALT_F3
K_F2 K_SHIFT_F2 K_CTRL_F2 K_ALT_F2
K_F1 K_SHIFT_F1 K_CTRL_F1 K_ALT_F1
K_PGDOWN K_SHIFT_PGDOWN K_CTRL_PGDOWN K_ALT_PGDOWN
K_PGUP K_SHIFT_PGUP K_CTRL_PGUP K_ALT_PGUP
K_END K_SHIFT_END K_CTRL_END K_ALT_END
K_HOME K_SHIFT_HOME K_CTRL_HOME K_ALT_HOME
K_DOWN K_SHIFT _DOWN K_CTRL_DOWN
K_UP K_SHIFT_UP K_CTRL_UP
K_RIGHT K_SHIFT _RIGHT K_CTRL_RIGHT
K_LEFT K_SHIFT_LEFT K_CTRL_LEFT
K_DEL K_SHIFT_DEL K_CTRL_DEL
K_INS K_SHIFT_INS K_CTRL_INS
K_ENTER
K_ESC
K_BACKSPACE
KEYDOWN – вызывается при получении сообщения wm_keydown, перед этим код символа записывается в переменную LASTKEY
KEYUP – вызывается при получении сообщения wm_keyup, перед этим код символа записывается в переменную LASTKEY
K_CHAR (вызывается при получении сообщения wm_char, перед этим код символа записывается в переменную LASTKEY)

Сообщения от мыши.
<MOUSE_MOVE> - вызывается при перемещении мыши
<MOUSE_LEFT> - вызывается при нажатии левой кнопки мыши
<MOUSE_RIGHT> - вызывается при нажатии правой кнопки мыши
<MOUSE_DBLLEFT> - вызывается при двойном нажатии левой кнопки мыши
<MOUSE_DBLRIGHT> - вызывается при двойном нажатии правой кнопки мыши
<MOUSE_LEFT_UP> - вызывается при отпускании левой кнопки мыши
<MOUSE_RIGHT_UP> - вызывается при отпускании правой кнопки мыши
<MOUSE-WHEEL> - вызывается при прокрутке колеса мыши

Прочие сообщения
<TIMER> - вызывается в процессе обработки сообщений системного таймера
<DEBUG> - вызывается после исполнения каждого слова, если оно было скомпилировано при установленной в ИСТИНУ переменной DEBUG
OK – вызывается после обработки каждой строки
DISPATCH-FNUMBER – вызывается после помещения на стек числа с плавающей точкой
DISPATCH-NUMBER – вызывается после помещения на стек целого числа

Слова DISPATCH-FNUMBER и DISPATCH-NUMBER вызываются только при ненулевом значении переменной CAN-DISPATCH.

<WINDOW_RESIZE> - вызывается при изменении размеров окна программы.
WINMSG – вызывается при получении любого сообщения от Windows. Параметры сообщения помещаются в переменные WMSG, WPARAM, LPARAM. Переменная PROCESSED должна содержать ИСТИНУ (не ноль), если дальнейшая обработка сообщения функцией DefWindowProc не требуется.

OpenGL

Quark-Forth основан на использовании «виртуального экрана» - хранящемся в памяти массиве, описывающем цвета пикселей в формате 2048x2048x32bpp. Версия транслятора в виде динамически загружаемой библиотеки предусматривает, что основная программа самостоятельно отобразит содержимое этой области памяти на экране в требуемом формате. Версия в виде исполняемого файла использует библиотеку OpenGL для отображения содержимого виртуального экрана в окне графического приложения.

Список констант OpenGL, доступных в ядре.
GL_LIGHT_MODEL_TWO_SIDE
GL_BACK
GL_FRONT
GL_POSITION
GL_SHININESS
GL_EMISSION
GL_SPECULAR
GL_AMBIENT_AND_DIFFUSE
GL_DIFFUSE
GL_AMBIENT
GL_FLAT
GL_COLOR_MATERIAL
GL_LIGHT1
GL_LIGHT0
GL_LIGHTING

GL_DEPTH_BUFFER_BIT
GL_COLOR_BUFFER_BIT
GL_DEPTH_TEST
GL_POLYGON
GL_QUAD_STRIP
GL_QUADS
GL_TRIANGLE_FAN
GL_TRIANGLE_STRIP
GL_TRIANGLES
GL_LINE_STRIP
GL_LINES
GL_POINTS

Версии документа

Версия и дата	**Описание**
1.0	Начальная реализация документа
1.1, 16/02/2010	Описание приведено в соответствие с версией 1.0.8.24
1.2, 03/09/2010	Описание приведено в соответствие с версией 1.0.10.28