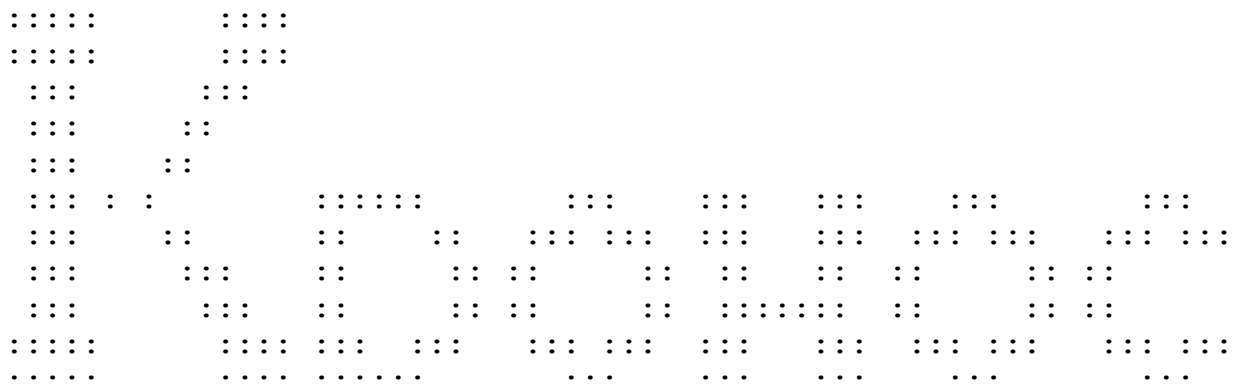
- Сибирское отделение АН СССР -

- Институт систем информатики -



АРХИТЕКТУРА ПРОЦЕССОРОВ СЕМЕЙСТВА КРОНОС

Новосибирск - 90

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Архитектура процессоров семейства КРОНОС ориентирована на поддержку языков высокого уровня (Си, Модула-2, Паскаль, Оккам), что дает возможность реализовать новейшие концепции в области использования ЭВМ. Наличие 32-разрядного машинного слова позволяет использовать процессоры семейства для решения вычислительных задач. Широкое адресное пространство (до 2 миллиардов слов) дает возможность создания виртуальной памяти для объектно-ориентированных моделей вычислений и тем самым поддерживает разработку систем искусственного интеллекта. Наличие механизма прерываний по событиям, по синхронизации процессов, а также компактность кода программ позволяет с уверенностью утверждать, что процессоры семейства КРОНОС могут успешно применяться в системах реального времени.

Любой процессор семейства может быть использован как в составе отдельной ЭВМ, так и в мультипроцессорных комплексах.

В настоящее время семейство включает три разработки: 2.2, 2.5, 2.6. У всех процессоров одна система команд; они полностью совместимы программно и различаются лишь по внутреннему функциональному устройству, быстродействию и конструктивному исполнению.

Логику функционирования всех блоков процессора реализует блок микропрограммного управления. Две шины данных объединяют арифметико-логическое устройство, блок регистров, быстрый аппаратный стек на 7 слов, устройства выборки команд и ввода/вывода.

Двухшинная внутренняя структура процессоров позволяет выполнять бинарные операции на стеке (сложение, вычитание, логические И, ИЛИ и т.д.) за один такт. Таким образом, за один такт исполняется большинство команд, что отвечает основным идеям RISC-архитектуры. Микропрограммное управление упрощает устройство процессоров и дает возможность реализовать сложные команды типа вызова процедуры.

Все узлы процессоров выполнены на советских ТТЛ и ТТЛШ микросхемах широкого применения серий 155, 531, 1802, 1804, 589, 556.

Процессор	Кронос 2.2	Кронос 2.5	Кронос 2.6
Конструктив	Электроника-60	Intel	Евромеханика
Количество плат	1	2	2[2)]
Шина	Q-bus 22	Multibus-1	локальная[3)]
Ширина микрокоманды, бит	56	64	64

Объем микропрограмм, Кслов	2	2	4
Прямо адресуемая память	4 Мбайт	2,5 Мбайт[1)]	8 Гбайт
Тактовая частота, мГц	4	3	3
Число простых операций над стеком, млн/сек	0,6	1	1,5

[1)]Существенным отличием Кронос 2.5 является наличие локальной памяти объемом 0,5-2 Мбайт - в зависимости от применяемых микросхем. Остальная память - на шине Multibus-1 (до 1 Мбайт).
[2)]В минимальный комплект входят: плата обрабатывающего тракта (АЛУ, стек, регистры), плата микропрограммного управления, плата локальной памяти (0.5-2 Мбайт), плата адаптера шины ввода/вывода.
[3)]Все устройства объединены локальной синхронной 32-разрядной шиной. Сам процессор не зависит от конкретной шины ввода/вывода. Настройка на конкретную шину производится с помощью соответствующего адаптера. К локальной шине могут быть добавлены платы памяти, адаптера межпроцессорной связи, контроллера к локальной сети и накопителя на магнитных дисках, плата памяти кода (при разделении плат кода и данных), bitmap-дисплея и т.д..

ЧАСТЬ I. ВИРТУАЛЬНАЯ МОДУЛА-2 МАШИНА

Виртуальная Модула-2 машина (ВМ2М) и ее программный и аппаратный интерпретаторы обеспечивают процесс исполнения программ. В данном разделе будет охарактеризован М-код, пояснена его интерпретация и проиллюстрированы отдельные команды.

Механизм ввода/вывода существенно зависит от конфигурации конкретного оборудования и здесь не рассматривается.

Основные отличия ВМ2М от традиционных машин:

1) вычисление выражений на быстром стеке небольшой аппаратно фиксированной глубины. Освобождение этого стека (копирование в память) при вызове процедур-функций;

2) разделение кода и области данных любого процесса, следствием чего является повторно-входимость всех программ и даже их отдельных частей (модулей);

3) отказ от абсолютной адресации в сегменте кода. Наличие таблицы смещений начал процедур упрощает вызовы процедур;

4) развитые виды адресации отражают понятия современных языков программирования. Имеется адресация локальных, глобальных, внешних объектов и объектов статически вложенных процедур;

5) специальные команды упрощают реализацию циклов, вызовов, выбирающих операторов и других сложных конструкций;

6) таблица раздельно загруженных модулей позволяет организовать динамическое связывание и загрузку программ;

7) имеются команды для работы с мультизначениями.

Для понимания этого раздела требуется знакомство с языком программирования Модула-2. В дальнейшем считается, что читатель знаком со следующими понятиями Модулы-2:

ПРОГРАММА
МОДУЛЬ
ИМПОРТ-ЭКСПОРТ ОБ'ЕКТОВ
ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ
ПРОЦЕДУРА
ЛОКАЛЬНАЯ ПРОЦЕДУРА
ЛОКАЛЬНЫЕ ПЕРЕМЕННЫЕ

1. Составные части виртуальной машины

ВМ2М состоит из процессора, стека для хранения обрабатываемых данных и вычисления выражений (арифметического стека), дескриптора исполняемого процесса и памяти.

1.1. Процессор

Процессор ВМ2М - устройство для интерпретации инструкций управления (**М-кода**).

1.2. Арифметический стек

А-стек (арифметический стек, стек выражений, expression stack) - быстрый стек небольшой, аппаратно фиксированной глубины, шириной в одно слово (здесь и далее: ширина машинного слова 32 бита), над которым определены операции помещения слова на стек (Push) и снятия слова с верхушки стека со счеркиванием (Pop). Переполнение и исчерпание стека отслеживаются аппаратно с возбуждением соответствующего прерывания.

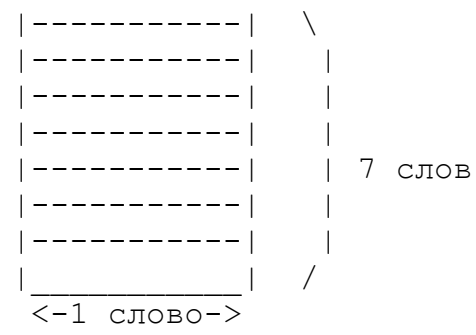


Рис.1. А-стек (стек выражений)

1.3. Процесс и его дескриптор

В каждый момент времени ВМ2М выполняет некоторый **процесс**. Процесс - это последовательность операций при выполнении программы и данные, используемые этими операциями.

Дескриптор процесса - 8 слов в памяти, содержащих указатели на специальные информационные структуры, связанные с процессом. Эти указатели в совокупности определяют весь контекст процесса, т.е. всю информацию, необходимую для его исполнения виртуальной машиной.

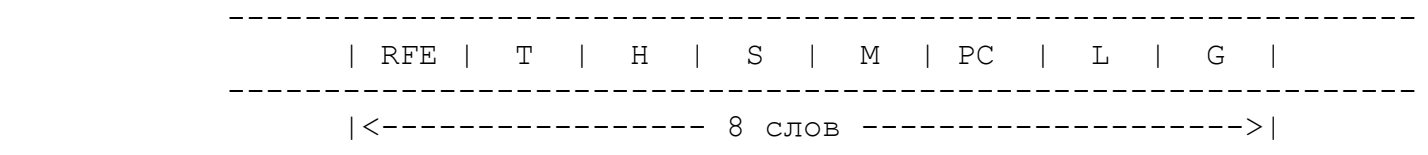


Рис.2. Дескриптор процесса

В терминах Модулы-2 дескриптор процесса представляет собой запись:

```
TYPE Process_Descriptor = RECORD
                            G, L, PC, M, S, H, T, RFE: WORD;
                          END;
```

Каждое из полей дескриптора имеет специальное назначение, описанное ниже. В дальнейшем поля дескриптора процесса для удобства изложения называются регистрами процессора (например, "Process_Descriptor.G" будем называть "G-регистр").

Примечание. Существующие реализации ВМ2М (процессоры

семейства КРОНОС) действительно имеют специализированные регистры, во время работы содержащие копии соответствующих полей дескриптора текущего процесса. При переключении процессов текущее содержимое этих регистров копируется в память по адресу, содержащемуся в P-регистре.

1.4. Память

Память в ВМ2М представляется линейной последовательностью слов размером 32 бита. Каждому слову сопоставлен тридцатиодноразрядный номер - адрес слова. Адресуется только все слово целиком. Адреса бывают только положительные.

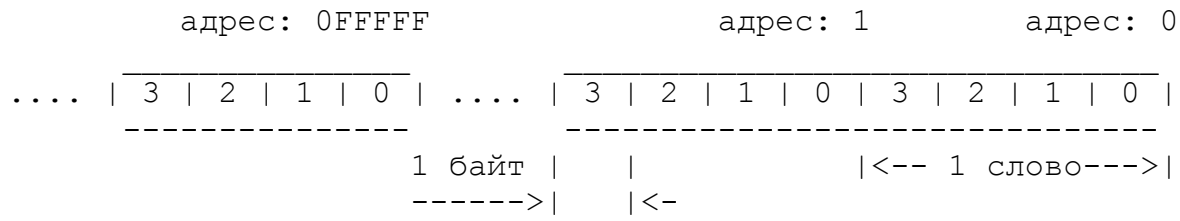


Рис.3. Память ВМ2М.

Обратите внимание, что на всех рисунках по традиции младшие адреса справа, а старшие - слева. Поэтому все изображенные ниже структуры располагаются справа налево.

2. Структуры в памяти

Во время работы ВМ2М в памяти размещены некоторые структуры, устройство и взаимодействие которых описаны в этом разделе.

2.1. Модуль, готовый к исполнению

Каждый из скомпилированных и загруженных в память модулей состоит из сегмента кода, области глобальных данных, области структурных констант, то есть строк, массивов, записей (строковый пул), области связей с внешними модулями (при наличии в модуле импорта объектов).

2.2. Сегмент кода и процедурная таблица

Сегмент кода занимает непрерывный кусок памяти и имеет следующую структуру. В начале сегмента находится процедурная таблица, занимающая до 256 слов. Процедурная таблица устанавливает соответствие между номером процедуры и байтовым смещением от начала сегмента до начала кода соответствующей процедуры (в нулевом слове - смещение 0-й процедуры, в n-м - смещение n-й). Внутри модуля процедура идентифицируется своим номером в процедурной таблице модуля. При исполнении любой процедуры модуля указатель на начало сегмента кода содержится в F-регистре процессора. В другом регистре - PC (Programm Counter)- содержится байтовое смещение от начала сегмента кода до байта, содержащего следующую за исполняемой в данный момент команду.

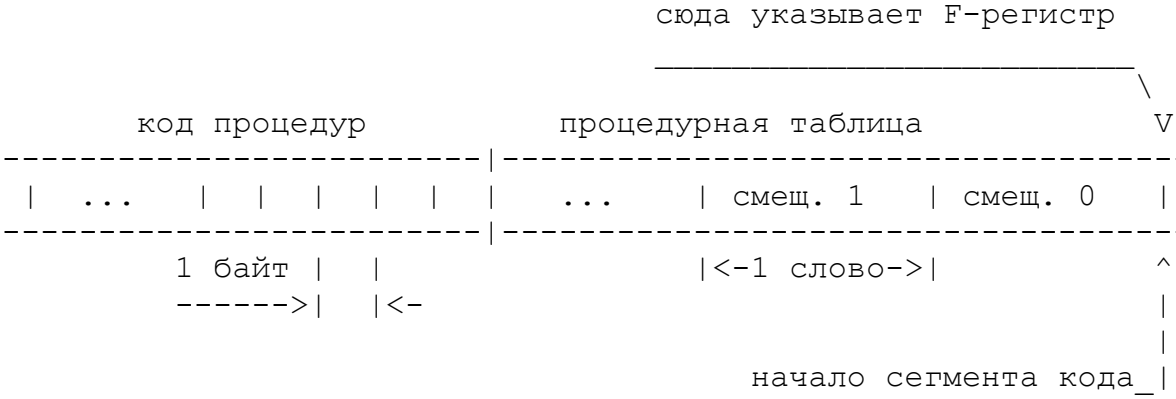


Рис.4. Сегмент кода

2.3. Область глобальных данных

Область глобальных данных одного модуля занимает непрерывный кусок памяти и представляется последовательностью слов. При этом однословные данные хранятся в них непосредственно, а структурные данные могут быть представлены в виде ссылки на специально отведенные для них области памяти.

На начало области глобальных данных текущего модуля указывает G-регистр процессора. Таким образом, доступ к глобальным переменным модуля организуется индексно по содержимому G-регистра, т.е. глобальное слово с номером 3 находится в ячейке памяти с адресом [G]+3 (здесь и далее запись [REG] обозначает содержимое регистра "REG"). Первые два слова области глобальных данных заняты специальной информацией: в нулевом слове области содержится указатель на начало сегмента кода данного модуля (копия F-регистра), в 1-м слове - указатель на начало строкового пула.

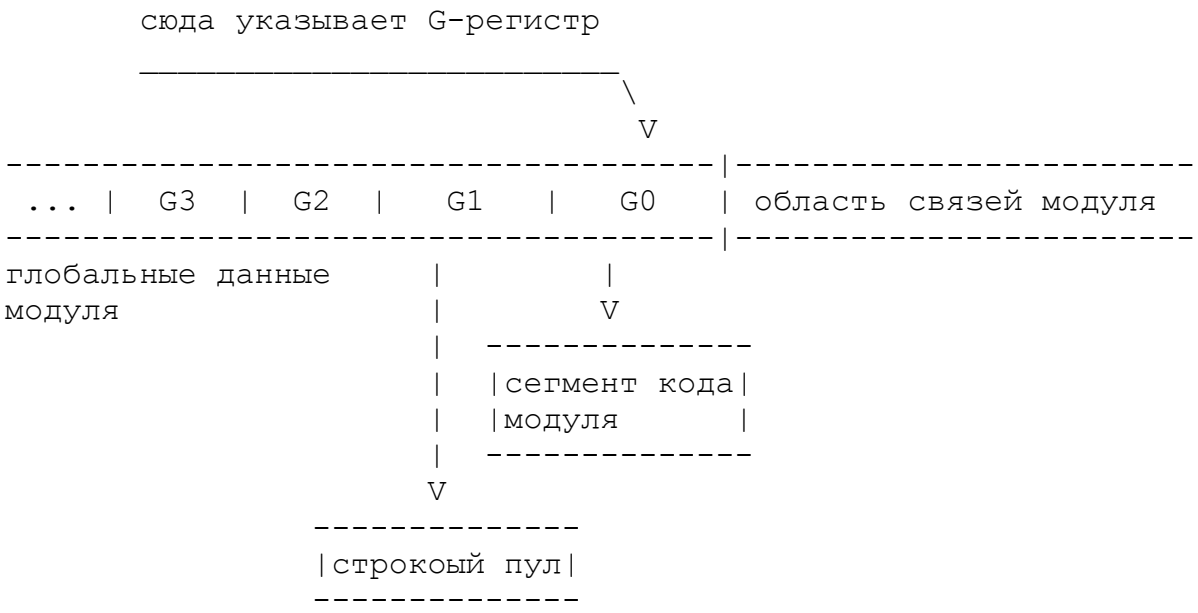


Рис.5. Область глобальных данных модуля. Стрелками изображены указатели.

2.4. Строковый пул

Строковый пул занимает непрерывный кусок памяти и содержит в себе структурные константы (строки, массивы, записи). На начало строкового пула указывает 1-е слово области глобальных данных модуля.

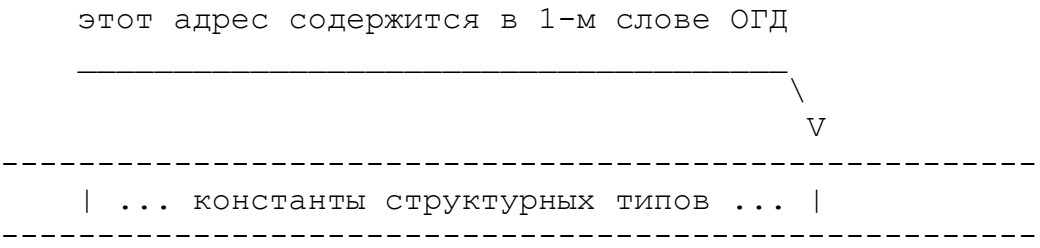


Рис.6. Строковый пул модуля

2.5. Область связей

Область связей с внешними модулями (локальная DFT - Data Frame Table) занимает непрерывный кусок памяти и представляется массивом слов, каждое из которых содержит ссылку на некоторое слово, в котором находится указатель на начало области глобальных данных соответствующего внешнего модуля. Таким образом, внешние модули идентифицируются индексом в локальной DFT модуля, т.е. адрес ссылки на область глобальных данных внешнего модуля с номером i находится в элементе DFT[i]. Сам модуль имеет в локальной DFT номер 0, то есть как бы является сам для себя внешним. Это позволяет, в частности, корректно выполнять вызов формальных процедур, значениями которых являются локальные процедуры этого модуля.

Локальная DFT располагается непосредственно перед областью глобальных данных модуля, т.е. элемент DFT[i] находится по адресу [G]-i-1 памяти. Заметим, что при такой организации информации для доступа к статически существующим внешним объектам (процедурам, переменным), достаточно знать только адрес начала области глобальных данных данного модуля и два смещения.

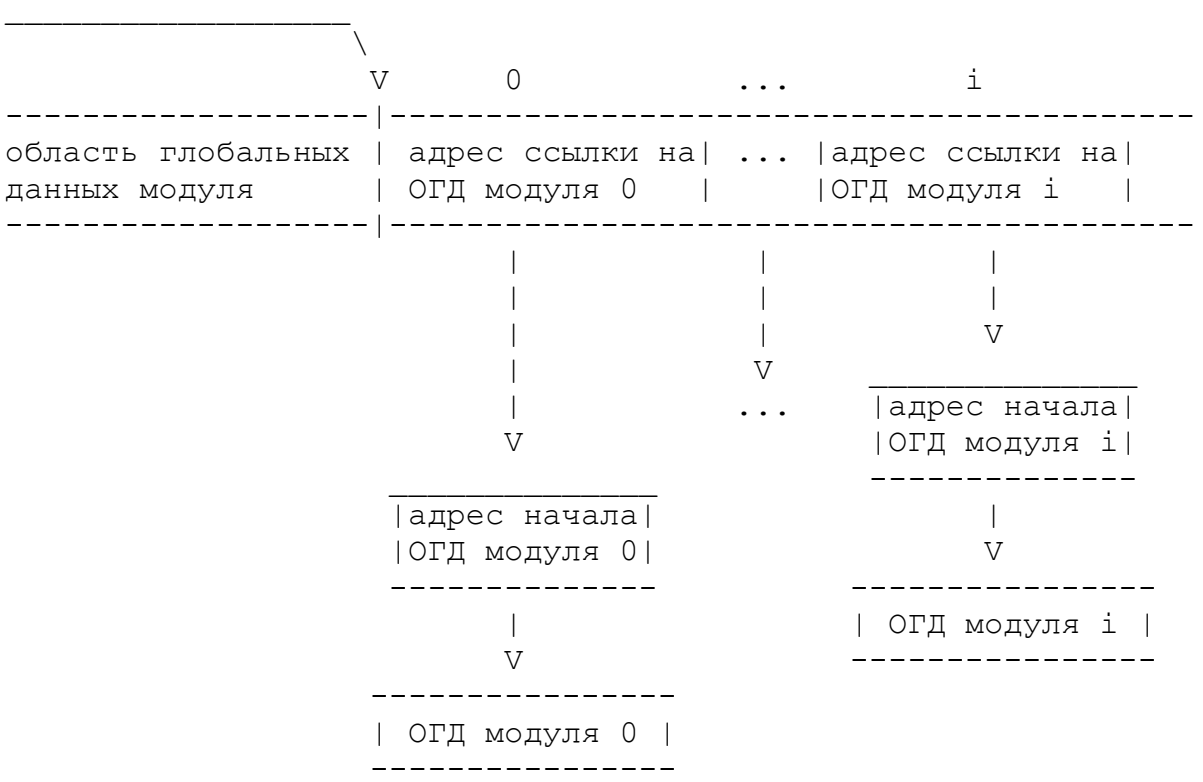


Рис.7. Область связей модуля

2.6. Процедурный стек

Процедурный стек (**P-стек**) занимает непрерывный кусок памяти и используется для размещения **локальных данных процедур**, организации вызовов и возвратов из процедур и размечен тремя регистрами процессора. S-регистр указывает на вершину P-стека, т.е. первое свободное на P-стеке слово. H-регистр указывает на последнее слово области памяти, отведенной под P-стек (предел увеличения [S]). L-регистр указывает на начало области локальных данных процедуры, исполняемой в данный момент. Перекрытие регистров S и H (т.е. ситуация, когда [S] >= [H]) называется переполнением P-стека и отслеживается аппаратно с возбуждением соответствующего прерывания.

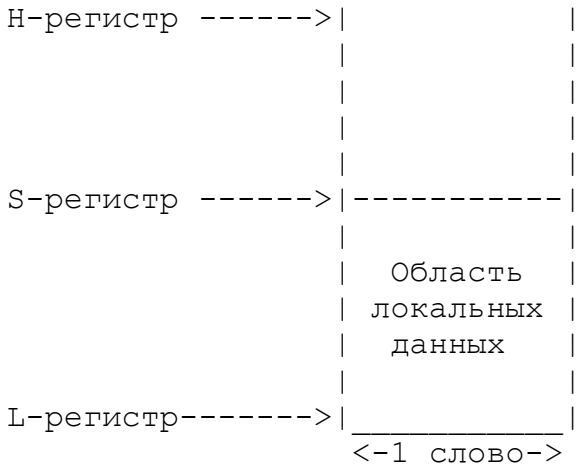


Рис.8. P-стек (процедурный стек) и его разметка. Стрелками изображены указатели.

2.6.1. Область локальных данных

Область **локальных данных** текущей процедуры располагается на P-стеке и представляет собой последовательность слов, содержащих локальные переменные непосредственно или ссылки на начала участков памяти, отведенных для переменных структурных типов. Доступ к локальным данным осуществляется индексно по L-регистру, т.е локальное слово с номером i находится по адресу [L]+i в памяти. Специальные команды облегчают доступ к локальным словам с номерами 4..255.

Первые 4 локальных слова с номерами 0..3 заняты специальной информацией: в нулевом локальном слове содержится указатель на начало области локальных данных объемлющей процедуры (процедуры, внутри которой текущая описана как локальная процедура), или указатель на область глобальных данных вызывающего модуля (если процедура была вызвана из внешнего модуля) - так называемая **статическая цепочка**. 1-е локальное слово указывает на начало области локальных данных процедуры, из которой произошел вызов текущей (**динамическая цепочка**). 2-е локальное слово содержит значение PC возврата. 3-е локальное слово оставлено для возможного дальнейшего использования (RFE - Reserved for Future Extension).

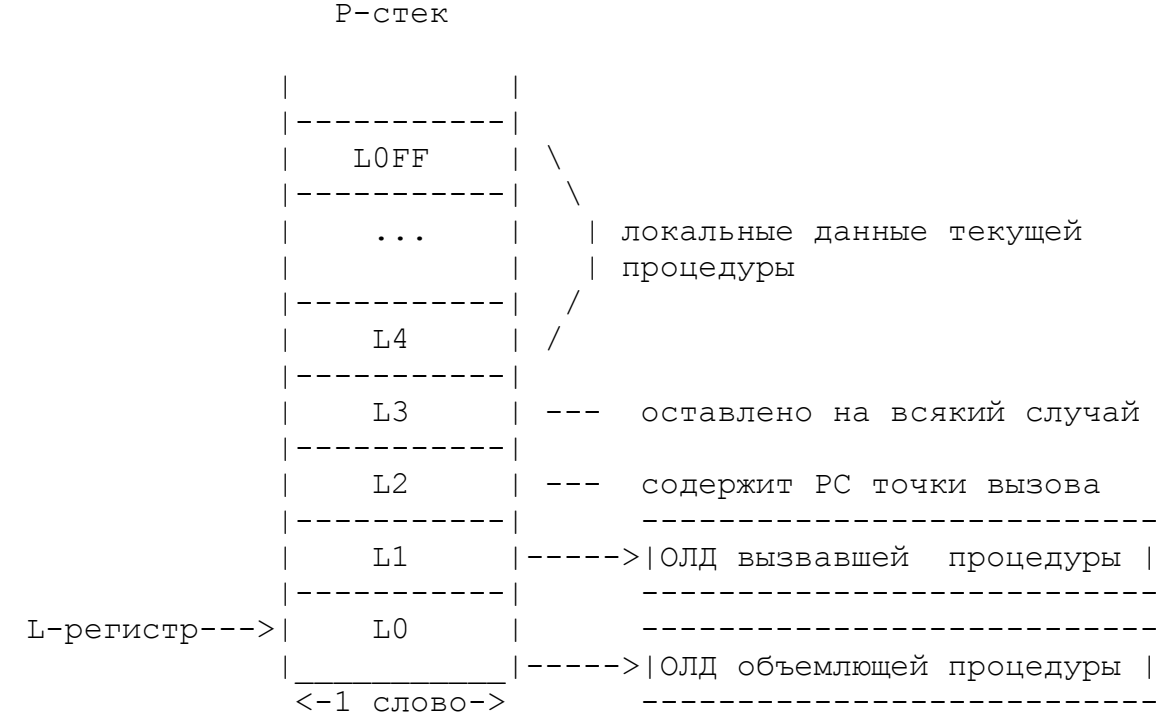


Рис.9. Область локальных данных исполняемой процедуры. Стрелками изображены указатели.

2.7. Еще раз о дескрипторе процесса

Итак, суммируем информацию о дескрипторе процесса. Он занимает непрерывный кусок памяти размером восемь слов, содержащих следующее:

1) G - адрес области глобальных данных;
2) L - адрес локальных данных;
3) PC - байтовое смещение команды, подлежащей исполнению;
4) M - маска прерываний;
5) S - адрес вершины процедурного стека;
6) H - адрес границы процедурного стека;
7) T - причина прерывания;
8) RFE - для дальнейших расширений.

3. M-код

Код BM2M (**M-код**) представляет собой поток байтов, т.е. код команды всегда занимает один байт. Команды BM2M не содержат в себе адресных полей, но по коду команды всегда точно известно, где находятся операнды.

3.1. Способы адресации

Различные команды адресуют данные несколькими способами, которые иллюстрирует таблица.

Способ адресации	Как вычисляется адрес
локальный	[L] + N
глобальный	[G] + N
внешний	DFT[M]^ + N
промежуточный	[L]^+ N или [[L]^]^+ N ...
косвенный	[POP()]^ + N,

где M - номер внешнего модуля, N - индекс объекта, символ "^" обозначает операцию разыменования указателя.

Промежуточный способ адресации предназначен для работы с объектами статически объемлющей процедуры.

3.2. Непосредственные операнды

Команды могут содержать от 4 бит до 4 байт непосредственных операндов. Непосредственные операнды - операнды, следующие в сегменте кода непосредственно за кодом команды, причем по коду операции всегда однозначно известно, сколько байт непосредственных операндов следует за ним. При извлечении данных из потока команд и самих команд счетчик команд (PC) увеличивается на 1..5 байта соответственно.

M-код имеет четыре типа непосредственных операндов:

3.2.1. Полубайтовый операнд: последние 4 бита команды интерпретируются как непосредственный операнд. Так, например, имеются команды

```
    LI0, LI1,...,LI15:
     (*  Load  Immediate  - загрузка непосредственная с кодами
0,1,...,15 и семантикой: *)
    Push(IR MOD 16)
(* где IR - регистр кода команды *)

    SLW4,...,SLW15:
(* Store Local Word *)
    MEM[L + IR MOD 16] := Pop();
(* MEM - физическая или логическая память *)
```

Таким образом, тело процедуры

```
    PROCEDURE p; CONST c=2;
      VAR i: INTEGER;
    BEGIN i:=c;
    END p;
```

реализуется в две следующие однобайтовые команды:

```
    LI2
    SLW4.
```

3.2.2. Байтовый операнд:

Например,

```
    LIB:
(* Load Immediate Byte *)
    Push(Next());
(* Next() - операция выборки одного байта из кода *)
```

При замене описания константы в предыдущем примере на следующее:

```
    c=153
```

породится код

```
    LIB
    153
    SLW4
```

3.2.3. Двухбайтовый операнд - представленный следующей за кодом парой байтов.

3.2.4. Словный операнд - представленный следующей за кодом четверкой байтов.

3.2.5. Существуют команды, имеющие несколько операндов различной длины.

3.3. Функции и назначение команд

Команды BM2M условно подразделяются на следующие группы: арифметико-логические, команды организации управления, команды для работы с A-стеком, вспомогательные.

Команды для работы со стеком содержат инструкции загрузки на стек и записи в память вершины стека.

Все арифметико-логические операции работают над одним или двумя верхними элементами стека и помещают результат на стек.

Команды для организации управления представлены обычными командами переходов, условных и безусловных, специальными командами для организации операторов FOR и CASE, набором инструкций вызова и возврата из процедуры. Сюда же относятся команды TRAP - программного прерывания и TRANSFER - переключения процессов, аналог переключения сопрограмм, дающий мощное средство для организации мультипрограммной работы и синхронизации.

BM2M имеет набор вспомогательных команд, облегчающих обработку мультизначений, параметров процедуры, команды ввода/вывода и др..

4. Прерывания

> Прерывание - прекращение выполнения текущей команды или текущей последовательности команд для обработки некоторого события; событие может быть вызвано командой или сигналом от внешнего устройства. Прерывание позволяет обработать возникшее событие специальной программой и вернуться к прерванной программе.
>
> А.Борковский
> Англо-русский словарь
> по программированию и информатике

В ВМ2М прерывания осуществляются как переключения процессов с занесением в Т-регистр причины прерывания (для прерываний, возбуждаемых процессором и связанных с процессом). Производится переключение с прерванного процесса на процесс обработки прерывания. Обрабатывающий процесс определяется номером прерывания.

4.1. Типы прерываний

Прерывания могут быть вызваны как аппаратными, так и программными средствами. Примером аппаратного прерывания может служить прерывание от таймера, которое производится 50 раз в секунду.

Программными средствами могут быть вызваны такие прерывания, как переполнение целого, выход за границы массива и т.д..

Прерывание полностью определяется своим номером. Перечислим прерывания и их номера:

```
01h    таймер;
02h    останов процессора;
03h    обращение к отсутсвующей памяти;
04h    авария питания;
05h    ошибка процессора;
06h    ошибка ввода вектора прерывания;
07h    нереализованная команда;
08h    по вызову процедуры      (П2.2);
09h    по возврату из процедуры (П2.2);

0Bh    трассировка (прерывание по каждой команде);
       (П2.5,П2.6)

40h    переполнение P-стека (S>H);
41h    переполнение целого или деление на 0;
42h    переполнение вещественного;
43h    исчерпание вещественного;
44h    переполнение адреса (П2.2);
```

49h INVLD команда;
4Ah выход из диапазона;
4Bh неверный параметр команды (аппаратный ASSERT);
4Ch исчерпание или переполнение стека выражений (П2.2).

Если в скобках указан номер модели процессора, то соответствующее прерывание реализовано только для этой модели.

4.2. Вектора прерываний

Каждому прерыванию соответствует пара слов, в первом из которых лежит указатель на дескриптор обрабатывающего процесса, а во втором - адрес слова, в которое должен быть занесен адрес дескриптора прерванного процесса. Эта пара называется вектором прерывания.

4.3. Пространство векторов прерываний

Адрес вектора прерывания может быть вычислен по номеру этого прерывания умножением на 2, с одним исключением. Все прерывания с номером, большим 3Fh, происходят по вектору 3Fh.

Таким образом, множество векторов прерываний занимает непрерывный кусок памяти с абсолютного адреса 2h (прерывания нумеруются с единицы) по адрес 7Fh, и называется пространством векторов прерываний.

4.4. Маска прерываний

Некоторые из прерываний могут быть запрещены. Это делается, например, с целью программной обработки причины прерывания без возбуждения такового. Разрешение прерывания осуществляется выставлением соответствующего бита в маске прерываний. Маска прерываний хранится в М-регистре процессора. При этом выставление 0-го бита разрешает прерывания от внешних устройств, 1-го бита - прерывания от таймера, 31-го бита - программные прерывания (все прерывания с номером, большим или равным 3Fh).

Более полное и подробное представление о ВМ2М можно получить, ознакомившись с программой интерпретатора М-кода, которая также является спецификацией микропрограмм процессоров Кронос.

ЧАСТЬ II. ИНТЕРПРЕТАТОР М-КОДА

```
(*
     00      20      40      60      80      A0      C0      E0

00   LI0     LLW     LXB     LSW0            LSS     MOVE    INCL
01   LI1     LGW     LXW     LSW1    QUIT    LEQ   **CHKNIL  EXCL
02   LI2     LEW     LGW2    LSW2    GETM    GTR     LSTA   *INL
03   LI3     LSW     LGW3    LSW3    SETM    GEQ     COMP   *QUOT
04   LI4     LLW4    LGW4    LSW4    TRAP    EQU     GB      INC1
05   LI5     LLW5    LGW5    LSW5    TRA     NEQ     GB1     DEC1
06   LI6     LLW6    LGW6    LSW6    TR      ABS     CHK     INC
07   LI7     LLW7    LGW7    LSW7    IDLE    NEG     CHKZ    DEC

08   LI8     LLW8    LGW8    LSW8    ADD     OR      ALLOC   STOT
09   LI9     LLW9    LGW9    LSW9    SUB     AND     ENTR    LODT
0A   LI0A    LLW0A   LGW0A   LSW0A   MUL     XOR     RTN     LXA
0B   LI0B    LLW0B   LGW0B   LSW0B   DIV     BIC     NOP     LPC
0C   LI0C    LLW0C   LGW0C   LSW0C   SHL     IN      CX    **BBU
0D   LI0D    LLW0D   LGW0D   LSW0D   SHR     BIT     CI    **BBP
0E   LI0E    LLW0E   LGW0E   LSW0E   ROL     NOT     CF    **BBLT
0F   LI0F    LLW0F   LGW0F   LSW0F   ROR     MOD     CL    **PDX

10   LIB     SLW     SXB     SSW0    IO0     DECS    CL0     SWAP
11   LID     SGW     SXW     SSW1    IO1     DROP    CL1     LPA
12   LIW     SEW     SGW2    SSW2    IO2     LODFV   CL2     LPW
13   LIN     SSW     SGW3    SSW3    IO3     STORE   CL3     SPW
14   LLA     SLW4    SGW4    SSW4    IO4     STOFV   CL4     SSWU
15   LGA     SLW5    SGW5    SSW5   *ARRCMP  COPT    CL5    **RCHK
16   LSA     SLW6    SGW6    SSW6   *WM      CPCOP   CL6    **RCHZ
17   LEA     SLW7    SGW7    SSW7   *BM      PCOP    CL7    **CM

18   JFLC    SLW8    SGW8    SSW8    FADD    *FOR1   CL8     *CHKBX
19   JFL     SLW9    SGW9    SSW9    FSUB    *FOR2   CL9     *BMG
1A   JFSC    SLW0A   SGW0A   SSW0A   FMUL    *ENTC   CL0A    ACTIV
1B   JFS     SLW0B   SGW0B   SSW0B   FDIV    *XIT    CL0B    USR
1C   JBLC    SLW0C   SGW0C   SSW0C   FCMP    ADDPC   CL0C    SYS
1D   JBL     SLW0D   SGW0D   SSW0D   FABS    JMP     CL0D  **NII
1E   JBSC    SLW0E   SGW0E   SSW0E   FNEG    ORJP    CL0E    DOT
1F   JBS     SLW0F   SGW0F   SSW0F   FFCT    ANDJP   CL0F    INVLD

 * -- реализовано только на П2.2.
** -- не реализовано на П2.2.
```

Последнее исправление Oct-1989

```
MODULE Kronos_Interpreter;  (* Leo 27-Nov-85. (c) Kronos *)
                            (* Ned 30-Aug-87. (c) KRONOS *)

(* Этот интерпретатор является спецификацией аппаратуры. *)

FROM SYSTEM    IMPORT  ADDRESS, WORD, ADR;
FROM KRONOS    IMPORT  ROR, ROL, SHR, SHL;

TYPE CPUs = (Kronos2_2, Kronos2_5, Kronos2_6);

VAR cpu: CPUs;

CONST ESdepth = 7; (* глубина стека выражений *)

TYPE
  BYTE     = [0..255];
  WORD16   = [0..0FFFFh];
  PC_Range = WORD16;
  CodePtr  = POINTER TO ARRAY PC_Range OF BYTE;

VAR
  PC:    PC_Range;    (* счетчик команд *)
  IR:    [0..0FFh];   (* регистр команды *)
  F :    CodePtr;     (* адрес сегмента кода *)
  G :    ADDRESS;     (* адрес сегмента глобальных данных *)
  L :    ADDRESS;     (* адрес сегмента локальных  данных *)
  S :    ADDRESS;     (* адрес вершины  P-стека *)
  H :    ADDRESS;     (* граница P-стека *)
  P :    ADDRESS;     (* адрес дескриптора процесса *)
  M :    BITSET;      (* маска прерываний *)
  Ipt:   BOOLEAN;     (* запрос на прерывание *)
  IptNo: WORD16;      (* номер прерывания *)


CONST  (* номера битов в слове L2 (см. рис.) *)
  ExternalBit   = 1Fh;
  ChangeMaskBit = 1Eh;

CONST
  NonVectBit = 1Fh;
(* бит, маскирующий программные прерывания *)

(* Замечание.  Регистр  H  задает заниженную на ESdepth+1 слов
   границу   стека.   Это   необходимо   для   сохранения   стека
   выражений при переключении процессов (см. SaveExpStack).
*)

VAR MEM: ARRAY ADDRESS OF WORD;
    ByteMEM: ARRAY OF BYTE; (* Наложен на MEM *)

MODULE InstructionFetch;

  IMPORT F, PC, WORD, WORD16, BYTE, MEM, ADDRESS;
  EXPORT Next, Next2, Next4, GetPc;
```

```
  PROCEDURE Next(): BYTE;
  BEGIN INC(PC); RETURN INTEGER(F^[PC-1])
  END Next;

  PROCEDURE Next2(): WORD16;
  BEGIN RETURN Next()+Next()*100h
  END Next2;

  PROCEDURE Next4(): WORD;
  BEGIN RETURN Next2()+Next2()*10000h
  END Next4;

  PROCEDURE GetPc(procno: INTEGER): INTEGER;
  (* Выдает PC начала процедуры *)
  BEGIN RETURN MEM[ADDRESS(F)+procno]
  END GetPc;

END InstructionFetch;

MODULE Mask;
  IMPORT M, NonVectBit, BYTE;
  EXPORT NotMasked;

  PROCEDURE NotMasked(N: BYTE): BOOLEAN;
  BEGIN
    IF    (N>=0Fh) & (N<3Fh)  THEN RETURN (0 IN M)
    ELSIF (N< 0Fh) & (N>0)    THEN RETURN (0 IN M) & (N IN M)
    ELSIF (N =3Fh)            THEN RETURN (NonVectBit IN M)
    ELSE ASSERT(FALSE)
    END
  END NotMasked;

END Mask;

MODULE ExpressionStack;
  IMPORT WORD, ESdepth, Ipt, IptNo;
  EXPORT Push, Pop, Empty;

  VAR A: ARRAY [0..ESdepth-1] OF WORD;  sp: [0..ESdepth];

  PROCEDURE Push(X: WORD);
  BEGIN A[sp]:=X;
    IF sp<ESdepth THEN INC(sp) ELSE Ipt:=TRUE; IptNo:=4Ch END;
  END Push;

  PROCEDURE Pop(): INTEGER;
  BEGIN
    IF sp=0 THEN Ipt:=TRUE; IptNo:=4Ch ELSE DEC(sp) END;
    RETURN A[sp];
  END Pop;

  PROCEDURE Empty(): BOOLEAN;
  BEGIN RETURN sp=0 END Empty;
```

```
BEGIN sp:=0 END ExpressionStack;

MODULE ProcessSupport;
  IMPORT PC,G,F,H,L,S,P,M, MEM, NotMasked, ESdepth
       , CodePtr, WORD16, ADDRESS;
  FROM ExpressionStack IMPORT Pop, Push, Empty;

  EXPORT SaveExpStack, RestoreExpStack, Transfer, TRAP;

  PROCEDURE SaveExpStack;
    VAR c: CARDINAL; (* счетчик глубины стека *)
  BEGIN c:=0;
    WHILE NOT Empty() DO MEM[S]:=Pop(); INC(S); INC(c) END;
    MEM[S]:=c; INC(S);
  END SaveExpStack;

  PROCEDURE RestoreExpStack;
    VAR c: CARDINAL; (* счетчик глубины стека *)
  BEGIN DEC(S); c:=MEM[S];
    WHILE c>0 DO DEC(c); DEC(S); Push(MEM[S]) END;
  END RestoreExpStack;

  PROCEDURE SaveRegs;
  BEGIN SaveExpStack;
    MEM[P+0]:=G;  MEM[P+1]:=L;
    MEM[P+2]:=PC; MEM[P+3]:=CARDINAL(M);
    MEM[P+4]:=S;  MEM[P+5]:=H+ESdepth+1;
  END SaveRegs;

  PROCEDURE RestoreRegs;
  BEGIN
    G:=MEM[P+0];    F :=CodePtr(MEM[G]);
    L:=MEM[P+1];    PC:=MEM[P+2];  M:=BITSET(MEM[P+3]);
    S:=MEM[P+4];    H:=MEM[P+5]-ESdepth-1;
    RestoreExpStack;
  END RestoreRegs;

  PROCEDURE Transfer(pFrom,pTo: ADDRESS);
    VAR j: CARDINAL;
  BEGIN (* заметьте - pFrom может быть равно pTo *)
    j:=MEM[pTo]; SaveRegs; MEM[pFrom]:=P; MEM[1]:=P;
    P:=j;         RestoreRegs;              MEM[0]:=P;
  END Transfer;

  PROCEDURE TRAP(N: WORD16);
  BEGIN MEM[P+6]:=N;
    IF N>3Fh THEN N:=3Fh END;
    IF NotMasked(N) THEN Transfer(N*2,MEM[N*2+1]) END;
  END TRAP;

END ProcessSupport;

(* Маркировка P-стека перед вызовом процедуры *)
```

```
PROCEDURE Mark(X: ADDRESS; External: BOOLEAN);
  VAR i: ADDRESS;
BEGIN i:=S;
  MEM[S]:=X; INC(S); (* статическая  цепочка *)
  MEM[S]:=L; INC(S); (* динамическая цепочка *)
  IF External THEN MEM[S]:=WORD(BITSET(PC)+{ExternalBit})
  ELSE              MEM[S]:=PC
  END; INC(S,2); L:=i;
END Mark;

PROCEDURE ioP2_2;
BEGIN
  ASSERT(IR-90h IN {0..7});
(* см. документацию на П2.2 *)
END ioP2_2;

PROCEDURE ioP2_5;
BEGIN
  ASSERT(IR-90h IN {0..7});
(* Запросы  на  В/В  передаются другим процессорам через общую
   память. См. LABTAM 3000 manuals.
*)
END ioP2_5;

PROCEDURE ioP2_6;
BEGIN ASSERT(IR-90h IN {0..7});
(* Смотря на какой шине. *)
END ioP2_6;

(* Рабочие переменные интерпретатора: *)

VAR i,j,k: CARDINAL;   X,Y  : REAL;
    v,w  : BITSET;     a,b  : CHAR;
    adr,adr1,sz,hi,low: CARDINAL;
    src,trg: INTEGER;
    dyn: POINTER TO
           RECORD
             adr : ADDRESS;
             high: INTEGER;
           END;

PROCEDURE ConsolMicroProgram;
BEGIN
(* Пультовая микропрограмма позволяет произвести начальную
   загрузку  программы  и  по команде оператора "Go" выполняет
   Transfer(0,1).
*)
END ConsolMicroProgram;

PROCEDURE Interpret;
BEGIN
  CASE IR OF
   00h..0Fh:(* LI0..LI0F  Load Immediate *) Push(IR MOD 10h);
```

```
|10h: (* LIB  Load Immediate Byte *) Push(Next())
|11h: (* LID  Load Immediate Double byte *) Push(Next2())
|12h: (* LIW  Load Immediate Word *) Push(Next4())
|13h: (* LIN  Load Immediate NIL  *) Push(NIL)
|14h: (* LLA  Load Local  Address *) Push(L+Next())
|15h: (* LGA  Load Global Address *) Push(G+Next())
|16h: (* LSA  Load Stack  Address *) Push(Pop()+Next())
|17h: (* LEA  Load External Addres *)
      i:=G-Next()-1; (* индекс в DFT модуля.*)
      adr:=MEM[i]; (* указатель на элемент большой DFT *)
      Push(MEM[adr]+Next())
|18h: (* JLFC Jump Long  Forward Condition *)
      IF Pop()=0 THEN PC:=Next2()+PC
      ELSE INC(PC,2) END
|19h: (* JLF  Jump Long  Forward *) PC:=Next2()+PC;
|1Ah: (* JSFC Jump Short Forward Condition *)
      IF Pop()=0 THEN PC:=Next()+PC
      ELSE INC(PC) END
|1Bh: (* JSF  Jump Short Forward *) PC:=Next()+PC;
|1Ch: (* JLBC Jump Long  Back    Condition *)
      IF Pop()=0 THEN PC:=-Next2()+PC
      ELSE INC(PC,2) END
|1Dh: (* JLB  Jump Long  Back    *) PC:=-Next2()+PC;
|1Eh: (* JSBC Jump Short Back    Condition *)
      IF Pop()=0 THEN PC:=-Next()+PC
      ELSE INC(PC) END
|1Fh: (* JSB  Jump Short Back    *) PC:=-Next()+PC;
|20h: (* LLW  Load Local  Word *)  Push(MEM[L+Next()])
|21h: (* LGW  Load Global Word *)  Push(MEM[G+Next()])
|22h: (* LEW  Load External Word *)
      i:=G-Next()-1; adr:=MEM[MEM[i]]; (* external G *)
      Push(MEM[adr+Next()])
|23h: (* LSW  Load Stack addressed Word *)
       Push(MEM[Pop()+Next()])
|24h..2Fh: (* LLW0..LLW0F Load Local Word *)
      Push(MEM[L+IR MOD 10h])
|30h: (* SLW  Store Local  Word *)  MEM[L+Next()]:=Pop()
|31h: (* SLW  Store Global Word *)  MEM[G+Next()]:=Pop()
|32h: (* SEW  Store External Word *)
      i:=G-Next()-1; adr:=MEM[MEM[i]]; (* external G *)
      MEM[adr+Next()]:=Pop()
|33h: (* SSW  Store Stack addressed Word *)
      i:=Pop(); MEM[Pop()+Next()]:=i
|34h..3Fh: (* SLW0..SLW0F  Store Local Word *)
      MEM[L+IR MOD 10h]:=Pop()

|40h: (* LXB  Load Indexed Byte *)
      i:=Pop(); Push(ByteMEM[Pop()*4+i]);
|41h: (* LXW  Load Indexed Word *)
      i:=Pop(); Push(MEM[Pop()+i])
|42h..4Fh: (* LGW02..LGW0F  Load Global Word *)
      Push(MEM[G+IR MOD 10h])
|50h: (* SXB  Store Indexed Byte *)
```

```
        j:=Pop(); i:=Pop(); ByteMEM[Pop()*4+i]:=j;
  |51h: (* SXW  Store Indexed Word *)
        j:=Pop(); i:=Pop(); MEM[Pop()+i]:=j
  |52h..5Fh: (* SGW02..SGW0F  Store Global Word *)
        MEM[G+IR MOD 10h]:=Pop()

  |60h..6Fh: (* LSW00..LSW0F  Load  Stack addressed Word *)
        Push(MEM[Pop()+IR MOD 10h])
  |70h..7Fh: (* SSW00..SSW0F  Store Stack addressed Word *)
        i:=Pop(); MEM[Pop()+IR MOD 10h]:=i

  |80h: TRAP(7h);
  |81h: (* QUIT   Stop processor *) ConsolMicroProgram
  |82h: (* GETM   Get Mask *) Push(M)
  |83h: (* SETM   Set Mask *) M:=BITSET(Pop);
  |84h: (* TRAP interrupt simulation *) TRAP(Pop())
  |85h: (* TRA   Transfer control between process *)
        i:=Pop(); Transfer(Pop(),i)
  |86h: (* TR    Test & Reset *)
        i:=Pop(); Push(MEM[i]); MEM[i]:=0
  |87h: (* IDLE  IDLE process *)
        DEC(PC); REPEAT (* не занимая шины *) UNTIL Ipt
(* В   следующих   шести   командах,   а   также   в   командах
   MOD,NEG,ABS,FOR2,INC,DEC,INC1,DEC1  в  случае  переполнения
   возбуждается прерывание с IptNo=41h.
*)
  |88h: (* ADD  integer ADD *) Push(Pop()+Pop())
  |89h: (* SUB  integer SUB *) i:=Pop(); Push(Pop()-i);
  |8Ah: (* MUL  integer MUL *) Push(Pop()*Pop())
  |8Bh: (* DIV  integer DIV *) i:=Pop(); Push(Pop() DIV i)
  |8Ch: (* SHL  integer SHift Left *)
        i:=Pop(); Push(SHL(Pop(),i))
  |8Dh: (* SHR  integer SHift Right *)
        i:=Pop(); Push(SHR(Pop(),i))

  |8Eh: (* ROL  word ROtate Left  *)
        i:=Pop() MOD 20h; Push(ROL(Pop(),i))
  |8Fh: (* ROR  word ROtate Right *)
        i:=Pop() MOD 20h; Push(ROR(Pop(),i))
  |90h..94h: (* io section *)
        CASE cpu OF
          |Kronos2_2: ioP2_2
          |Kronos2_5: ioP2_5
          |Kronos2_6: ioP2_6
        ELSE
        END

  |95h: (* ARRCMP array compare *)
        sz:=Pop(); adr:=Pop(); adr1:=Pop();
        IF sz<0 THEN Push(sz); TRAP(4Fh)
        ELSIF sz=0 THEN Push(adr1); Push(adr1)
        ELSE LOOP
               IF (adr^ # adr1^) OR (sz=1) THEN
                   Push(adr1); Push(adr); EXIT
```

```
            END;
            DEC(sz); INC(adr); INC(adr1);
          END;
      END;

  |96h: (* WM     word move *):
      sz:=Pop(); f:=Pop(); t:=Pop();
      IF t>f THEN
        t:=t+sz-1; f:=f+sz-1;
        WHILE sz>0 DO
          MEM[t]:=MEM[f]; DEC(t); DEC(f); DEC(sz)
        END
      ELSE
        WHILE sz>0 DO
          MEM[t]:=MEM[f]; INC(t); INC(f); DEC(sz)
        END
      END;

  |97h: (* BM     bit  move *)
      sz:=Pop(); -- размер пересылаемой области в битах
      i:=Pop(); src:=Pop(); -- смещение и адрес источника
      j:=Pop(); trg:=Pop(); -- смещение и адрес приемника
      src:=src*32+i;
      trg:=trg*32+j;
      IF src>=trg THEN                    n:=+1
      ELSE INC(src,sz-1); INC(trg,sz-1); n:=-1
      END;
      FOR k:=0 TO sz-1 DO
        i:=trg DIV 32; j:=trg MOD 32;
        IF (src MOD 32) IN BITSET(MEM[src DIV 32]) THEN
          MEM[i]:=INTEGER( BITSET(MEM[i]) + {j} )
        ELSE
          MEM[i]:=INTEGER( BITSET(MEM[i]) - {j} )
        END;
        INC(src,n); INC(trg,n)
      END

(* В следующих восьми командах (за исключением FCMP) при
   исчезновении порядка или переполнении возбуждаются
   прерывания с IptNo=42h или 43h соответственно.
*)
  |98h: (* FADD  Float ADD *) Push(REAL(Pop())+REAL(Pop()))
  |99h: (* FSUB  Float SUB *)
      X:=REAL(Pop()); Push(REAL(Pop())-X)
  |9Ah: (* FMUL  Float MUL *) Push(REAL(Pop())*REAL(Pop()))
  |9Bh: (* FDIV  Float DIV *)
      X:=REAL(Pop()); Push(REAL(Pop())/X)
  |9Ch: (* FCMP  Float CoMPare *)
      X:=REAL(Pop()); Y:=REAL(Pop());
      IF    X<Y THEN Push(1); Push(0)
      ELSIF X>Y THEN Push(0); Push(1)
      ELSE Push(0); Push(0) END
  |9Dh: (* FABS  Float ABS *) X:=REAL(Pop());
      IF X<0.0 THEN Push(-X) ELSE Push(X) END
```

```
 |9Eh: (* FNEG  Float NEG *) Push(-REAL(Pop()))
 |9Fh: (* FFCT  Float FunCTions *) i:=Next();
       IF    i=0 THEN Push(FLOAT(INTEGER(Pop())))
       ELSIF i=1 THEN Push(TRUNC(   REAL(Pop())))
       ELSE DEC(PC); TRAP(7h)
       END;

|0A0h: (* LSS  int LeSS             *) i:=Pop(); Push(Pop()<i)
|0A1h: (* LEQ  int Less    or EQual *) i:=Pop(); Push(Pop()<=i)
|0A2h: (* GTR  int GreaTeR          *) i:=Pop(); Push(Pop()>i)
|0A3h: (* GEQ  int Greater or EQual *) i:=Pop(); Push(Pop()>=i)
|0A4h: (* EQU  int EQUal     *) Push(Pop()=Pop())
|0A5h: (* NEQ  int Not EQual *) Push(Pop()#Pop())
|0A6h: (* ABS  int ABSolute value *) Push(ABS(Pop()))
|0A7h: (* NEG  int NEGate         *) Push(-Pop())

|0A8h: (* OR   logical bit per bit OR *)
       v:=BITSET(Pop()); w:=BITSET(Pop()); Push(w+v)
|0A9h: (* AND  logical bit per bit AND *)
       v:=BITSET(Pop()); w:=BITSET(Pop()); Push(w*v)
|0AAh: (* XOR  logical bit per bit XOR *)
       v:=BITSET(Pop()); w:=BITSET(Pop()); Push(w/v)
|0ABh: (* BIC  logical bit per bit BIt Clear *)
       v:=BITSET(Pop()); w:=BITSET(Pop()); Push(w-v)
|0ACh: (* IN   membership to bitset *)
       v:=BITSET(Pop()); Push(Pop() IN v)
|0ADh: (* BIT  setBIT *) i:=Pop();
       IF (i<0) OR (i>=20h) THEN TRAP(4Ah)
       ELSE w:={}; INCL(w,i); Push(w) END

|0AEh: (* NOT  boolean NOT (not bit per bit!) *) Push(Pop()=0)
|0AFh: (* MOD  integer MODulo *) i:=Pop(); Push(Pop() MOD i)

|0B0h: (* DECS  DECriment S register (reverse to ALLOC) *)
       DEC(S,Pop())
|0B1h: (* DROP *) i:=Pop();
|0B2h: (* LODF  reLOaD expr. stack after Function return *)
       i:=Pop(); RestoreExpStack; Push(i)
|0B3h: (* STORE STORE expr. stack before function call *)
       IF S+ESdepth+1>H THEN DEC(PC); TRAP(40h)
       ELSE SaveExpStack
       END
|0B4h: (* STOFV STOre expr. stack with Formal function Value
          on top before function call (см. команду CF)
        *)
       IF S+ESdepth+2>H THEN DEC(PC); TRAP(40h)
       ELSE i:=Pop(); SaveExpStack; MEM[S]:=i; INC(S) END
|0B5h: (* COPT  COPy Top of expr. stack *)
       i:=Pop(); Push(i); Push(i)
|0B6h: (* CPCOP Character array Parameter COPy *)
       i:=Pop(); (* High *) sz:=(i+4) DIV 4;
       IF S+sz>H THEN Push(i); DEC(PC); TRAP(40h)
       ELSE MEM[L+Next()]:=S; adr:=Pop();
         WHILE sz>0 DO MEM[S]:=MEM[adr]; INC(S); INC(adr) END
```

```
        END
 |0B7h: (* PCOP  structure Parameter allocate and COPy *)
        i:=Pop(); (* High *) sz:=i+1;
        IF S+sz>H THEN Push(i); DEC(PC); TRAP(40h)
        ELSE MEM[L+Next()]:=S; adr:=Pop();
          WHILE sz>0 DO MEM[S]:=MEM[adr]; INC(S); INC(adr) END
        END
 |0B8h: (* FOR1  enter  FOR statment *)
        IF S+2>H THEN DEC(PC); TRAP(40h)
        ELSE sz:=Next(); (* =0 up; #0 down *)
          hi:=Pop(); low:=Pop(); adr:=Pop(); k:=Next2()+PC;
          IF ((sz=0) & (low<=hi)) OR ((sz#0) & (low>=hi)) THEN
            MEM[adr]:=low;
            MEM[S]:=adr; INC(S); MEM[S]:=hi; INC(S);
          ELSE (* цикл не исполняется не разу *) PC:=k
          END
        END
 |0B9h: (* FOR2  end of FOR statment *)
        hi:=MEM[S-1]; adr:=MEM[S-2]; sz:=Next();
        IF sz>7Fh THEN sz:=7Fh-sz END; (* шаг [-128..127] *)
        k:=-Next2()+PC;  i:=MEM[adr]+sz;
        IF ((sz>=0) & (i>hi)) OR ((sz<0) & (i<hi)) THEN
          DEC(S,2); (* terminate *)
        ELSE MEM[adr]:=i; PC:=k (* continue *)
        END
 |0BAh: (* ENTC  ENTer Case statment *)
        IF S+1>H THEN DEC(PC); TRAP(40h)
        ELSE PC:=Next2()+PC; (* jump to case table *)
          k:=Pop(); low:=Next2(); hi:=Next2();
          MEM[S]:=PC + 2*(hi-low) + 4; INC(S);
          (* PC для выхода *)
          IF (k>=low) & (k<=hi) THEN
            PC:=PC+2*(k-low+1) (* jump into case table *)
          END;
          PC:=-Next2()+PC (* jump back to variant's code *)
        END
 |0BBh: (* XIT  eXIT from case or control structure *)
        DEC(S); PC:=MEM[S]
 |0BCh: (* ADDPC  add to program counter *)
        Push(Pop()+PC);
 |0BDh: (* JMP *)
        PC:=Pop();
 |0BEh: (* ORJP   short circuit OR  JumP *)
        IF Pop()#0 THEN Push(1); PC:=Next()+PC
        ELSE INC(PC) END
 |0BFh: (* ANDJP  short circuit AND JumP *)
        IF Pop()=0 THEN Push(0); PC:=Next()+PC
        ELSE INC(PC)
        END

 |0C0h: (* MOVE   MOVE block *) sz:=Pop();
        i:=Pop(); j:=Pop();
        WHILE sz>0 DO
          MEM[j]:=MEM[i]; INC(i); INC(j); DEC(sz)
```

```
        END
 |0C1h: (* CHKNIL check address for NIL *)
        i:=Pop(); Push(i);
        IF i=NIL THEN DEC(PC); TRAP(41h) END
 |0C2h: (* LSTA  Load STring Address *)
        Push(MEM[G+1]+Next2());
 |0C3h: (* COMP  COMPare strings *) i:=Pop()*4; j:=Pop()*4;
        REPEAT a:=CHAR(ByteMEM[i]); b:=CHAR(ByteMEM[j]);
        INC(i); INC(j)
        UNTIL (a=0c) OR (b=0c) OR (a#b); Push(a); Push(b)
 |0C4h: (* GB  Get procedure Base n level down *)
        i:=L; k:=Next();
        WHILE k>0 DO i:=MEM[i]; DEC(k) END; Push(i)
 |0C5h: (* GB1 :=Pop(); i:=Pop(); Push(i);
 |0C6h: (* CHK  array boundary CHecK *)
        hi:=Pop(); low:=Pop(); i:=Pop(); Push(i);
        IF (i<low) OR (i>hi) THEN
          Push(low); Push(hi); TRAP(4Ah)
        END
 |0C7h: (* CHKZ  array boundary CHecK (low=Zero) *)
        hi:=Pop(); i:=Pop(); Push(i);
        IF (i<0) OR (i>hi) THEN Push(hi); TRAP(4Ah) END
 |0C8h: (* ALLOC ALLOCate block *) sz:=Pop();
        IF S+sz>H THEN  Push(sz); DEC(PC); TRAP(40h)
        ELSE Push(S); INC(S,sz) END
 |0C9h: (* ENTR  ENTeR procedure *) sz:=Next();
        IF S+sz>H THEN DEC(PC,2); TRAP(40h)
        ELSE INC(S,sz) END
 |0CAh: (* RTN   ReTurN from procedure *)
        S:=L; L:=MEM[S+1];
        PC:=WORD(BITSET(MEM[S+2])*{0..0Fh});
        IF ExternalBit IN BITSET(MEM[S+2]) THEN
          (* external called *)
          G:=MEM[S]; F:=CodePtr(MEM[G])
        END;
 |0CBh: (* NOP   No OPeration  *)
 |0CCh: (* CX    Call eXternal *)
        IF S+4<=H THEN j:=MEM[G-Next()-1];
          i:=Next(); Mark(G,TRUE);
          G:=MEM[j]; F:=CodePtr(MEM[G]); PC:=GetPc(i);
        ELSE DEC(PC); TRAP(40h) END
 |0CDh: (* CI    Call procedure at Intermediate level *)
        IF S+4<=H THEN
          i:=Next(); Mark(Pop(),FALSE); PC:=GetPc(i);
        ELSE DEC(PC); TRAP(40h) END
 |0CEh: (* CF    Call Formal procedure *)
        IF S+3<=H THEN i:=MEM[S-1]; DEC(S); Mark(G,TRUE);
          k:=i DIV 1000000h; i:=i MOD 1000000h;
          G:=MEM[i]; F:=CodePtr(MEM[G]); PC:=GetPc(k);
        ELSE DEC(PC); TRAP(40h) END
 |0CFh: (* CL    Call Local procedure *)
        IF S+4<=H THEN i:=Next(); Mark(L,FALSE); PC:=GetPc(i);
        ELSE DEC(PC); TRAP(40h) END
 |0D0h..0DFh: (* CL0..CL0F  Call Local procedure *)
```

```
        IF S+4<=H THEN Mark(L,FALSE); PC:=GetPc(IR MOD 10h);
        ELSE DEC(PC); TRAP(40h) END
 |0E0h: (* INCL  INCLude in set *)
        i:=Pop(); j:=Pop() + i DIV 32;
        MEM[j]:=INTEGER( BITSET(MEM[j]) + {i MOD 32} );
 |0E1h: (* EXCL  EXCLude from set *)
        i:=Pop();
        j:=Pop() + i DIV 32;
        MEM[j]:=INTEGER( BITSET(MEM[j]) - {i MOD 32} );
 |0E2h: (* INL  membership IN Long set *)
        k:=Pop(); j:=Pop(); i:=Pop();
        IF (i<0) OR (i>=k) THEN Push(0)
        ELSE
          Push( (i MOD 32) IN BITSET(MEM[j+i DIV 32]) );
        END;
 |0E3h: (* QUOT QUOTient commands *)
        i:=Pop(); j:=Pop();
        CASE Next() OF
          |0: Push( j SHRQ i ); -- деление на степень двойки
          |1: Push( j QOU  i ); -- деление с округлением к нулю
          |2: Push( j ANDQ i ); -- модуль по степени двойки
          |3: Push( j REM  i ); -- модуль
        ELSE TRAP(7); DEC(PC,2)
        END;
 |0E4h: (* INC1  INCrement by 1 *) INC(MEM[Pop()])
 |0E5h: (* DEC1  DECrement by 1 *) DEC(MEM[Pop()])
 |0E6h: (* INC   INCrement *) i:=Pop(); INC(MEM[Pop()],i)
 |0E7h: (* DEC   DECrement *) i:=Pop(); DEC(MEM[Pop()],i)
 |0E8h: (* STOT  STOre Top on proc stack *)
        IF S+1>H THEN DEC(PC); TRAP(40h)
        ELSE MEM[S]:=Pop(); INC(S)
        END
 |0E9h: (* LODT  LOaD   Top of proc stack *)
        DEC(S); Push(MEM[S])
 |0EAh: (* LXA   Load indeXed Address *)
        sz:=Pop(); i:=Pop(); adr:=Pop(); Push(adr+i*sz)
 |0EBh: (* LPC   Load Procedure Constant *)
        i:=Next(); j:=Next(); Push(j*1000000h+MEM[G-i-1])

(* Следующие  3  команды  работают  с вырезками битов. Битовый
   адрес - это пара (адрес,битовое смещение). Битовое смещение
   может  быть  больше  32.  Вырезка  может переходить границы
   слова.
*)
 |0ECh: (* BBU  Bit Block Unpack *)
        sz:=Pop();
        IF (sz<1) OR (sz>32) THEN
          Push(sz); DEC(PC); TRAP(4Ah)
        END;
        i:=Pop(); adr:=Pop();
        (* j:=битовая вырезка длиной sz, начиная с
           битового адреса (adr,i)
        *)
        Push(j);
```

```
|0EDh: (* BBP  Bit Block Pack *)
       j:=Pop(); sz:=Pop();
       IF (sz<1) OR (sz>32) THEN
         Push(sz); DEC(PC); TRAP(4Ah)
       END;
       i:=Pop(); adr:=Pop();
       (* Упаковывает sz младших битов из j по битовому
          адресу (adr,i)
       *)
|0EEh: (* BBLT Bit BLock Transfer *)
       sz:=Pop(); (* Длина может любой, больше 0 *)
       i:=Pop(); adr:=Pop();
       j:=Pop(); adr1:=Pop();
       (* Переслать биты (adr,i) -> (adr1,j) длиной sz *)
       (* Куски не должны перекрываться!!! *)
|0EFh: (* PDX Prepare Dynamic indeX *)
       i:=Pop();
       dyn:=Pop();
       Push(dyn^.adr); Push(i);
       IF (i<0) OR (i>dyn^.high)
       THEN TRAP(4Ah)
       END;
|0F0h: (* SWAP *)
       i:=Pop(); j:=Pop(); Push(i); Push(j)
|0F1h: (* LPA Load Parameter Address *)
       Push(L-Next()-1);
|0F2h: (* LPW Load Parameter WORD *)
       Push(MEM[L-Next()-1]);
|0F3h: (* SPW Store Parameter WORD *)
       MEM[L-Next()-1]:=Pop();
|0F4h: (* SSWU Store Stack Word Undestructive *)
       i:=Pop(); MEM[Pop()]:=i; Push(i)
|0F5h: (* RCHK  range CHecK *)
       hi:=Pop(); low:=Pop(); i:=Pop(); Push(i);
       Push( (low<=i) & (i<=hi) );
|0F6h: (* RCHZ range check (low=Zero) *)
       hi:=Pop(); i:=Pop(); Push(i);
       Push( (0<=i) & (i<=hi) );
|0F7h: (* CM Call procedure from dynamic Module *)
       IF S+4<=H THEN
         i:=Next();
         DEC(S); j:=MEM[S];
         Mark(G,TRUE);
         G:=j; F:=CodePtr(MEM[G]); PC:=GetPc(i);
       ELSE DEC(PC); TRAP(40h)
       END;
|0F8h: (* CHKBX  CHecK BoX *)
       p0:=Pop(); p1:=Pop();
       x00:=INTEGER(p0^)     MOD 10000h;
       y00:=INTEGER(p0^<<16) MOD 10000h;
       INC(p0);
       x01:=INTEGER(p0^)     MOD 10000h;
       y01:=INTEGER(p0^<<16) MOD 10000h;
       x10:=INTEGER(p1^)     MOD 10000h;
```

```
        y10:=INTEGER(p1^<<16) MOD 10000h;
        INC(p1);
        x11:=INTEGER(p1^)     MOD 10000h;
        y11:=INTEGER(p1^<<16) MOD 10000h;
        Push((x10<=x01) & (y10<=y01) & (x00<=x11) & (y00<=y11));
 |0F9h: (* BMG Bit Map Graphic commands *)
        CASE Next() OF
          |0: IN_RECT -- IN RECTangle?
          |1: DVL     -- Display Vertical Lines
          |2: BBLT_G  -- BBLT-loGic
          |3: DCH     -- Display CHar
          |4: CLP     -- CLiPping
          |5: DLN     -- Display  LiNe
          |6: CRC     -- CiRCus
          |7: ARC     -- ARC
          |8: TRF     -- TRiangle Filling
          |9: CRF     -- CiRcus  Filling
        ELSE
          Trap(49h)
        END
 |0FAh: (* ACTIVe process *) Push(P)
 |0FBh: (* USR User defined functions *) i:=Next(); (*  *)
 |0FCh: (* SYS SYStem rarely functions *)
        CASE Next() OF
         |00h: (* PID Processor IDent {2,5,6} *)
               (* Push(PID) *)
               (* Остальные могут быть различными в разных
моделях *)
         |02h: (* PM  Processor Model *)
        ELSE TRAP(7h)
        END;
 |0FDh: (* NII Never Implemented Instruction *) TRAP(7h);
 |0FFh: (* INVLD INVaLiD Operation *) TRAP(49h)
  ELSE  (* unimplemented instruction *) TRAP(7h)
  END (*CASE*)
END Interpret;

BEGIN (* "Power On" *)
 (* cpu:=Kronos2_2 | Kronos2_5 | Kronos2_6 *)
 ORIGIN(ByteMEM,ADR(MEM),SIZE(MEM));
 ConsolMicroProgram;
 LOOP
   IF Ipt THEN TRAP(IptNo) END;
   IR:=Next();
   Interpret;
 END (*LOOP*)
END Kronos_Interpreter.
```

ЧАСТЬ III. ОПИСАНИЕ СИСТЕМЫ КОМАНД

Данное описание не является самостоятельным документом. Можно рассматривать его как подробный и пространный комментарий к интерпретатору системы команд процессоров КРОНОС. Здесь опущены определения основных понятий архитектуры семейства КРОНОС, их можно найти в разделе "Виртуальная Модула-2 машина".

Код команды

Исполнение любой команды начинается с выборки ее кода. Код команды имеет размер 8 бит. Он интерпретируется как номер команды и полностью определяет алгоритм исполнения команды. В нем нет никаких полей или способов адресации. Код команды - это только байт и больше ничего.

Операнды

После выборки кода команды определен алгоритм ее исполнения. Некоторые операнды команд располагаются на стеке выражений (A-стек), другие могут следовать непосредственно за кодом команды, как последовательность байтов. Последние называются непосредственными операндами. Они никогда не интерпретируются как абсолютные адреса.

Выборка кода

Выборка кода производится с помощью процедур Next, Next2, Next4 (см. Интерпретатор системы команд). Эти процедуры выбирают из кода байт, два байта и слово, увеличивая, соответственно, счетчик команд PC на один, два или четыре. В терминах Модулы-2 эти процедуры выглядят так:

```
PROCEDURE Next(): BYTE;
BEGIN INC(PC); RETURN INTEGER(F^[PC-1])
END Next;

PROCEDURE Next2(): WORD16;
BEGIN RETURN Next()+Next()*100h
END Next2;

PROCEDURE Next4(): WORD;
BEGIN RETURN Next2()+Next2()*10000h
END Next4;
```

Стек выражений

Стек выражений используется для вычисления выражений и

хранения аргументов команд и результатов их выполнения. Работа со стеком определяется процедурами Pop (взять со стека) и Push (положить на стек).

```
PROCEDURE Push(X: WORD);
BEGIN A[sp]:=X;
  IF sp<ESdepth THEN INC(sp) ELSE Ipt:=TRUE; IptNo:=4Ch END;
END Push;

PROCEDURE Pop(): INTEGER;
BEGIN
  IF sp=0 THEN Ipt:=TRUE; IptNo:=4Ch ELSE DEC(sp) END;
  RETURN A[sp];
END Pop;
```

Везде в дальнейшем под словами "взять со стека", "брать со стека" следует понимать действия:

1) проверяется, не пуст ли стек выражений; если пуст, то возбуждается прерывание с номером 4Ch;

2) иначе с вершины стека выражений считывается слово и счетчик стека уменьшается на единицу; таким образом, слово оказывается счеркнутым.

Под словами "поместить на стек", "загрузить на стек" следует понимать действия:

1) элементу стека выражений, на который указывает счетчик, присваивается то значение, которое загружается;

2) если счетчик меньше границы стека выражений, то он увеличивается на единицу, иначе вызывается прерывание с номером 4Ch.

Если не указано, о каком стеке идет речь, то имеется в виду стек выражений.

4-битовые непосредственные операнды

Поскольку система команд Кроноса очень маленькая (менее 128 команд), было сочтено возможным использовать 4 младших бита в некоторых часто исполняемых командах для 4-битового непосредственного операнда. Такая команда может быть описана либо как множество из 16-ти различных команд (см. "Интерпретатор"), либо как просто 4-битовая команда с четырьмя битами непосредственного операнда.

Заметим, что все числа, встречающиеся далее - шестнадцатеричные (и поэтому снабжены символом 'h' - 'hex').

LI0..LI0F **00h..0Fh**

Load Immediate

Код операции: 4 бита 0h
Непосредственные операнды: 4 бита 0h..0Fh
Длина команды: 1 байт
Действие:

Загружает на стек значение, соответствующее 4-м младшим битам кода команды (из отрезка 00h..0Fh), в виде 32-битового слова с нулями в старших 28 битах (ведущие нули).

PC увеличивается на 1.

В терминах интерпретатора:

```
Push(IR MOD 10h);
```

LIB **10h**

Load Immediate Byte

Код операции: 1 байт 10h
Непосредственные операнды: 1 байт 0h..0FFh
Длина команды: 2 байта
Действие:

Загружает значение непосредственного операнда из диапазона 00h..0FFh (1 байта кода, следующих за командой) и помещает его на стек в виде 32-битового слова с ведущими нулями.

PC увеличивается на 2.

В терминах интерпретатора:

```
Push(Next())
```

LID **11h**

Load Immediate Double Byte

Код операции: 1 байт 11h
Непосредственные операнды: 2 байта 0h..0FFFFh
Длина команды: 3 байта
Действие:

Загружает значение непосредственного операнда из диапазона 00h..0FFFFh (2 байта кода, следующих за командой) и помещает его на стек в виде 32-битового слова с ведущими нулями.

PC увеличивается на 3.

В терминах интерпретатора:

```
Push(Next2())
```

LIW **12h**

Load Immediate Word

Код операции: 1 байт 12h
Непосредственные операнды: 4 байта 0h..0FFFFFFFFh
Длина команды: 5 байт
Действие:

Загружает значение непосредственного операнда из диапазона 00h..0FFFFFFFFh (4 байта кода, следующих за командой) и помещает его на стек в виде 32-битового слова.

PC увеличивается на 5.

Замечание. Числа представляются в двоично-дополнительном коде. Диапазон целых:
-2**31 <= x <= 2**31-1,
или, в шестнадцатеричном представлении,
80000000h <= x <= 7FFFFFFFh.
"-1" представляется как 0FFFFFFFFh.

Замечание. Только команда LIW позволяет положить на стек отрицательное число. Небольшие отрицательные числа порождаются по-другому: загружаются их абсолютные величины и командой NEG изменяется знак.

В терминах интерпретатора:
Push(Next4())

LIN **13h**
Load Immediate Nil
Код операции: 1 байт 13h
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:
Загружает на стек несуществующий адрес (NIL).
PC увеличивается на 1.

Замечание. Значение NIL зависит от модели процессора. Это не влияет на переносимость программного обеспечения. Так, например, в Кронос 2.2 NIL равен FFFFFh. В этой модели этот адрес проверяется аппаратно, в остальных моделях может быть проверен явно командой CHKNIL.

В терминах интерпретатора:
Push(NIL)

LLA **14h**
Load Local Address
Код операции: 1 байт 14h
Непосредственные операнды: 1 байт 0h..0FFh
Длина команды: 2 байта
Действие:
Прибавляет к содержимому L-регистра непосредственный операнд (байт, следующий за кодом команды) и помещает полученную сумму на стек. PC увеличивается на 2. Эта команда позволяет загрузить на стек адрес локальной переменной. Если смещение переменной относительно L-регистра больше 0FFh, задача решается комбинированием команд LLA, LI, LIB, LID или LIW с ADD.

В терминах интерпретатора:
Push(L+Next())

LGA **15h**
Load Global Address
Код операции: 1 байт 15h

Непосредственные операнды: 1 байт 0h..0FFh
Длина команды: 2 байта
Действие:

Прибавляет к содержимому G-регистра непосредственный операнд (байт, следующий за кодом команды) и помещает полученную сумму на стек. PC увеличивается на 2. Эта команда позволяет загрузить на стек адрес глобальной переменной. Если смещение переменной относительно G-регистра больше 0FFh, задача решается комбинированием команд LGA, LI, LIB, LID или LIW с ADD.

В терминах интерпретатора:

```
Push(G+Next())
```

LSA **16h**

Load Stack Address
Код операции: 1 байт 16h
Непосредственные операнды: 1 байт 0h..0FFh
Длина команды: 2 байта
Действие:

Берет слово со стека, прибавляет к нему непосредственный операнд (байт, следующий за кодом команды), и помещает полученную сумму на стек. PC увеличивается на 2. Эта команда позволяет заменить адрес, лежащий на стеке, на адрес, смещенный относительно него. Это используется для получения адресов полей в тех случаях, когда адрес записи уже загружен на стек. Когда смещение поля больше 0FFh, задача решается комбинированием команд LSA, LI, LIB, LID или LIW с ADD.

Замечание. LSA XX семантически эквивалентно последовательности LIB XX ADD.

В терминах интерпретатора:

```
Push(Pop()+Next())
```

LEA **17h**

Load External Address
Код операции: 1 байт 17h
Непосредственные операнды: 2 байта 0h..0FFh
Длина команды: 3 байта
Действие:

Выбирается однобайтовый непосредственный операнд, который интерпретируется как номер внешнего модуля, по которому в DFT модуля ищется ссылка на адрес ОГД внешнего модуля. Слово из ОГД внешнего модуля со смещением, взятым из следующего однобайтового непосредственного операнда, загружается на стек.

PC увеличивается на 3.

В терминах интерпретатора:

```
i:=G-Next()-1;
adr:=MEM[i];
Push(MEM[adr]+Next())
```

JFLC **18h**

Jump Forward Long Condition

Код операции:	1 байт	18h
Непосредственные операнды:	2 байта	0h..0FFh
Длина команды:	3 байта	

Действие:

Выбирает двухбайтовый непосредственный операнд - размер (N) возможного перехода к следующей команде. PC увеличивается на 3. Со стека берется значение. Если это 0, то PC увеличивается на N, иначе (при любом не равном нулю значении) далее будет выполняться следующая команда.

В терминах интерпретатора:

```
IF Pop()=0 THEN PC:=Next2()+PC
ELSE INC(PC,2)
END
```

JFL **19h**

Jump Forward Long

Код операции:	1 байт	19h
Непосредственные операнды:	2 байта	0h..0FFFFh
Длина команды:	3 байта	

Действие:

Выбирается двухбайтовый непосредственный операнд - размер (N) перехода к следующей команде. PC увеличивается на 3+N.

В терминах интерпретатора:

```
PC:=Next2()+PC;
```

JFSC **1Ah**

Jump Forward Short Condition

Код операции:	1 байт	1Ah
Непосредственные операнды:	1 байт	0h..0FFh
Длина команды:	2 байта	

Действие:

Выбирает однобайтовый непосредственный операнд - размер (N) возможного перехода к следующей команде. PC увеличивается на 2. Со стека берется значение. Если это 0, то PC увеличивается на N, иначе (при любом ненулевом значении) далее будет выполняться следующая команда.

В терминах интерпретатора:

```
IF Pop()=0 THEN PC:=Next()+PC
ELSE INC(PC)
END
```

JFS **1Bh**

Jump Forward Short

Код операции:	1 байт	1Bh
Непосредственные операнды:	1 байт	0h..0FFh
Длина команды:	2 байта	

Действие:

Выбирается однобайтовый непосредственный операнд - размер (N) перехода к следующей команде. PC увеличивается на 2+N.

В терминах интерпретатора:

```
PC:=Next()+PC;
```

JBLC **1Ch**

Jump Back Long Condition

Код операции:	1 байт	1Ch
Непосредственные операнды:	2 байта	0h..0FFFFh
Длина команды:	3 байта	

Действие:

PC увеличивается на 3.

При условии, что значение, взятое со стека, равно нулю, PC уменьшается на столько, сколько указано в двухбайтовом непосредственном операнде. Иначе выполняется следующая команда.

В терминах интерпретатора:

```
IF Pop()=0 THEN PC:=-Next2()+PC
ELSE INC(PC,2) END
```

JBL **1Dh**

Jump Back Long

Код операции:	1 байт	1Dh
Непосредственные операнды:	2 байта	0h..0FFFFh
Длина команды:	3 байта	

Действие:

PC увеличивается на 3 (в результате выборки кода и двухбайтового непосредственного операнда), после чего уменьшается на столько, сколько указано в непосредственном операнде.

В терминах интерпретатора:

```
PC:=-Next2()+PC;
```

JBSC **1Eh**

Jump Back Short Condition

Код операции:	1 байт	1Eh
Непосредственные операнды:	1 байт	0h..0FFh
Длина команды:	2 байта	

Действие:

PC увеличивается на 2 в результате выборки команды и непосредственного операнда.

При условии, что значение, взятое со стека, равно нулю, PC уменьшается на столько, сколько указано в однобайтовом непосредственном операнде. Иначе выполняется следующая команда.

В терминах интерпретатора:

```
IF Pop()=0 THEN PC:=-Next()+PC
ELSE INC(PC) END
```

JBS **1Fh**

Jump Back Short

Код операции:	1 байт	1Fh

Непосредственные операнды: 1 байт 0h..0FFh
Длина команды: 2 байта
Действие:

PC увеличивается на 2 в результате выборки кода и однобайтового непосредственного операнда, после чего уменьшается на столько, сколько указано в непосредственном операнде.

В терминах интерпретатора:

```
PC:=-Next()+PC;
```

LLW **20h**

Load Local Word
Код операции: 1 байт 20h
Непосредственные операнды: 1 байт 0h..0FFh
Длина команды: 2 байта
Действие:

Загружает на стек слово, взятое по адресу, смещенному относительно L-регистра на величину, равную значению непосредственного операнда. команда позволяет загрузить на стек значение локальной переменной. Если смещение переменной относительно L-регистра больше 0FFh, задача решается комбинированием команд LLA, LIW, LSW с ADD.

PC увеличивается на 2.

В терминах интерпретатора:

```
Push(MEM[L+Next()])
```

LGW **21h**

Load Global Word
Код операции: 1 байт 21h
Непосредственные операнды: 1 байт 0h..0FFh
Длина команды: 2 байта
Действие:

Загружает на стек слово, взятое по адресу, смещенному относительно G-регистра на величину, равную значению непосредственного операнда. Эта команда позволяет загрузить на стек значение глобальной переменной. Если смещение переменной относительно G-регистра больше 0FFh, задача решается комбинированием команд LGA, LI, LIB, LID или LIW с ADD и LSW.

PC увеличивается на 2.

В терминах интерпретатора:

```
Push(MEM[G+Next()])
```

LEW **22h**

Load External Word
Код операции: 1 байт 22h
Непосредственные операнды: 2 байта 0h..0FFh
Длина команды: 3 байта
Действие:

Из сегмента кода выбираются два операнда. Первый соответствует номеру внешнего модуля, второй - смещению относительно G-регистра этого модуля. На стек загружается

внешнее глобальное слово из указанного модуля с указанным смещением.

PC увеличивается на 3.

В терминах интерпретатора:

```
i:=G-Next()-1; adr:=MEM[MEM[i]];
Push(MEM[adr+Next()])
```

LSW **23h**

Load Stack addressed Word

Код операции:	1 байт	23h
Непосредственные операнды:	1 байт	0h..0FFh
Длина команды:	2 байта	

Действие:

Загружает на стек слово с адресом, смещенным относительно значения, взятого со стека на величину, равную значению непосредственного операнда.

PC увеличивается на 2.

Замечание. Эта команда может быть использована для доступа к первым 256 словам записи (RECORD) при компиляции с языков высокого уровня.

В терминах интерпретатора:

```
Push(MEM[Pop()+Next()])
```

LLW4..LLW0F **24h..2Fh**

Load Local Word

Код операции:	4 бита	2h
Непосредственные операнды:	4 бита	4h..0Fh
Длина команды:	1 байт	

Действие:

Загружает на стек локальное слово со смещением в диапазоне от 4h до 0Fh относительно L-регистра.

PC увеличивается на 1.

Замечание. Первые четыре слова локальной области содержат специальную информацию, поэтому команд LLW0..LLW3 нет.

Замечание. Данные команды являются, по существу, более компактной версией команды LLW. Такое упрощение оправдано исключительно частым обращением к первым 12-ти локальным переменным процедур.

В терминах интерпретатора:

```
Push(MEM[L+IR MOD 10h])
```

SLW **30h**

Store Local Word

Код операции:	1 байт	30h
Непосредственные операнды:	1 байт	0h..0FFh
Длина команды:	2 байта	

Действие:

Присваивает значение, взятое со стека, локальному слову

со смещением, указанным в непосредственном операнде.
PC увеличивается на 2.

В терминах интерпретатора:
```
MEM[L+Next()]:=Pop()
```

SGW **31h**

Store Global Word

Код операции:	1 байт	31h
Непосредственные операнды:	1 байт	0h..0FFh
Длина команды:	2 байта	

Действие:
Присваивает значение, взятое со стека, глобальному слову со смещением, указанным в непосредственном операнде.
PC увеличивается на 2.

В терминах интерпретатора:
```
MEM[G+Next()]:=Pop()
```

SEW **32h**

Store External Word

Код операции:	1 байт	32h
Непосредственные операнды:	2 байта	0h..0FFh
Длина команды:	3 байта	

Действие:
Из кода выбираются два операнда - номер внешнего модуля и смещение относительно его G-регистра. Значение, взятое со стека, присваивается внешнему глобальному слову из указанного модуля с указанным смещением.
PC увеличивается на 3.

В терминах интерпретатора:
```
i:=G-Next()-1; adr:=MEM[MEM[i]]; (* external G *)
MEM[adr+Next()]:=Pop()
```

SSW **33h**

Store Stack addressed Word

Код операции:	1 байт	33h
Непосредственные операнды:	1 байт	0h..0FFh
Длина команды:	2 байта	

Действие:
Со стека берутся значение и адрес. Присваивает указанное значение слову, смещенному относительно указанного адреса на величину, равную значению непосредственного операнда.
PC увеличивается на 2.

В терминах интерпретатора:
```
i:=Pop(); MEM[Pop()+Next()]:=i
```

SLW4..SLW0F **34h..3Fh**

Store Local Word

Код операции:	4 бита	3h
Непосредственные операнды:	4 бита	4h..0Fh
Длина команды:	1 байт	

Действие:
Присваивает значение, взятое со стека, локальному слову со смещением в диапазоне от 4h до 0Fh.
PC увеличивается на 1.

В терминах интерпретатора:
MEM[L+IR MOD 10h]:=Pop()

LXB **40h**
Load indeXed Byte
Код операции: 1 байт 40h
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:
На стеке - два операнда. Нижний - словный адрес, верхний - байтовое смещение относительно этого адреса. Оба операнда счеркиваются со стека, и на стек загружается байт, взятый из памяти по указанному словному адресу с указанным байтовым смещением, дополненный нулями в старших разрядах.
PC увеличивается на 1.

В терминах интерпретатора:
i:=Pop(); Push(ByteMEM[Pop()*4+i]);

LXW **41h**
Load indeXed Word
Код операции: 1 байт 41h
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:
На стеке - два операнда. Нижний - словный адрес, верхний - словное смещение относительно этого адреса. Оба операнда счеркиваются со стека, и на стек загружается слово, взятое из памяти по указанному словному адресу с указанным смещением.
PC увеличивается на 1.

В терминах интерпретатора:
i:=Pop(); Push(MEM[Pop()+i])

LGW2..LGW0F **42h..4Fh**
Load Global Word
Код операции: 4 бита 4h
Непосредственные операнды: 4 бита 2h..0Fh
Длина команды: 1 байт
Действие:
Загружает на стек глобальное слово со смещением из диапазона от 2h до 0Fh.
PC увеличивается на 1.

В терминах интерпретатора:
Push(MEM[G+IR MOD 10h])

SXB **50h**
Store indeXed Byte

Код операции: 1 байт 50h
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

На стеке - три операнда. Нижний - словный адрес, выше - байтовое смещение относительно этого адреса. Верхний операнд - слово, старшие 24 бита которого игнорируются, а оставшийся байт записывается в память по указанному словному адресу с указанным байтовым смещением. Все три операнда счеркиваются со стека.

PC увеличивается на 1.

В терминах интерпретатора:

```
j:=Pop(); i:=Pop(); ByteMEM[Pop()*4+i]:=j;
```

SXW **51h**

Store indeXed Word

Код операции: 1 байт 51h
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

На стеке - три операнда. Нижний - словный адрес, выше - словное смещение относительно этого адреса. Верхний операнд - слово, которое записывается в память по указанному словному адресу с указанным смещением. Все три операнда счеркиваются со стека.

PC увеличивается на 1.

В терминах интерпретатора:

```
j:=Pop(); i:=Pop(); MEM[Pop()+i]:=j
```

SGW2..SGW0F **52h..5Fh**

Store Global Word

Код операции: 4 бита 5h
Непосредственные операнды: 4 бита 2h..0Fh
Длина команды: 1 байт
Действие:

Присваивает значение, взятое со стека, глобальному слову со смещением из диапазона от 2h до 0Fh.

PC увеличивается на 1.

В терминах интерпретатора:

```
MEM[G+IR MOD 10h]:=Pop()
```

LSW0..LSW0F **60h..6Fh**

Load Stack addressed Word

Код операции: 4 бита 6h
Непосредственные операнды: 4 бита 0h..0Fh
Длина команды: 1 байт
Действие:

Загружает на стек из памяти слово со смещением из диапазона 0..0Fh относительно адреса, взятого со стека.

PC увеличивается на 1.

В терминах интерпретатора:
    Push(MEM[Pop()+IR MOD 10h])

SSW0..SSW0F **70h..7Fh**
Store Stack addressed Word
Код операции: 4 бита 7h
Непосредственные операнды: 4 бита 0h..0Fh
Длина команды: 1 байт
Действие:
    Слово, взятое со стека, записывает в память по адресу, смещенному на 0..0Fh относительно адреса, взятого со стека.
    PC увеличивается на 1.

В терминах интерпретатора:
    i:=Pop(); MEM[Pop()+IR MOD 10h]:=i

QUIT **81h**
stop processor
Код операции: 1 байт 81h
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:
    Останов процессора.
    PC увеличивается на 1 байт.

В терминах интерпретатора:
    ConsolMicroProgram

GETM **82h**
GET Mask
Код операции: 1 байт 82h
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:
    Загружает на стек слово, содержащее маску прерываний (M-регистр процессора).
    PC увеличивается на 1 байт.

В терминах интерпретатора:
    Push(M)

SETM **83h**
SET Mask
Код операции: 1 байт 83h
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:
    Заносит маску прерываний, взятую со стека, в M-регистр процессора.
    PC увеличивается на 1.

В терминах интерпретатора:
    M:=BITSET(Pop());

TRAP **84h**

interrupt simulation

Код операции: 1 байт 84h
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Берет значение со стека и возбуждает прерывание с таким номером.

PC увеличивается на 1.

В терминах интерпретатора:

```
TRAP(Pop())
```

TRA **85h**

TRAnsfer control between processes

Код операции: 1 байт 85h
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Происходит переключение двух процессов.

Состояние процесса определяется состоянием стека и значениями шести регистров (G, L, PC, M, S, H).

На стеке - два слова. Верхнее содержит адрес указателя на дескриптор процесса, который требуется запустить, нижнее - адрес, по которому нужно записать адрес указателя на дескриптор текущего процесса.

Регистры текущего процесса записываются в дескриптор процесса, который лежит в памяти по адресу, указанному в регистре P.

P-регистр текущего процесса записывается по адресу, взятому в качестве второго операнда со стека.

Стек выражений спасается на процедурный стек текущего процесса. Для того, чтобы при спасении стека выражений не возникло переполнения процедурного стека, регистр границы процедурного стека - H - указывает на (глубину стека выражений + 1) слов ниже фактической границы процедурного стека.

В верхнем операнде содержится адрес слова, в котором лежит указатель на дескриптор процесса (P-регистр), подлежащего запуску. По новому P-регистру восстанавливаются значения всех регистров процесса. С вершины процедурного стека восстанавливается стек выражений.

PC увеличивается на 1.

В терминах интерпретатора:

```
i:=Pop(); Transfer(Pop(),i)
```

TR **86h**

Test & Reset

Код операции: 1 байт 86h
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Загружает на стек слово, взятое из памяти по адресу, взятому со стека. По этому адресу помещается 0.

PC увеличивается на 1.

Замечание. Команда выполняется как неделимое действие "чтение с разрушением". В случае работы нескольких процессоров с общей памятью используется для межпроцессорной синхронизации (как, например, в Кронос-2.5).

В терминах интерпретатора:

```
i:=Pop(); Push(MEM[i]); MEM[i]:=0
```

IDLE **87h**

IDLE process

Код операции:	1 байт	87h
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

Уменьшает PC на 1. Процессор ждет прерывания, не занимая шину.

Поскольку при чтении байта кода PC увеличивался на единицу, после выполнения команды IDLE PC остается на прежнем месте.

Замечание. Любой процесс, выполнивший команду IDLE, будет продолжать ее выполнение вечно, пока какой-либо другой процесс (обработчик прерываний, например) принудительно не увеличит PC процесса, выполнявшего команду IDLE.

В терминах интерпретатора:

```
DEC(PC); REPEAT (* не занимая шины *) UNTIL Ipt
```

ADD **88h**

integer ADDition

Код операции:	1 байт	88h
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

Берет со стека два целых числа, складывает их и помещает сумму на стек. Если при этом происходит переполнение целого, возбуждается прерывание с номером 41h.

PC увеличивается на 1.

Замечание. В процессоре П2.6 после возникновения прерывания по переполнению (независимо от того, разрешено оно или нет) на стек помещаются 32 младших бита результата арифметической операции.

В терминах интерпретатора:

```
Push(Pop()+Pop())
```

SUB **89h**

integer SUBtraction

Код операции:	1 байт	89h

Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Берет со стека два операнда - сначала вычитаемое, затем уменьшаемое. Производит вычитание и помещает результат на стек. Если произошло переполнение целого, возбуждается прерывание с номером 41h.

PC увеличивается на 1.

В терминах интерпретатора:

```
i:=Pop(); Push(Pop()-i);
```

MUL **8Ah**

integer MULtiplication
Код операции: 1 байт 8Ah
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Берет со стека два целых числа, перемножает их и помещает произведение на стек. Если в результате произошло переполнение целого, возбуждается прерывание с номером 41h.

PC увеличивается на 1.

В терминах интерпретатора:

```
Push(Pop()*Pop())
```

DIV **8Bh**

integer DIVision
Код операции: 1 байт 8Bh
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Берет со стека два целых числа - сначала делитель, затем делимое, производит деление нацело и помещает результат на стек. Округляет в сторону меньшего. Так, (-1 DIV 2)=-1. Если в результате произошло переполнение целого, возбуждается прерывание с номером 41h.

PC увеличивается на 1.

Замечание. В Кроносе-2.2 округление происходит в сторону нуля.

В терминах интерпретатора:

```
i:=Pop(); Push(Pop() DIV i)
```

SHL **8Ch**

integer SHift Left
Код операции: 1 байт 8Ch
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Арифметический сдвиг влево. Со стека берутся величина сдвига и сдвигаемое слово. Слово сдвигается влево на указанную величину. В младшие разряды дописывается соответствующее число нулей. При несовпадении знаковых разрядов сдвигаемого слова и

результата возбуждается прерывание с номером 41h (переполнение целого). Результат помещается на стек.

PC увеличивается на 1.

Замечание. Таким образом, команда SHL выполняет умножение числа со знаком на степень двойки.

В терминах интерпретатора:

```
i:=Pop(); Push(SHL(Pop(),i))
```

SHR **8Dh**

integer SHift Right

Код операции:	1 байт	8Dh
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

Арифметический сдвиг вправо. Со стека берутся величина сдвига и сдвигаемое слово. Слово сдвигается вправо на величину сдвига. Содержимое знакового разряда восстанавливается, старшие разряды заполняются содержимым знакового разряда. Результат помещается на стек.

PC увеличивается на 1.

Замечание. Таким образом, команда SHR выполняет деление числа со знаком на степень двойки с округлением в сторону меньшего.

В терминах интерпретатора:

```
i:=Pop(); Push(SHR(Pop(),i))
```

ROL **8Eh**

word ROtate Left

Код операции:	1 байт	8Eh
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

Циклический сдвиг влево.

При циклическом сдвиге влево на единицу старший разряд слова переносится вправо и становится его младшим разрядом. При циклическом сдвиге на N эта операция повторяется N раз.

Со стека берутся величина сдвига и сдвигаемое слово. Слово сдвигается влево на число разрядов, равное величине сдвига по модулю 32. Результат помещается на стек.

PC увеличивается на 1.

В терминах интерпретатора:

```
i:=Pop() MOD 20h; Push(ROL(Pop(),i))
```

ROR **8Fh**

word ROtate Right

Код операции:	1 байт	8Fh
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

Циклический сдвиг вправо.

При циклическом сдвиге вправо на единицу младший разряд слова переносится влево и становится его старшим разрядом. При циклическом сдвиге на N эта операция повторяется N раз.

Со стека берутся величина сдвига и сдвигаемое слово. Слово сдвигается вправо на число разрядов, равное указанной величине по модулю 32. Результат помещается на стек.

PC увеличивается на 1.

В терминах интерпретатора:

```
i:=Pop() MOD 20h; Push(ROR(Pop(),i))
```

IO0..IO4 **90h..94h**

Input-Output

Код операции: 1 байт 94h

Непосредственные операнды: зависит от модели процессора

Длина команды: зависит от модели процессора

Действие:

Команды работы с шиной. Индивидуальны для каждого процессора.

В терминах интерпретатора:

```
CASE cpu OF
  |Kronos2_2: ioP2_2
  |Kronos2_5: ioP2_5
  |Kronos2_6: ioP2_6
END
```

ARRCMP **95h**

ARRay CoMPare

Код операции: 1 байт 95h

Непосредственные операнды: 0 байт

Длина команды: 1 байт

Действие:

Команда предназначена для сравнения словных массивов.

Со стека берется размер массивов, затем адрес начала первого массива, затем адрес начала второго массива.

Если размер оказался отрицательным числом, это число помещается обратно на стек и возбуждается прерывание с номером 4Fh. Если размер равен 0, на стек дважды загружается адрес второго массива.

Иначе массивы сравниваются пословно до тех пор, пока сравнение не дойдет до пары последних слов или до пары неравных слов, после чего адреса этой пары слов грузятся на стек.

PC увеличивается на 1.

Замечание. Команда не реализована на П2.2.

В терминах интерпретатора:

```
sz:=Pop(); adr:=Pop(); adr1:=Pop();
IF sz<0 THEN Push(sz); TRAP(4Fh)
ELSIF sz=0 THEN Push(adr1); Push(adr1)
ELSE LOOP
```

```
            IF (adr^ # adr1^) OR (sz=1) THEN
              Push(adr1); Push(adr); EXIT
            END;
            DEC(sz); INC(adr); INC(adr1);
          END;
     END;
```

WM **96h**

Word Move

Код операции: 1 байт 96h
Непосредственные операнды: 0 байт
Длина команды: 1 байт

Действие:

Со стека берется размер словного сегмента, который будет передвинут, затем адрес, начиная с которого двигать, затем адрес, начиная с которого писать этот сегмент. Допускается перекрытие областей источника и приемника.

PC увеличивается на 1.

Замечание. Команда реализована только на Кронос-2.6.

В терминах интерпретатора:

```
sz:=Pop(); f:=Pop(); t:=Pop();
IF t>f THEN
  t:=t+sz-1; f:=f+sz-1;
  WHILE sz>0 DO
    MEM[t]:=MEM[f]; DEC(t); DEC(f); DEC(sz)
  END
ELSE
  WHILE sz>0 DO
    MEM[t]:=MEM[f]; INC(t); INC(f); DEC(sz)
  END
END;
```

BM **97h**

Bit Move

Код операции: 1 байт 97h
Непосредственные операнды: 0 байт
Длина команды: 1 байт

Действие:

Со стека берутся размер пересылаемого битового сегмента, битовое смещение и словный адрес пересылаемого сегмента, битовое смещение и словный адрес, куда пересылается сегмент, и производится пересылка. Допускается перекрытие областей источника и приемника.

PC увеличивается на 1.

Замечание. Команда не реализована на П2.2.

В терминах интерпретатора:

```
sz:=Pop(); -- размер пересылаемой области в битах
i:=Pop(); src:=Pop(); -- смещение и адрес источника
j:=Pop(); trg:=Pop(); -- смещение и адрес приемника
src:=src*32+i;
```

```
trg:=trg*32+j;
IF src>=trg THEN                    n:=+1
ELSE INC(src,sz-1); INC(trg,sz-1); n:=-1
END;
FOR k:=0 TO sz-1 DO
  i:=trg DIV 32; j:=trg MOD 32;
  IF (src MOD 32) IN BITSET(MEM[src DIV 32]) THEN
    MEM[i]:=INTEGER( BITSET(MEM[i]) + {j} )
  ELSE
    MEM[i]:=INTEGER( BITSET(MEM[i]) - {j} )
  END;
  INC(src,n); INC(trg,n)
END
```

FADD **98h**

Float ADDition

Код операции: 1 байт 98h
Непосредственные операнды: 0 байт
Длина команды: 1 байт

Действие:

Берет со стека два вещественных числа, производит сложение и помещает сумму на стек. Если произошло переполнение вещественного, на стек загружается 0 и возбуждается прерывание с номером 42h.

PC увеличивается на 1.

В терминах интерпретатора:

```
Push(REAL(Pop())+REAL(Pop()))
```

FSUB **99h**

Float SUBtraction

Код операции: 1 байт 99h
Непосредственные операнды: 0 байт
Длина команды: 1 байт

Действие:

Берет со стека два вещественных числа - сначала вычитаемое, затем уменьшаемое. Производит вычитание и помещает разность на стек. Если произошло переполнение вещественного, на стек загружается 0 и возбуждается прерывание с номером 42h.

PC увеличивается на 1.

В терминах интерпретатора:

```
X:=REAL(Pop()); Push(REAL(Pop())-X)
```

FMUL **9Ah**

Float MULtiplication

Код операции: 1 байт 9Ah
Непосредственные операнды: 0 байт
Длина команды: 1 байт

Действие:

Перемножает два вещественных числа, взятых со стека. Произведение помещает на стек. Если произошло переполнение вещественного, возбуждается прерывание с номером 42h.

Если результат меньше 2 в степени -128, возбуждается

прерывание с номером 43h (исчезновение порядка).
PC увеличивается на 1.

В терминах интерпретатора:
Push(REAL(Pop())*REAL(Pop()))

FDIV **9Bh**

Float DIVision
Код операции: 1 байт 9Bh
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Берет со стека два вещественных числа - сначала делитель, затем делимое, производит деление и помещает результат на стек. Если произошло переполнение вещественного, возбуждается прерывание с номером 42h.

Если результат меньше 2 в степени -128, возбуждается прерывание с номером 43h (исчезновение порядка).

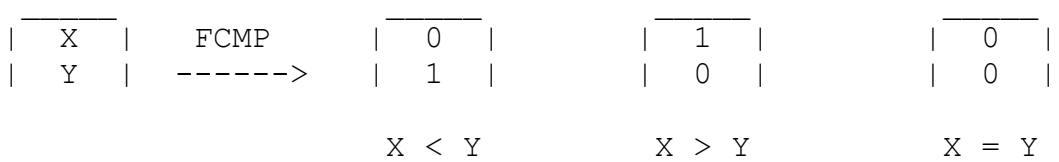
PC увеличивается на 1.

В терминах интерпретатора:
X:=REAL(Pop()); Push(REAL(Pop())/X)

FCMP **9Ch**

Float CoMPare
Код операции: 1 байт 9Ch
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Сравнивает два взятых со стека вещественных числа. Если верхнее меньше нижнего, на стек помещаются последовательно 1 и 0; если верхнее больше нижнего, на стек помещаются последовательно 0 и 1; если они равны, то на стек помещаются 0 и 0.



PC увеличивается на 1.

Замечание. Комбинации команд
FCMP EQU
FCMP NEG
FCMP LSS
....

дают возможность осуществить любое сравнение вещественных чисел.

В терминах интерпретатора:
X:=REAL(Pop()); Y:=REAL(Pop());
IF X<Y THEN Push(1); Push(0)

```
     ELSIF X>Y THEN Push(0); Push(1)
     ELSE Push(0); Push(0) END
```

FABS **9Dh**

Float ABSolute

Код операции: 1 байт 9Dh
Непосредственные операнды: 0 байт
Длина команды: 1 байт

Действие:

Берет со стека вещественное число и помещает на стек его абсолютное значение.

PC увеличивается на 1.

Замечание. Команда никогда не возбуждает прерываний 42h, 43h.

В терминах интерпретатора:

```
     X:=REAL(Pop());
     IF X<0.0 THEN Push(-X) ELSE Push(X) END
```

FNEG **9Eh**

Float NEGative

Код операции: 1 байт 9Eh
Непосредственные операнды: 0 байт
Длина команды: 1 байт

Действие:

Берет со стека вещественное число и помещает на стек его противоположное значение.

PC увеличивается на 1.

Замечание. Команда никогда не возбуждает прерываний 42h, 43h.

В терминах интерпретатора:

```
     Push(-REAL(Pop()))
```

FFCT **9Fh**

Float FunCTion

Код операции: 1 байт 9Fh
Непосредственные операнды: 1 байт
Длина команды: 2 байта

Действие:

Если значение непосредственного операнда равно 0, то со стека берется целое число, преобразуется в вещественное и помещается на стек; если в следующем за командой байте кода лежит 1, то со стека берется вещественное число, у которого отсекается затем дробная часть. Полученное таким образом целое число помещается на стек.

В этих случаях PC увеличивается на 2.

Если в следующем за командой байте кода не 0 и не 1, PC уменьшается на 1 и возбуждается прерывание с номером 7h.

В терминах интерпретатора:

```
     i:=Next();
     IF    i=0 THEN Push(FLOAT(INTEGER(Pop())))
     ELSIF i=1 THEN Push(TRUNC(   REAL(Pop())))
```

```
     ELSE DEC(PC); TRAP(7h)
     END;
```

LSS **0A0h**

integer LeSS
Код операции: 1 байт 0Ah
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Сравниваются два целых числа, взятых со стека. Если нижнее меньше верхнего, на стек помещается 1, иначе 0.

PC увеличивается на 1.

Замечание. Поскольку сравнение не осуществляет вычитания, прерывание 41h (переполнение целого) никогда не возбуждается.

В терминах интерпретатора:

```
     i:=Pop(); Push(Pop()<i)
```

LEQ **0A1h**

integer Less or EQual
Код операции: 1 байт 0A1h
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Сравниваются два целых числа, взятых со стека. Если нижнее меньше или равно верхнему, на стек помещается 1, иначе 0.

PC увеличивается на 1.

Замечание. Поскольку сравнение не осуществляет вычитания, прерывание 41h (переполнение целого) никогда не возбуждается.

В терминах интерпретатора:

```
     i:=Pop(); Push(Pop()<=i)
```

GTR **0A2h**

integer GreaTeR
Код операции: 1 байт 0A2h
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Сравниваются два целых числа, взятых со стека. Если нижнее больше верхнего, на стек помещается 1, иначе 0.

PC увеличивается на 1.

Замечание. Поскольку сравнение не осуществляет вычитания, прерывание 41h (переполнение целого) никогда не возбуждается.

В терминах интерпретатора:

```
     i:=Pop(); Push(Pop()>i)
```

GEQ **0A3h**

integer Greater or EQual

Код операции: 1 байт 0A3h
Непосредственные операнды: 0 байт
Длина команды: 1 байт

Действие:

Сравниваются два целых числа, взятых со стека. Если нижнее больше или равно верхнему, на стек помещается 1, иначе 0.

PC увеличивается на 1.

Замечание. Поскольку сравнение не осуществляет вычитания, прерывание 41h (переполнение целого) никогда не возбуждается.

В терминах интерпретатора:

```
i:=Pop(); Push(Pop()>=i)
```

EQU **0A4h**

integer EQUal

Код операции: 1 байт 0A4h
Непосредственные операнды: 0 байт
Длина команды: 1 байт

Действие:

Сравниваются два целых числа, взятых со стека. Если они равны между собой, на стек помещается 1, иначе 0.

PC увеличивается на 1.

В терминах интерпретатора:

```
Push(Pop()=Pop())
```

NEQ **0A5h**

integer NOt eQual

Код операции: 1 байт 0A5h
Непосредственные операнды: 0 байт
Длина команды: 1 байт

Действие:

Сравниваются два целых числа, взятых со стека. Если они не равны между собой, на стек помещается 1, иначе 0.

PC увеличивается на 1.

В терминах интерпретатора:

```
Push(Pop()#Pop())
```

ABS **0A6h**

integer ABSolute

Код операции: 1 байт 0A6h
Непосредственные операнды: 0 байт
Длина команды: 1 байт

Действие:

Берет со стека целое число и помещает на стек его абсолютное значение.

PC увеличивается на 1.

В терминах интерпретатора:
Push(ABS(Pop()))

NEG **0A7h**

integer NEGative

Код операции:	1 байт	0A7h
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

Берет со стека целое число и помещает на стек его противоположное значение.

PC увеличивается на 1.

Замечание. Если операнд на стеке равен min(INTEGER), то в результате выполнения команды возбуждается прерывание с номером 41h (переполнение целого).

В терминах интерпретатора:
Push(-Pop())

OR **0A8h**

logical bit per bit OR

Код операции:	1 байт	0A8h
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

Берет со стека два слова и помещает на стек результат их сложения, то есть слово, в котором биты выставлены в тех позициях, что в первом и/или во втором слове.

Результат действия команды может быть проиллюстрирован следующей таблицей:

	0	1
0	0	1
1	1	1

PC увеличивается на 1.

В терминах интерпретатора:
v:=BITSET(Pop()); w:=BITSET(Pop()); Push(w+v)

AND **0A9h**

logical bit per bit AND

Код операции:	1 байт	0A9h
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

Берет со стека два слова и помещает на стек результат их перемножения, то есть слово, в котором единицы только в тех позициях, где были единицы в обоих словах.

Результат действия команды может быть проиллюстрирован

следующей таблицей:

	0	1
0	0	0
1	0	1

PC увеличивается на 1.

В терминах интерпретатора:
v:=BITSET(Pop()); w:=BITSET(Pop()); Push(w*v)

XOR **0AAh**
logical bit per bit XOR
Код операции: 1 байт 0AAh
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Берет со стека два слова и помещает на стек результат деления нижнего на верхнее, то есть слово, в котором единицы в тех позициях, где биты не совпали в первом и втором словах, и нули - где совпали.

Результат действия команды может быть проиллюстрирован следующей таблицей:

	0	1
0	0	1
1	1	0

PC увеличивается на 1.

В терминах интерпретатора:
v:=BITSET(Pop()); w:=BITSET(Pop()); Push(w/v)

BIC **0ABh**
logical bit per bit BIt Clear
Код операции: 1 байт 0ABh
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Берет со стека два слова и помещает на стек результат вычитания верхнего из нижнего, то есть нижнее слово, у которого единицы заменились на нули в тех позициях, где у верхнего были единицы.

Результат действия команды может быть проиллюстрирован следующей таблицей:

	0	1
0	0	0
1	1	0

Обратите внимание, что действие команды несимметрично относительно операндов. В таблице биты верхнего слова изображены в верхней горизонтали, биты нижнего слова - в левой вертикали.

PC увеличивается на 1.

В терминах интерпретатора:

```
v:=BITSET(Pop()); w:=BITSET(Pop()); Push(w-v)
```

IN **0ACh**

IN bitset?

Код операции: 1 байт 0ACh
Непосредственные операнды: 0 байт
Длина команды: 1 байт

Действие:

Берет со стека слово и номер позиции, и помещает на стек единицу, если соответствующий бит в слове имеет значение 1, иначе помещает ноль.

PC увеличивается на 1.

Замечание. Команда возбуждает прерывание 4Ah, если номер позиции не в диапазоне 0..31.

В терминах интерпретатора:

```
v:=BITSET(Pop()); Push(Pop() IN v)
```

BIT **0ADh**

set BIT

Код операции: 1 байт 0ADh
Непосредственные операнды: 0 байт
Длина команды: 1 байт

Действие:

В слове, в котором все биты равны нулю, выставляет единицу в позиции, равной значению, взятому со стека, и помещает полученное таким образом слово на стек.

PC увеличивается на 1.

Замечание. Команда возбуждает прерывание 4Ah, если номер позиции не в диапазоне 0..31.

В терминах интерпретатора:

```
i:=Pop();
IF (i<0) OR (i>=20h) THEN TRAP(4Ah)
ELSE w:={}; INCL(w,i); Push(w) END
```

NOT **0AEh**

boolean NOT (@Anot@a bit per bit!)

Код операции: 1 байт 0AEh

Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:
Если значение, взятое со стека, равно нулю, помещает на стек единицу, в противном случае ноль (эквивалентно паре команд LI0 EQU).
PC увеличивается на 1.

В терминах интерпретатора:
Push(Pop()=0)

MOD **0AFh**
integer MODule
Код операции: 1 байт 0AFh
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:
Берет со стека два целых числа. Результат взятия нижнего числа по модулю верхнего помещает на стек.
PC увеличивается на 1.

Замечание. Здесь взятие по модулю - арифметическая операция, определяемая формулой:
X MOD N ⇓ X - (X DIV N) * N,
где DIV - деление с округлением в сторону меньшего для П2.6 и в сторону нуля для младших моделей процессора. См. также команду QUOT.

В терминах интерпретатора:
i:=Pop(); Push(Pop() MOD i)

DECS **0B0h**
DECrement S-register
Код операции: 1 байт 0B0h
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:
Действие, обратное действию ALLOC: берет со стека значение i и на i слов уменьшает S-регистр.
PC увеличивается на 1.

В терминах интерпретатора:
DEC(S,Pop())

DROP **0B1h**
DROP
Код операции: 1 байт 0B1h
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:
Счеркивает слово со стека.
PC увеличивается на 1.

В терминах интерпретатора:

```
     i:=Pop();
```

LODF **0B2h**

reLOaD expression stack after Function return

Код операции: 1 байт 0B2h
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Со стека берется слово, затем из памяти загружается ранее спасенный стек, и сверху на стек грузится взятое слово.

PC увеличивается на 1.

В терминах интерпретатора:

```
     i:=Pop(); RestoreExpStack; Push(i)
```

STORE **0B3h**

STORE expression stack after function call

Код операции: 1 байт 0B3h
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Если величина значения S-регистра, увеличенная на глубину стека выражений + 1, превышает значение границы P-стека (H-регистр), то PC уменьшается на единицу и возбуждается прерывание с номером 40h. Такая проверка производится несмотря на то, что H-регистр дает заниженную на размер стека выражений плюс единица границу P-стека, поскольку этот запас предназначается для сохранения стека выражений в случае переключения процессов.

Иначе стек выражений записывается в память, начиная с адреса S, и следом записывается размер сохраненного стека. В этом случае в результате выполнения команды PC увеличивается на 1.

В терминах интерпретатора:

```
     IF S+ESdepth+1>H THEN DEC(PC); TRAP(40h)
     ELSE SaveExpStack
     END
```

STOFV **0B4h**

STORE expression stack with Formal function value on top

Код операции: 1 байт 0B4h
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Если величина значения S-регистра, увеличенная на глубину стека выражений и еще на двойку, превышает значение границы P-стека (H-регистр), то PC уменьшается на единицу и возбуждается прерывание с номером 40h.

Иначе со стека берется значение, затем стек выражений записывается в память, начиная с адреса S, следом записывается размер спасенного стека, а за ним взятое со стека значение. В этом случае в результате выполнения команды PC увеличивается на 1.

Замечание. Команда STOFV необходима для корректного использования команды CF (см.).

В терминах интерпретатора:

```
IF S+ESdepth+2>H THEN DEC(PC); TRAP(40h)
ELSE i:=Pop(); SaveExpStack; MEM[S]:=i; INC(S) END
```

COPT **0B5h**

COPy Top of expression stack

Код операции:	1 байт	0B5h
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

Берет со стека значение и затем загружает его на стек дважды (дублирует вершину стека).

PC увеличивается на 1.

В терминах интерпретатора:

```
i:=Pop(); Push(i); Push(i)
```

```
(* Будем понимать под словами "откат команды" следующие
   действия:
   1) операнды, взятые со стека в ходе выполнения команды,
   помещаются обратно на стек;
   2) PC возвращается в положение, предшествовавшее выборке
   исполняемой команды;
   3) возбуждается прерывание с указанным номером.
*)
```

CPCOP **0B6h**

Character array Parameter COPy

Код операции:	1 байт	0B6h
Непосредственные операнды:	1 байт	0h..0FFh
Длина команды:	2 байта	

Действие:

Со стека берется верхняя граница байтового массива и вычисляется размер массива в словах. Если значение S-регистра, увеличенное на размер массива в словах, превышает границу P-стека, происходит откат команды с прерыванием номер 40h.

Иначе из кода выбирается непостредственный операнд - смещение, и в память с этим смещением относительно L-регистра записывается текущее значение S-регистра (для того, чтобы запомнить, откуда будет начинаться копия массива). Затем с начала свободной области памяти (S-регистр) поэлементно копируется массив, адрес которого берется со стека, до тех пор, пока не исчерпается размер массива. В этом случае PC увеличивается на 2, S-регистр увеличивается на размер массива.

В терминах интерпретатора:

```
i:=Pop(); (* High *) sz:=(i+4) DIV 4;
IF S+sz>H THEN Push(i); DEC(PC); TRAP(40h)
ELSE MEM[L+Next()]:=S; adr:=Pop();
  WHILE sz>0 DO MEM[S]:=MEM[adr]; INC(S); INC(adr) END
END
```

PCOP **0B7h**

structure Parameter allocate and COPy

Код операции:	1 байт	0B7h
Непосредственные операнды:	1 байт	0h..0FFh
Длина команды:	2 байта	

Действие:

Со стека берется верхняя граница словного массива, увеличивается на единицу для получения его размера. Если значение S-регистра, увеличенное на размер массива, превышает границу P-стека (H-регистр), происходит откат команды с прерыванием номер 40h.

Иначе из кода выбирается непостредственный операнд - смещение, и в память с этим смещением относительно L-регистра записывается текущее значение S-регистра (для того, чтобы запомнить, откуда будет начинаться копия массива). Затем с начала свободной области памяти записывается массив, адрес которого берется со стека, до тех пор, пока не исчерпается размер. В этом случае PC увеличивается на 2, а S-регистр увеличивается на размер массива.

В терминах интерпретатора:

```
i:=Pop(); (* High *) sz:=i+1;
IF S+sz>H THEN Push(i); DEC(PC); TRAP(40h)
ELSE MEM[L+Next()]:=S; adr:=Pop();
  WHILE sz>0 DO MEM[S]:=MEM[adr]; INC(S); INC(adr) END
END
```

FOR1 **0B8h**

enter FOR statement

Код операции:	1 байт	0B8h
Непосредственные операнды:	3 байта	0h..0FFh, 0h..0FFFFh
Длина команды:	4 байта	

Действие:

Если S-регистр, увеличенный на 2, превышает границу P-стека, происходит откат команды с прерыванием номер 40h.

Иначе со стека берется верхняя и нижняя границы цикла, адрес переменной цикла. Из кода выбирается шаг цикла - если в следующем байте 0, шаг положительный, если не 0 - шаг отрицательный. Если условия цикла не выполняются сразу (конечное значение меньше начального при положительном шаге или начальное меньше конечного при отрицательном шаге), PC изменяется на столько, сколько указано в двухбайтовом непосредственном операнде, плюс 4. Иначе нижняя граница записывается по адресу переменной цикла, адрес и верхняя граница размещаются на P-стеке для команды FOR2. В этом случае PC увеличивается в результате выполнения команды на 4.

Замечание. Команда реализована только на Кронос-2.2.

В терминах интерпретатора:

```
IF S+2>H THEN DEC(PC); TRAP(40h)
ELSE sz:=Next(); (* =0 up; #0 down *)
  hi:=Pop(); low:=Pop(); adr:=Pop(); k:=Next2()+PC;
  IF ((sz=0) & (low<=hi)) OR ((sz#0) & (low>=hi)) THEN
```

```
        MEM[adr]:=low;
        MEM[S]:=adr; INC(S); MEM[S]:=hi; INC(S);
      ELSE (* цикл не исполняется не разу *) PC:=k
      END
    END
```

FOR2 **0B9h**

end of FOR statement

Код операции:	1 байт	0B9h
Непосредственные операнды:	3 байта	0h..0FFh, 0h..0FFFFh
Длина команды:	4 байта	

Действие:

Если значение шага цикла, взятое из однобайтового непосредственного операнда, больше 7Fh, ему переприсваивается отрицательное значение, равное (7Fh-шаг) (шаг цикла всегда в промежутке [-128..127]).

Если возможна еще одна итерация, то есть значение переменной цикла, измененное на шаг, не выходит за верхнюю границу, то значение переменной цикла изменяется на размер шага, PC уменьшается на значение двухбайтового непосредственного операнда.

Иначе S-регистр сдвигается на 2 слова назад, счеркивая с P-стека значение верхней границы и адрес переменной цикла.

Замечание. Команда реализована только на Кронос-2.2.

В терминах интерпретатора:

```
hi:=MEM[S-1];
adr:=MEM[S-2];
sz:=Next();
IF sz>7Fh THEN
  sz:=7Fh-sz    (* шаг [-128..127] *)
END;
k:=-Next2()+PC;
i:=MEM[adr]+sz;
IF ((sz>=0) & (i>hi)) OR ((sz<0) & (i<hi)) THEN
  DEC(S,2); (* terminate *)
ELSE MEM[adr]:=i; PC:=k (* continue *)
END
```

ENTC **0BAh**

ENTer Case statement

Код операции:	1 байт	0BAh
Непосредственные операнды:	2 байта	0h..0FFFFh
Длина команды:	3 байта +	размер таблицы

Действие:

Выбор нужной альтернативы из таблицы альтернатив выбирающего оператора (CASE) и передача управления на ее код.

Если S-регистр, увеличенный на 2, превышает границу P-стека, происходит откат команды с прерыванием номер 40h.

Иначе выбирает двухбайтовый непостредственный операнд и осуществляет переход на таблицу альтернатив. Далее из кода выбираются минимальное и максимальное значения альтернатив,

представленные двухбайтовыми непосредственными операндами; вычисляется точка выхода из оператора CASE по формуле:

PC'⇓ PC + 2(high-low),

где high и low - максимальное и минимальное значения альтернатив соответственно. Полученное значение PC' записывается на P-стек.

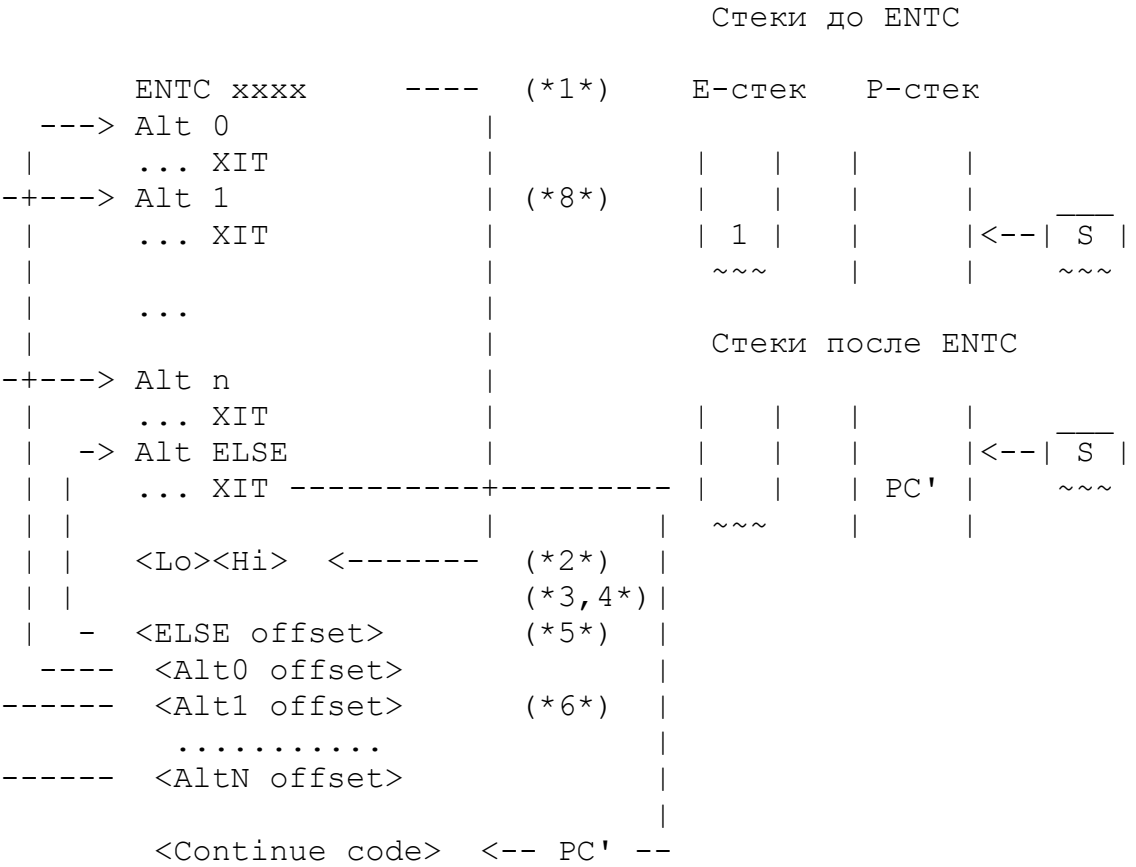
Затем со стека берется параметр оператора CASE. В случае, если его значение не лежит в границах значений альтернатив, PC остается неизменным. Иначе осуществляется переход в таблицу альтернатив по формуле:

PC' = PC + 2(i-low +1),

где i - значение параметра CASE; low - минимальное значение альтернатив.

Далее из кода выбирается двухбайтовое смещение, определяющее переход на код альтернативы. PC изменяется по формуле:

PC' = PC - offset.



Замечание. Команда реализована только на Кронос-2.2.

В терминах интерпретатора:

```
IF S+1>H THEN DEC(PC); TRAP(40h)
ELSE PC:=Next2()+PC; (* jump to case table *)
  k:=Pop(); low:=Next2(); hi:=Next2();
  MEM[S]:=PC + 2*(hi-low) + 4; INC(S);(*PC for exit*)
  IF (k>=low) & (k<=hi) THEN
    PC:=PC+2*(k-low+1) (* jump into case table *)
  END;
```

```
      PC:=-Next2()+PC (* jump back to variant's code *)
    END
```

XIT **0BBh**

eXIT from case or control structure

Код операции: 1 байт 0BBh
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Выход в точку в коде, помеченную при входе в структуру управления (ENTC или ENTS). PC устанавливается равным значению, взятому с P-стека.

Замечание. Команда реализована только на Кронос-2.2.

В терминах интерпретатора:

```
    DEC(S); PC:=MEM[S]
```

ADDPC **0BCh**

ADD Program Counter

Код операции: 1 байт 0BCh
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Грузит на стек PC, сложенный с величиной, взятой со стека.

PC увеличивается на 1.

Замечание. ADDPC не реализована в П2.2.

В терминах интерпретатора:

```
    Push(Pop()+PC);
```

JMP **0BDh**

JuMP

Код операции: 1 байт 0BDh
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

PC присваивается значение, взятое со стека.

Замечание. JMP не реализована в П2.2.

В терминах интерпретатора:

```
       PC:=Pop();
```

ORJP **0BEh**

short circuit OR JumP

Код операции: 1 байт 0BEh
Непосредственные операнды: 1 байт
Длина команды: 2 байта
Действие:

PC увеличивается на 1 в результате выборки кода команды.

Если значение, взятое со стека, не равно нулю, помещает на

стек единицу. PC увеличивается на столько, сколько указано в однобайтовом непосредственном операнде плюс единица (за счет выборки этого операнда).

Иначе увеличивает PC на единицу. Таким образом, в этом случае PC в процессе выполнения команды увеличивается на два.

В терминах интерпретатора:

```
IF Pop()#0 THEN Push(1); PC:=Next()+PC
ELSE INC(PC)
END
```

ANDJP **0BFh**

short circuit AND JumP

Код операции:	1 байт	0BFh
Непосредственные операнды:	1 байт	0h..0FFh
Длина команды:	2 байта	

Действие:

PC увеличивается на 1 в результате выборки кода команды.

Если значение, взятое со стека, равно нулю, помещает на стек ноль. PC увеличивается на столько, сколько указано в однобайтовом непосредственном операнде плюс единица (за счет выборки этого операнда).

Иначе увеличивает PC на единицу. В этом случае PC в процессе выполнения команды увеличивается на два.

В терминах интерпретатора:

```
IF Pop()=0 THEN Push(0); PC:=Next()+PC
ELSE INC(PC)
END
```

MOVE **0C0h**

MOVE block

Код операции:	1 байт	0Ch
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

Со стека берется размер словного сегмента, который будет передвинут, затем адрес, начиная с которого двигать, затем адрес, начиная с которого писать этот сегмент. Передвижка происходит пословно, поэтому в случае определенного перекрытия сегментов хвост передвигаемого сегмента может испортиться еще до того, как его перенесли (см. текст на Модуле). Аккуратный сдвиг выполняется командой WM (96h).

Сдвиг перекрывающихся областей памяти в сторону младших адресов производится корректно; сдвиг в сторону старших можно использовать для заполнения области одинаковыми значениями.

PC увеличивается на 1.

В терминах интерпретатора:

```
sz:=Pop();
i:=Pop(); j:=Pop();
WHILE sz>0 DO
  MEM[j]:=MEM[i]; INC(i); INC(j); DEC(sz)
END
```

CHKNIL **0C1h**

CHecK address for NIL

Код операции:	1 байт	0C1h
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

PC увеличивается на 1 . Если на стеке лежит NIL, то PC уменьшается на 1 и возбуждается прерывание.

Замечание. Команда не реализована на Кронос-2.2.

В терминах интерпретатора:

```
i:=Pop(); Push(i);
IF i=NIL THEN DEC(PC); TRAP(41h) END;
```

LSTA **0C2h**

Load STring Address

Код операции:	1 байт	0C2h
Непосредственные операнды:	2 байта	0h..0FFFFh
Длина команды:	3 байта	

Действие:

Грузит на стек адрес структуры, полученный сложением смещения, взятого из двухбайтового непосредственного операнда и содержимого 1-го слова области глобальных данных.

PC увеличивается на 3.

В терминах интерпретатора:

```
Push(MEM[G+1]+Next2());
```

COMP **0C3h**

COMPare strings

Код операции:	1 байт	0C3h
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

Берет со стека словные адреса двух строк, преобразует их в байтовые, и читает из памяти по два байта, начиная с указанных адресов, до тех пор, пока один из байтов не окажется равным 0, либо до тех пор, пока они не окажутся разными. Тогда на стек загружается последний прочитанный байт сначала второй, затем первой строки с ведущими нулями. Полученные результаты можно сравнить командами EQU, NEQ, LSS и т.д..

PC увеличивается на 1.

В терминах интерпретатора:

```
i:=Pop()*4; j:=Pop()*4;
REPEAT
  a:=CHAR(ByteMEM[i]); b:=CHAR(ByteMEM[j]);
  INC(i); INC(j)
UNTIL (a=0c) OR (b=0c) OR (a#b); Push(a); Push(b)
```

GB **0C4h**

Get procedure Base n level down

Код операции:	1 байт	0C4h

Непосредственные операнды: 1 байт 0h..0FFh
Длина команды: 2 байт
Действие:

Осуществляет проход по статической цепочке на столько уровней, сколько указано в непосредственном операнде, и грузит на стек адрес области локальных данных соответствующей статически объемлющей процедуры.

PC увеличивается на 2.

В терминах интерпретатора:

```
i:=L; k:=Next();
WHILE k>0 DO i:=MEM[i]; DEC(k) END; Push(i)
```

GB1 **0C5h**

Get procedure Base 1 level down
Код операции: 1 байт 0C5h
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Берет из памяти и грузит на стек начало области локальных данных статически объемлющей процедуры.

PC увеличивается на 1.

В терминах интерпретатора:

```
Push(MEM[L])
```

CHK **0C6h**

range boundary CHecK
Код операции: 1 байт 0C6h
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Берет со стека два значения, которые рассматривает как верхнюю и нижнюю границу соответственно; берет и грузит обратно следующий элемент стека, чтобы узнать его значение; если оно не лежит в пределах границ, помещает границы обратно на стек и возбуждает прерывание 4Ah.

PC увеличивается на 1.

В терминах интерпретатора:

```
hi:=Pop(); low:=Pop(); i:=Pop(); Push(i);
IF (i<low) OR (i>hi) THEN
  Push(low); Push(hi); TRAP(4Ah)
END
```

CHKZ **0C7h**

array boundary CHecK (low=Zero)
Код операции: 1 байт 0C7h
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Берет со стека значение, которое интерпретирует как верхнюю границу; в качестве нижней границы подразумевается 0. Берет и грузит обратно следующий элемент стека, чтобы узнать

его значение; если оно не лежит в пределах границ, помещает верхнюю границу обратно на стек и возбуждает прерывание 4Ah.

PC увеличивается на 1.

В терминах интерпретатора:

```
hi:=Pop(); i:=Pop(); Push(i);
IF (i<0) OR (i>hi) THEN Push(hi); TRAP(4Ah) END
```

ALLOC **0C8h**

ALLOCate block

Код операции:	1 байт	0C8h
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

Берет со стека размер отводимой памяти в словах. Если S-регистр, увеличенный на размер, превышает границу P-стека, происходит откат команды с прерыванием номер 40h.

Иначе на стек грузится текущее значение S-регистра (начало отводимой памяти), S-регистр увеличивается на столько слов, каков размер отведенной памяти.

В этом случае PC увеличивается на 1.

В терминах интерпретатора:

```
sz:=Pop();
IF S+sz>H THEN  Push(sz); DEC(PC); TRAP(40h)
ELSE Push(S); INC(S,sz) END
```

ENTR **0C9h**

ENTeR procedure

Код операции:	1 байт	0C9h
Непосредственные операнды:	1 байт	0h..0FFh
Длина команды:	2 байта	

Действие:

Выбирает из кода размер сегмента локальных данных в словах. Если S-регистр, увеличенный на размер, превышает границу P-стека, происходит откат команды с прерыванием номер 40h.

Иначе увеличивает S-регистр на размер локального сегмента. В этом случае в результате выполнения команды PC увеличивается на 2.

В терминах интерпретатора:

```
sz:=Next();
IF S+sz>H THEN DEC(PC,2); TRAP(40h)
ELSE INC(S,sz) END
```

RTN **0CAh**

ReTurN from procedure

Код операции:	1 байт	0CAh
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

Восстанавливаются значение S-регистра до вызова процедуры (т.е. освобождается память, занятая в ходе выполнения

процедуры), значение L-регистра процедуры, из которой была вызвана данная процедура, значение PC и, если вызывалась процедура из внешнего модуля, восстанавливается указатель на область глобальных данных исполняемого модуля и по нему F-регистр.

В терминах интерпретатора:

```
S:=L; L:=MEM[S+1]; PC:=WORD(BITSET(MEM[S+2])*{0..0Fh});
IF ExternalBit IN BITSET(MEM[S+2]) THEN
  (* called from external module *)
  G:=MEM[S]; F:=CodePtr(MEM[G])
END;
```

NOP **0CBh**

No OPeration

Код операции:	1 байт	0CBh
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

PC увеличивается на 1.

При исполнении этой команды больше ничего не делается.

CX **0CCh**

Call eXternal

Код операции:	1 байт	0CCh
Непосредственные операнды:	2 байта	0h..0FFh
Длина команды:	3 байта	

Действие:

Вызов внешней процедуры.

Если значение S-регистра, увеличенное на 4, превосходит границу P-стека, происходит откат команды с прерыванием номер 40h.

Из кода выбираются два однобайтовых непосредственных операнда. Первый интерпретируется как номер внешнего модуля в локальной DFT, из которого вызывается процедура, а второй - как номер процедуры. После этого производится разметка P-стека для вызова указанной процедуры, то есть в нулевое слово области локальных данных заносится G-регистр (для последующего восстановления командой RTN), в первое слово - указатель на область локальных данных той процедуры, из которой произошел вызов (выстраивается динамическая цепочка), во второе слово - PC точки вызова с пометкой в 31-м бите о том, что вызов был внешним (для последующего восстановления PC). Затем по ссылке восстанавливается G-регистр внешнего модуля, по нему - F-регистр, PC устанавливается на начало вызванной процедуры.

В терминах интерпретатора:

```
IF S+4<=H THEN j:=MEM[G-Next()-1]; (* big DFT *)
  i:=Next(); Mark(G,TRUE);
  G:=MEM[j]; F:=CodePtr(MEM[G]); PC:=GetPc(i);
ELSE DEC(PC); TRAP(40h)
END
```

CI **0CDh**

Call procedure at Intermediate level

Код операции:	1 байт	0CDh
Непосредственные операнды:	1 байт	0h..0FFh
Длина команды:	2 байта	

Действие:

Если значение S-регистра, увеличенное на 4, превосходит границу P-стека, происходит откат команды с прерыванием номер 40h.

Иначе выбирается непосредственный операнд, интерпретируемый как номер вызываемой процедуры, после чего производится разметка P-стека для вызова процедуры, то есть в нулевое слово области локальных данных заносится слово, взятое со стека (например, указатель на область локальных данных процедуры, статически объемлющей ту, из которой происходит вызов, для обеспечения доступа к ее локальным данным - строится статическая цепочка), в первое слово - указатель на область локальных данных той процедуры, из которой произошел вызов (строится динамическая цепочка), во второе слово - PC точки вызова (для последующего восстановления PC при возврате из процедуры). Затем PC устанавливается на начало вызываемой процедуры.

В терминах интерпретатора:

```
IF S+4<=H THEN
  i:=Next(); Mark(Pop(),FALSE); PC:=GetPc(i);
ELSE DEC(PC); TRAP(40h) END
```

CF **0CEh**

Call Formal procedure

Код операции:	1 байт	0CEh
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

Если значение S-регистра, увеличенное на 3, превосходит границу P-стека, происходит откат команды с прерыванием номер 40h.

Иначе с вершины P-стека берется слово, в котором содержится процедурное значение вызываемой процедуры, после чего производится разметка P-стека для вызова процедуры, то есть в нулевое слово области локальных данных заносится G-регистр (выстраивается статическая цепочка), в первое слово - указатель на область локальных данных той процедуры, из которой произошел вызов (динамическая цепочка), во второе слово - PC точки вызова с пометкой в 31-м бите о том, что вызов был внешним, поскольку формальная процедура может быть как локальной, так и внешней (для последующего восстановления PC). Затем PC устанавливается на начало вызываемой процедуры. Старший байт процедурного значения интерпретируется как номер процедуры, три младшие - как адрес входа в глобальную DFT. По этому адресу вычисляется адрес глобальных данных модуля, в котором содержится вызываемая процедура, по нему - F-регистр. PC устанавливается на начало процедуры.

Замечания. Ссылка на адрес своей области глобальных данных в 0-м слове локальной DFT модуля необходима, чтобы команда CF выполнялась корректно для собственных локальных процедур модуля.

В терминах интерпретатора:

```
IF S+3<=H THEN i:=MEM[S-1]; DEC(S); Mark(G,TRUE);
  k:=i DIV 1000000h; i:=i MOD 1000000h;
  G:=MEM[i]; F:=CodePtr(MEM[G]); PC:=GetPc(k);
ELSE DEC(PC); TRAP(40h)
END
```

CL **0CFh**

Call Local procedure

Код операции:	1 байт	0CFh
Непосредственные операнды:	1 байт	0h..0FFh
Длина команды:	2 байта	

Действие:

Если значение S-регистра, увеличенное на 4, не превышает границы P-стека (H-регистр), выбирается непосредственный операнд, интерпретируемый как номер вызываемой процедуры, затем производится разметка P-стека для вызова локальной процедуры: в нулевое и первое слова, начиная с адреса S, заносится дважды значение L-регистра; во второе слово - текущее значение PC, после чего в L-регистр заносится значение S-регистра, затем значение S-регистра увеличивается на 4, PC устанавливается на начале процедуры, номер которой был взят из кода.

Иначе происходит откат команды с прерыванием номер 40h.

В терминах интерпретатора:

```
IF S+4<=H THEN i:=Next(); Mark(L,FALSE); PC:=GetPc(i);
ELSE DEC(PC); TRAP(40h) END
```

CL0..CL0F **0D0h..0DFh**

Call Local procedure

Код операции:	4 бита	0Dh
Непосредственные операнды:	4 бита	0h..0Fh
Длина команды:	1 байт	

Действие:

Команды для вызова локальных процедур с номерами из диапазона 0h..0Fh (см. CL).

В терминах интерпретатора:

```
IF S+4<=H THEN Mark(L,FALSE); PC:=GetPc(IR MOD 10h);
ELSE DEC(PC); TRAP(40h) END
```

INCL **0E0h**

INCLude in set

Код операции:	1 байт	0E0h
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

Со стека берется значение N, которое интерпретируется как

номер бита. Он может превышать 31.

Затем со стека берется адрес слова, и в N-й битовой позиции относительно этого адреса выставляется единица. PC увеличивается на 1.

В терминах интерпретатора:

```
i:=Pop(); j:=Pop() + i DIV 32;
MEM[j]:=INTEGER( BITSET(MEM[j]) + {i MOD 32} );
```

EXCL **0E1h**

EXCLude from set

Код операции:	1 байт	0E1h
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

Со стека берется значение N, которое интерпретируется как номер бита. Он может превышать 31.

Затем со стека берется адрес слова, и в N-й битовой позиции относительно этого адреса выставляется ноль. PC увеличивается на 1.

В терминах интерпретатора:

```
i:=Pop(); j:=Pop() + i DIV 32;
MEM[j]:=INTEGER( BITSET(MEM[j]) - {i MOD 32} );
```

INL **0E2h**

membership IN Long set

Код операции:	1 байт	0E2h
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

Со стека берутся размер множества в битах адрес и номер бита. Номер бита может быть произвольным числом. Если номер бита меньше нуля либо не меньше размера множества, то на стек помещается значение 0. Иначе значение, извлеченное из соответствующей битовой позиции относительно взятого адреса, расширяется ведущими нулями до слова и помещается на стек.

PC увеличивается на 1.

Замечание. Команда не реализована на П2.2.

В терминах интерпретатора:

```
k:=Pop(); j:=Pop(); i:=Pop();
IF (i<0) OR (i>=k) THEN Push(0)
ELSE
  Push( (i MOD 32) IN BITSET(MEM[j+i DIV 32]) );
END;
```

QUOT **0E3h**

QUOTient commands

Код операции:	1 байт	0E3h
Непосредственные операнды:	1 байт	0h..0FFh
Длина команды:	2 байта	

Действие:

Со стека берутся два операнда и выбирается однобайтовый непосредственный операнд из кода.

PC увеличивается на 2.

Если значение непосредственного операнда:

- 0, то командой осуществляется деление на степень двойки с округлением к нулю; при этом делимое - нижний операнд, степень - верхний.
- 1, то командой осуществляется деление нижнего операнда на верхний с округлением к нулю;

  Определим операцию "взятие по модулю" формулой:
  X MOD N = X - (X QOU N) * N, (1)
  где QOU - деление с округлением к нулю.

- 2, то команда является операцией "взятие по модулю", определяемой формулой (1), где N - степень двойки; при этом делимое - нижний операнд, степень - верхний;
- 3, то команда является операцией "взятие по модулю", определяемой формулой (1), при этом нижний операнд берется по модулю верхнего;
- любое другое, то PC откатывается назад и возбуждается прерывание с номером 7h.

Замечание. Команда не реализована для П2.2.

В терминах интерпретатора:

```
i:=Pop(); j:=Pop();
CASE Next() OF
  |0: Push( j SHRQ i ); -- деление на степень двойки
  |1: Push( j QOU  i ); -- деление с округлением к нулю
  |2: Push( j ANDQ i ); -- модуль по степени двойки
  |3: Push( j REM  i ); -- модуль
ELSE TRAP(7); DEC(PC,2)
END;
```

INC1 **0E4h**

INCrement by 1

Код операции:	1 байт	0E4h
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

Увеличивает на единицу значение, лежащее в памяти по адресу, взятому со стека.

PC увеличивается на 1.

В терминах интерпретатора:

```
INC(MEM[Pop()])
```

DEC1 **0E5h**

DECrement by 1

Код операции:	1 байт	0E5h
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:
Уменьшает на единицу значение, лежащее в памяти по адресу, взятому со стека.
PC увеличивается на 1.

В терминах интерпретатора:
```
DEC(MEM[Pop()])
```

INC **0E6h**

INCrement

Код операции:	1 байт	0E6h
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:
Берет со стека целое число и адрес. Значение, лежащее в памяти по указанному адресу, увеличивает на вышеупомянутое число.
PC увеличивается на 1.

В терминах интерпретатора:
```
i:=Pop(); INC(MEM[Pop()],i)
```

DEC **0E7h**

DECrement

Код операции:	1 байт	0E7h
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:
Берет со стека целое число и адрес. Значение, лежащее в памяти по указанному адресу, уменьшает на указанное число.
PC увеличивается на 1.

В терминах интерпретатора:
```
i:=Pop(); DEC(MEM[Pop()],i)
```

STOT **0E8h**

STOre Top on procedure stack

Код операции:	1 байт	0E8h
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:
Если значение S-регистра, увеличенное на 1, превышает границу P-стека, происходит откат команды с прерыванием номер 40h.
Иначе со стека выражений берется слово и помещается на P-стек.
PC увеличивается на 1.

В терминах интерпретатора:
```
IF S+1>H THEN DEC(PC); TRAP(40h)
ELSE MEM[S]:=Pop(); INC(S)
END
```

LODT **0E9h**

LOaD Top of procedure stack

Код операции: 1 байт 0E9h
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Значение с вершины P-стека (элемент с адресом S-1) загружается на стек выражений. Значение S-регистра уменьшается на 1.

PC увеличивается на 1.

В терминах интерпретатора:

```
DEC(S); Push(MEM[S])
```

LXA **0EAh**

Load indeXed Address

Код операции: 1 байт 0EAh
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

По взятым со стека размеру элемента (в словах), номеру элемента и адресу структуры (например, массива) вычисляет адрес элемента и помещает его на стек.

PC увеличивается на 1.

В терминах интерпретатора:

```
sz:=Pop(); i:=Pop(); adr:=Pop(); Push(adr+i*sz)
```

LPC **0EBh**

Load Procedure Constant

Код операции: 1 байт 0EBh
Непосредственные операнды: 2 байта 0h..0FFh
Длина команды: 3 байта
Действие:

Из кода выбираются два однобайтовых непосредственных операнда. Первый интерпретируется как номер модуля в локальной DFT, второй - как номер процедуры. Номера упаковываются следующим образом: в старший байт помещается номер процедуры, в младшие три - элемент локальной DFT с указанным номером модуля. Полученное слово грузится на стек.

PC увеличивается на 3.

В терминах интерпретатора:

```
i:=Next(); j:=Next(); Push(j*1000000h+MEM[G-i-1])
```

BBU **0ECh**

Bit Block Unpack

Код операции: 1 байт 0ECh
Непосредственные операнды: 0 байт
Длина команды: 1 байт
Действие:

Со стека берется размер битовой вырезки. Если он меньше 1 или больше 32, происходит откат команды с прерыванием номер

4Ah.

Иначе формируется битовая вырезка указанного размера, с битовым смещением, взятым со стека, и начинающаяся со словного адреса, взятого со стека. Вырезка дополняется ведущими нулями до слова и грузится на стек.

PC увеличивается на 1.

Замечание. Команда не реализована на Кронос-2.2.

В терминах интерпретатора:

```
sz:=Pop();
IF (sz<1) OR (sz>32) THEN
  Push(sz); DEC(PC); TRAP(4Ah)
END;
i:=Pop(); adr:=Pop();
(* j:=битовая вырезка длиной sz, начиная с
   битового адреса (adr,i)
*)
Push(j);
```

BBP **0EDh**

Bit Block Pack

Код операции:	1 байт	0EDh
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

PC увеличивается на 1.

Со стека берется битовое значение и его размер. Если размер меньше 1 или больше 32, размер грузится на стек, PC уменьшается на 1 и возбуждается прерывание с номером 4Ah.

Иначе указанное размером число младших битов вырезки с битовым смещением, взятым со стека, упаковывается по словному адресу, взятому со стека.

Замечание. Команда не реализована на Кронос-2.2.

В терминах интерпретатора:

```
j:=Pop(); sz:=Pop();
IF (sz<1) OR (sz>32) THEN
  Push(sz); DEC(PC); TRAP(4Ah)
END;
i:=Pop(); adr:=Pop();
(* Упаковывает sz младших битов из j по битовому адресу
(adr,i)*)
```

BBLT **0EEh**

Bit BLock Transfer

Код операции:	1 байт	0EEh
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

Со стека берутся битовый размер пересылаемого сегмента (может быть больше 32), битовое смещение и словный адрес пересылаемого сегмента, битовое смещение и словный адрес, куда

пересылается сегмент, и производится пересылка.
PC увеличивается на 1.

Замечание. В случае перекрытия сегментов можно получить самый неожиданный результат. Ожидаемый результат даст использование команды BM (97h).

Замечание. Команда не реализована на Кронос-2.2.

В терминах интерпретатора:

```
sz:=Pop();
i:=Pop(); adr:=Pop();
j:=Pop(); adr1:=Pop();
(* Переслать sz битов (adr,i) -> (adr1,j) *)
```

PDX **0EFh**

Prepare Dynamic indeX

Код операции:	1 байт	0EFh
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

PC увеличивается на 1.

Со стека берутся два операнда, нижний интерпретируется как указатель на дескриптор массива, верхний - как индекс элемента массива. Дескриптор массива - пара слов в памяти, в первом из них лежит адрес начала массива, во втором - верхняя граница массива.

На стек грузится адрес начала массива и индекс элемента. Если индекс оказался меньше 0 или больше, чем граница, указанная в дескрипторе, возбуждается прерывание с номером 4Ah.

Замечание. Команда не реализована на Кронос-2.2.

В терминах интерпретатора:

```
i:=Pop(); dyn:=Pop();
Push(dyn^.adr); Push(i);
IF (i<0) OR (i>dyn^.high)
  THEN TRAP(4Ah)
END
```

SWAP **0F0h**

SWAP

Код операции:	1 байт	0F0h
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

Меняет местами два верхних элемента стека.
PC увеличивается на 1.

В терминах интерпретатора:

```
i:=Pop(); j:=Pop(); Push(i); Push(j)
```

LPA **0F1h**

Load Parameter Address

Код операции:	1 байт	0F1h
Непосредственные операнды:	1 байт	0h..0FFh
Длина команды:	2 байта	

Действие:

Грузит на стек величину, равную значению L-регистра минус значение непосредственного операнда минус единица.

PC увеличивается на 2.

В терминах интерпретатора:

```
Push(L-Next()-1);
```

LPW **0F2h**

Load Parameter Word

Код операции:	1 байт	0F2h
Непосредственные операнды:	1 байт	0h..0FFh
Длина команды:	2 байта	

Действие:

Грузит на стек слово, лежащее по адресу, вычисленному следующим образом: значение L-регистра минус значение непосредственного операнда минус единица.

PC увеличивается на 2.

В терминах интерпретатора:

```
Push(MEM[L-Next()-1]);
```

SPW **0F3h**

Store Parameter Word

Код операции:	1 байт	0F3h
Непосредственные операнды:	1 байт	0h..0FFh
Длина команды:	2 байта	

Действие:

Берет со стека слово и записывает в память по адресу, равному значению L-регистра минус значение непосредственного операнда минус единица.

PC увеличивается на 2.

В терминах интерпретатора:

```
MEM[L-Next()-1]:=Pop();
```

Замечание. Три вышеописанные команды LPA, LPW и SPW поддерживают способ адресации, позволяющий работать с процедурными параметрами, положенными на P-стек до вызова процедуры.

SSWU **0F4h**

Store Stack Word Undestructive

Код операции:	1 байт	0F4h
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

Берет со стека слово и записывает его в память по адресу, взятому со стека, после чего снова загружает это слово на

стек.
PC увеличивается на 1.

В терминах интерпретатора:
```
i:=Pop(); MEM[Pop()]:=i; Push(i)
```

RCHK **0F5h**

Range CHecK

Код операции:	1 байт	0F5h
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:
Со стека берутся два операнда, которые интерпретируются как верхняя (верхний операнд) и нижняя (нижний операнд) границы отрезка. Если число, лежащее на стеке, выходит за границы отрезка, на стек грузится 0 (FALSE), иначе - 1 (TRUE).
PC увеличивается на 1.

Замечание. Команда не реализована на Кронос-2.2.

В терминах интерпретатора:
```
hi:=Pop(); low:=Pop(); i:=Pop(); Push(i);
Push( (i>=low) & (i<=hi) );
```

RCHZ **0F6h**

Range CHeck (low=Zero)

Код операции:	1 байт	0F6h
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:
Со стека берется операнд, который интерпретируется как верхняя граница отрезка. Нижней границей служит 0. Если число, лежащее на стеке, выходит за границы отрезка, на стек грузится 0 (FALSE), иначе - 1 (TRUE).
PC увеличивается на 1.

Замечание. Команда не реализована на Кронос-2.2.

В терминах интерпретатора:
```
hi:=Pop(); i:=Pop(); Push(i);
Push( (i>=0) & (i<=hi) );
```

CM **0F7h**

Call procedure from dynamic Module

Код операции:	1 байт	0F7h
Непосредственные операнды:	1 байт	0h..0FFh
Длина команды:	2 байта	

Действие:
Если позволяет размер процедурного стека, из кода выбирается непосредственный операнд, интерпретируемый как номер процедуры. Осуществляется вызов процедуры с указанным номером из модуля, G-регистр которого взят с P-стека (с которого он при этом счеркивается). Разметка P-стека производится так же, как в случае вызова формальной процедуры

(см. CF). PC устанавливается на начало вызываемой процедуры.
Иначе происходит откат команды с прерыванием номер 40h.

Замечание. Команда не реализована на Кронос-2.2.

В терминах интерпретатора:

```
IF S+4<=H THEN
  i:=Next();
  DEC(S); j:=MEM[S];
  Mark(G,TRUE);
  G:=j; F:=CodePtr(MEM[G]); PC:=GetPc(i);
ELSE DEC(PC); TRAP(40h)
END;
```

CHKBX **0F8h**

CHecK Boxes

Код операции:	1 байт	0F8h
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

Берет со стека два операнда, которые интерпретируются как указатели на упакованные прямоугольники. Прямоугольник определяется диагональю, проведенной из левого нижнего угла в правый верхний. Координаты каждого конца диагонали упакованы в слово, в котором старшие 16 битов определяют координату по вертикали, младшие 16 - по горизонтали. Все координаты положительные (в диапазоне 0..0FFFFh). Если прямоугольники пересекаются, на стек помещается 1 (TRUE), иначе - 0 (FALSE).

PC увеличивается на 1.

Замечание. Команда реализована только на Кронос-2.6.

В терминах интерпретатора:

```
p0:=Pop(); p1:=Pop();
x0b:=INTEGER( BITSET(p0^)     *{0..15});
y0b:=INTEGER((BITSET(p0^)<<16)*{0..15});
INC(p0);
x0e:=INTEGER( BITSET(p0^)     *{0..15});
y0e:=INTEGER((BITSET(p0^)<<16)*{0..15});
x1b:=INTEGER( BITSET(p1^)     *{0..15});
y1b:=INTEGER((BITSET(p1^)<<16)*{0..15});
INC(p1);
x1e:=INTEGER( BITSET(p1^)     *{0..15});
y1e:=INTEGER((BITSET(p1^)<<16)*{0..15});
Push( (x1b<=x0e) & (y1b<=y0e) & (x0b<=x1e) & (y0b<=y1e) );
```

BMG **0F9h**

BitMap Graphic commands

Код операции:	1 байт	0F9h
Непосредственные операнды:	1 байт	0h..0FFh
Длина команды:	2 байта	

Действие:

Команда для поддержки bitmap-графики. Поскольку точное описание команды на языке Модула-2 весьма обширно и

представляет интерес только для специального изучения, оно выделено в самостоятельный документ "Интерпретатор графических команд". Здесь приводится лишь поверхностное описание функций команды BMG.

Команда реализована на Кронос-2.6 и с некоторыми ограничениями на Кронос-2.5.

Для применения команды рекомендуется использовать соответствующую библиотеку графики.

Команда выбирает байт из кода. Если полученное целое число выходит из диапазона [0..9], то происходит откат PC на 2 и возбуждается прерывание номером 49h. Иначе число интерпретируется как номер одной из описанных ниже подкоманд и соответствующая подкоманда исполняется.

Все параметры передаются командам либо непосредственно через стек, либо через указатели, размещенные на стеке, которые указывают на массив параметров в памяти. В описании операнды перечисляются в том порядке, в котором они берутся со стека.

Команды осуществляют операции над областью памяти (битовой картой), определенной так называемым bitmap-дескриптором (BMD).

Bitmap-дескриптор представляет собой запись из пяти слов, первое из которых - горизонтальный размер битовой карты в битах, второе - вертикальный размер битовой карты в битах, третье - число слов в одной строке изображения, четвертое - адрес битовой карты, пятое - шаблон, который использует подкоманда DLN (см. ниже).

Код: 0
Подкоманда: IN_RECT (IN RECTangle?)
Действие: проверка принадлежности точки прямоугольнику с диагональю, проведенной из левого нижнего угла в правый верхний, с координатами (0,0) и (w,h).
Операнды на стеке:
1,2) горизонтальная и вертикальная координаты точки;
3,4) горизонтальная и вертикальная координаты правого верхнего угла прямоугольника.
Результат на стеке:
FALSE (0) - если точка не принадлежит прямоугольнику; TRUE (1) - если принадлежит.

Код: 1
Подкоманда: DVL (Display Vertical Lines)
Действие: построение на битовой карте изображения отрезка вертикальной линии указанной длины в указанной моде от указанной точки в сторону увеличения адресов.
Операнды на стеке:
1) режим, в котором будет строиться изображение отрезка (REPLACE, OR, XOR, BIC), определяется операндом, равным 0,1,2 или 3 соответственно.
2) указатель на BMD;
3,4) координаты точки по горизонтали и вертикали;

5) длина линии.
Результат на стеке: нет.

Код: 2
Подкоманда: BBLT_G (BBLT-Graphic)
Действие: команда BBLT с режимами: производит логические операции (REPLACE, OR, XOR, BIC) над приемником и источником, являющимися массивами битов.
Операнды на стеке:
1) режим, указывающий, какая операция будет произведена над приемником и источником (REPLACE, OR, XOR, BIC), определяется операндом, равным 0,1,2 или 3 соответственно.
2) операнды, аналогичные операндам команды BBLT.
Результат на стеке: нет.

Код: 3
Подкоманда: DCH (Display CHar)
Действие: построение на битовой карте изображения символа с указанной кодировкой из указанного шрифта.
Операнды на стеке:
1) режим, в котором будет изображаться символ (REPLACE, OR, XOR, BIC), определяется операндом, равным 0, 1, 2 или 3 соответственно;
2) указатель на BMD;
3,4) координаты символа по горизонтали и по вертикали;
5) указатель на дескриптор шрифта;
6) кодировка символа, изображение которого требуется построить.
Результат на стеке: нет.

Код: 4
Подкоманда: CLP (CLiPping)
Действие: клипирование отрезка прямой прямоугольником с диагональю, проведенной из левого нижнего угла в правый верхний, с координатами (0,0) и (w,h).
Операнды на стеке:
1) указатель на массив из 4-х параметров. Перед выполнением команды в эти 4 слова должны быть занесены координаты концов отрезка. Если отрезок пересекается с прямоугольником, то команда заносит в них координаты концов клипированного отрезка.
2,3) горизонтальная и вертикальная координаты правого верхнего угла прямоугольника.
Результат на стеке:
TRUE, если отрезок пересекается с прямоугольником, иначе FALSE.

Код: 5
Подкоманда: DLN (Display LiNe)
Действие: построение на битовой карте изображения отрезка произвольной прямой по двум точкам.
Операнды на стеке:

1) режим, в котором будет изображаться отрезок (REPLACE, OR, XOR, BIC), определяется операндом, равным 0,1,2 или 3 соответственно;
2) указатель на BMD;
3,4) горизонтальная и вертикальная координаты одного конца отрезка;
5,6) горизонтальная и вертикальная координаты другого конца отрезка.

Результат на стеке: нет.

Замечание. Подкоманда DLN не отслеживает выход отрезка за границы битовой карты.

Код: 6
Подкоманда: CRC (CiRCus)
Действие: реализует тело цикла изображения на битовой карте окружности с указанными центром и радиусом. Строит на битовой карте 8 точек, принадлежащих этой окружности.
Операнды на стеке:
1) режим, в котором будет строиться изображение (REPLACE, OR, XOR, BIC), определяется операндом, равным 0,1,2 или 3 соответственно;
2) указатель на BMD;
3) указатель на массив параметров (радиус, 0 и радиус, поделенный надвое).
4,5) горизонтальная и вертикальная координаты центра окружности.

Результат на стеке: нет.

Код: 7
Подкоманда: ARC (ARC)
Действие: реализует тело цикла изображения на битовой карте дуги окружности с указанными центром, радиусом и углом, заданным двумя точками.
Операнды на стеке:
1) режим, в котором будет строиться изображение (REPLACE, OR, XOR, BIC), определяется операндом, равным 0,1,2 или 3 соответственно;
2) указатель на BMD;
3) указатель на массив параметров.

Результат на стеке: нет.

Код: 8
Подкоманда: TRF (TRiangle Filling)
Действие: поддерживает процедуру заполнения области, ограниченной прямыми. Вычисляет координаты точек, между которыми нужно провести горизонтальную линию, чтобы закрасить область.
Операнды на стеке:
указатель на массив параметров. В результате выполнения команды в него заносятся вычисленные координаты.

Результат на стеке: нет.

```
Код:     9
Подкоманда: CRF (CiRcus  Filling)
Действие:   поддерживает процедуру заполнения круга. Вычисляет
            координаты точек, между которыми нужно провести
            горизонтальную линию, чтобы закрасить круг.
Операнды на стеке:
            указатель  на массив параметров. В результате
            выполнения команды в них заносятся вычисленные
            координаты.
Результат на стеке: нет.

В терминах интерпретатора:
     CASE Next() OF
       |0: IN_RECT
       |1: DVL
       |2: BBLT_G
       |3: DCH
       |4: CLP
       |5: DLN
       |6: CRC
       |7: ARC
       |8: TRF
       |9: CRF
     ELSE
       Trap(49h)
     END;
```

ACTIV **0FAh**

```
ACTIVe process
Код операции:                   1 байт          0FAh
Непосредственные операнды:      0 байт
Длина команды:                  1 байт
Действие:
     Загружает  на  стек  адрес дескриптора активного процесса
(содержимое P-регистра процессора).
     PC увеличивается на 1.

В терминах интерпретатора:
     Push(P)
```

USR **0FBh**

```
USeR defined functions
Код операции:                   1 байт          0FBh
Непосредственные операнды:      1 байт          0h..0FFh
Длина команды:                  2 байта
Действие:
     Команда  оставлена  для дальнейшего возможного расширения
системы команд.
     PC увеличивается на 1.

В терминах интерпретатора:
     i:=Next(); (*  *)
```

SYS **0FCh**

rarely SYStem functions

Код операции:	1 байт	0FCh
Непосредственные операнды:	1 байт	0h..0FFh
Длина команды:	2 байта	

Действие:

Если в значение непосредственного операнда равно 0, то грузит на стек слово, идентифицирующее процессор. Если 2 - модель процессора. Иначе возбуждается прерывание с номером 7h.

PC увеличивается на 2.

В терминах интерпретатора:

```
CASE Next() OF
  |00h: (* PID Processor IDent *)
       (* Push(PID) *)
  |02h: (* PM Processor Model *)
       (* Push(PM) *)
 (* Остальные могут быть различными в разных моделях *)
ELSE TRAP(7h)
END;
```

NII **0FDh**

Never Implemented Instruction

Код операции:	1 байт	0FDh
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

При исполнении этой команды возбуждается прерывание с номером 7h. PC увеличивается на 1.

Замечание. Команда не реализована на Кронос-2.2.

В терминах интерпретатора:

```
TRAP(7h);
```

INVLD **0FFh**

INVaLiD command

Код операции:	1 байт	0FFh
Непосредственные операнды:	0 байт	
Длина команды:	1 байт	

Действие:

При вызове этой команды возбуждается прерывание с номером 49h. PC увеличивается на 1.

В терминах интерпретатора:

```
TRAP(49h)
```

ЧАСТЬ IV. ИЛЛЮСТРАЦИИ К АРХИТЕКТУРЕ ПРОЦЕССОРОВ

Данный раздел может быть использован как пособие по архитектуре семейства процессоров КРОНОС, системе команд (М-код) и работе Модула-2 компилятора.

Будем вести изложение в следующей форме:

Текст программы на Модуле - 2	Генерируемый код с комментариями

Заметим, что приведенный код является иллюстративным и поэтому может отличаться от кода, порождаемого текущей версией компилятора. Причины следующие:

1) в примерах не отражены оптимизации, которые повышают эффективность кода, но затрудняют восприятие примера;

2) в некоторых примерах приведено несколько вариантов генерации, а в текущей версии компилятора реализован, естественно, какой-то один;

3) опущены команды динамического контроля (например, контроля границ массивов).

В примерах используются мнемоники команд М-кода. Формальное описание команд приведено в интерпретаторе М-кода.

1. ОПЕРАТОРЫ

1.1. Присваивание

```
MODULE   M;

VAR  G:INTEGER;
     B:BITSET;

BEGIN
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
| G:=1;                                LI1 SGW2                 |
| G:=G+255;                            LGW2 LIB FF ADD SGW2     |
- - - - - - - - - - - - - - - - - - - - |- - -\-/- -|- -|- --
                                        |      |    |   |
                                        G     255  '+'  G:=
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
| B:={0..31};                          LIW FFFFFFFF SGW3        |
- - - - - - - - - - - - - - - - - - - - - \-/- - - - -|- - --
END M.                                      |          |
                                          {0..31}       B:=
```

1.2. Доступ к глобальным переменным

```
MODULE  M;

VAR  G2,G3,G4, ... ,G255:INTEGER;
     G256: INTEGER;


BEGIN
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - -
| G2  :=  G2;                          LGW02  SGW02 |
| G255:=G255;                          LGW FF SGW FF|  (1.2.1)
| G256:=G256;                               ---     |  (1.2.2)
- - - - - - - - - - - - - - - - - - - - - - - - - - -
END M.
```

Примечание 1.2.1. К первым 14 глобалам с номерами 2..15 доступ однобайтовый, к глобалам с номерами 16..255 - двухбайтовый.

Примечание 1.2.2. Это не компилируется текущей версией компилятора, но может порождаться так:

```
      LGA FF LSW1      LGA FF SSW1
```

1.3. Доступ к внешним переменным

Внешними переменными называются глобальные переменные других модулей.

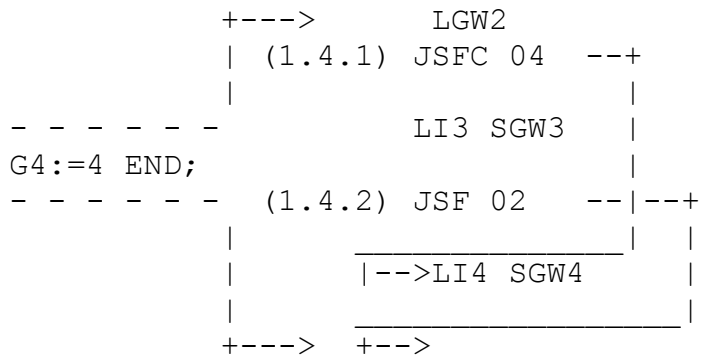
```
DEFINITION MODULE M;
  VAR i: INTEGER;
END M.
```

```
                      номер модуля M в локальной DFT модуля N
MODULE N;                              | Номер
FROM M IMPORT i;                       | переменной i в модуле M
BEGIN                                  |  |
- - - - - - - - - - - - - - - - - - - |- |- - - - - - --
| i:=i;                          LEW 01 02  SEW 01 02 |
- - - - - - - - - - - - - - - - - - - - - - - - - - - - --
END N.
```

Подробнее о локальной DFT можно узнать из примеров, относящихся к процедурам (см. раздел 2).

1.4. Условный оператор (IF)

```
MODULE M;

VAR bool :BOOLEAN;
    G3,G4:INTEGER;
BEGIN
 - - - - - - - - - - - - - - - - - -
|IF bool THEN G3:=3 ELSE G4:=4 END;
 - - - - - - - - - - - - - - - - - -
END M.
```



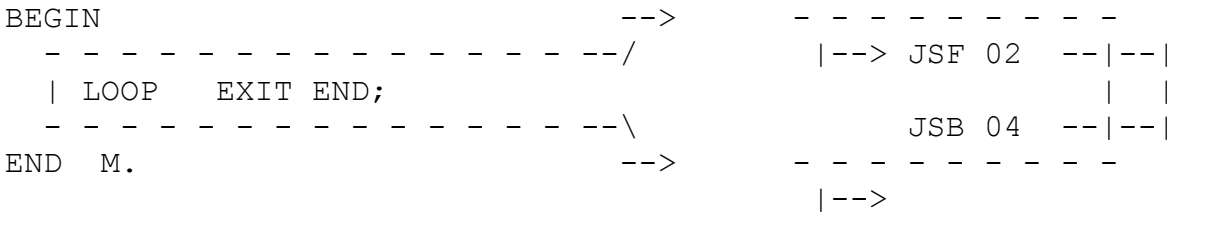
Примечание 1.4.1. Если на A-стеке 0, то PC увеличивается на 4 байта, пропуская TRUE-альтернативу. 0 интерпретируется как FALSE, #0 - TRUE.

Примечание 1.4.2. Безусловный переход (если есть FALSE-альтернатива).

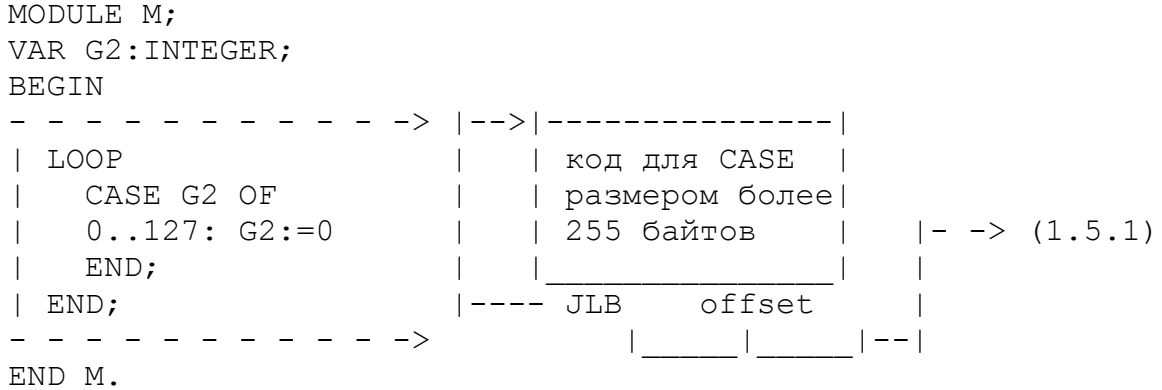
1.5. Оператор цикла (LOOP)

Генерация LOOP-цикла переходом (как реализовано в текущей версии):

```
MODULE M;
```

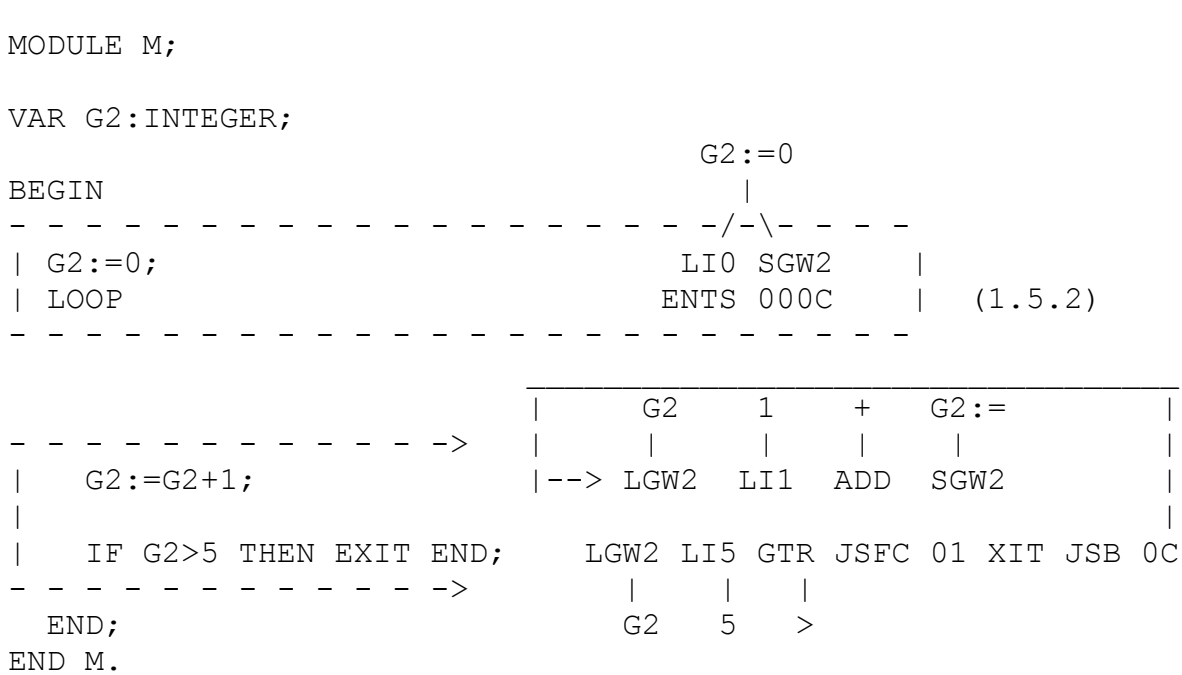


Если тело цикла большое, то переход может быть двухбайтовым.



Примечание 1.5.1. Двухбайтовое смещение для переходов длиннее 255 байт.

Генерация LOOP-цикла с помощью команд ENTS и XIT:



Примечание 1.5.2. ENTS кладет на вершину P-стека текущее значение PC + значение следующих за командой двух байтов. XIT берет с верхушки P-стека новое значение PC, совершая переход.

1.6. Оператор цикла (REPEAT)

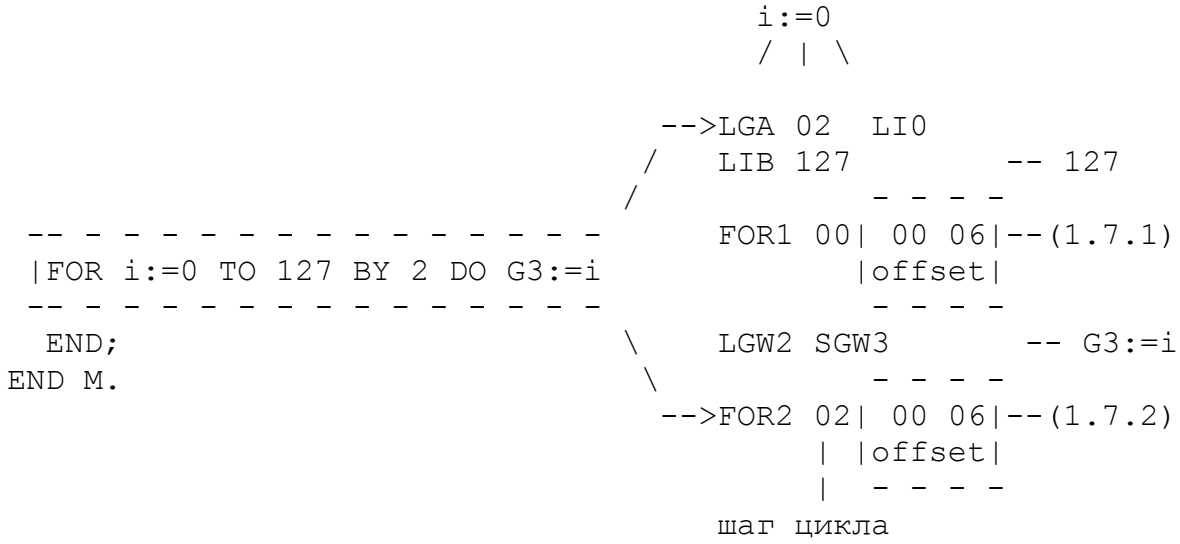
```
MODULE M;
```

```
  VAR G2:INTEGER; bool: BOOLEAN;
BEGIN
 - - - - - - - - - - - - - - - - - ->
 | REPEAT G2:=1 UNTIL bool;
 - - - - - - - - - - - - - - - - - ->
END M.
```

```
 _____________________
|--> LI1  SGW2       |
     LGW3 JSBC 05 --|
```

1.7. Оператор цикла (FOR)

```
MODULE M;

 VAR i,G3:INTEGER;

BEGIN
```



Примечание 1.7.1. FOR1 берет со стека конечное значение, начальное значение, адрес параметра цикла. Знак шага - следующий байт кода. Если в следующем байте 0, шаг положительный, если не 0 - шаг отрицательный. Если условия цикла не выполняются сразу (конечное значение меньше начального при положительном шаге или наоборот), тело цикла обходится с помощью указанного размера тела (offset); иначе начальное значение присваивается по указанному адресу параметра цикла, на P-стек помещаются адрес параметра цикла и конечное значение, после чего выполняется тело цикла.

Примечание 1.7.2. FOR2 по параметрам, взятым с P-стека, и шагу цикла определяет возможность еще одной итерации. Если она возможна, увеличивает значения, лежащие по указанному адресу на 'шаг' и осуществляет переход назад на указанное число байтов (offset). Иначе

счеркивает с P-стека параметры (адрес переменной цикла и конечное значение).

1.8. Оператор выбора (CASE)

```
MODULE M;

  VAR G2,G3:  INTEGER;

BEGIN


- - - - - - - - - - ->                  LGW2
| CASE G2 OF                            ENTC 0009 - - |
|  |1..2: G3:=2           |- - - - ->LI2 SGW3  XIT |
|  |5   : G3:=3        |---|--------->LI3 SGW3  XIT | (1.8.1)
| ELSE    G3:=4        |   |   |----->LI4 SGW3  XIT |
| END;                 |   |   |   _ _ _ _ _ _ _ _|
- - - - - - - - - - ->  |   |   |   |
                        |   |   |   |->0001 0005       (1.8.2)
END M.                  |   |   |
                        |   |   |   смещение  альтернатива
                        |   |   |       |        |
                        |   |   |------0009 -- ELSE
                        |   |- -|- - - 0011 -- 1
                        |   |- -|- - - 0013 -- 2
                        |       |------000F -- ELSE
                        |       |------0011 -- ELSE
                       ------------- 0016 -- 5
```

Примечание 1.8.1. ENTC выбирает двухбайтовое смещение из кода и переходит в таблицу выбора.

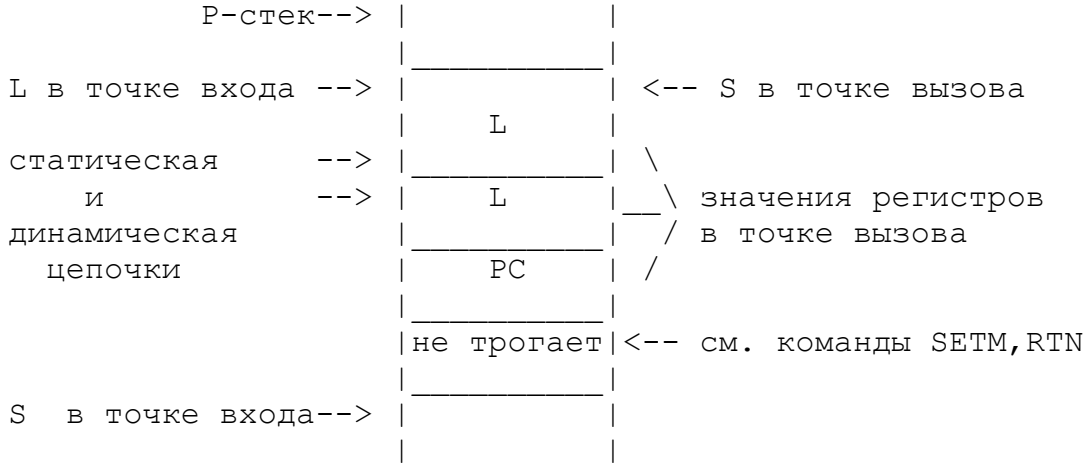
Примечание 1.8.2. Из кода выбираются минимальное (lo=1) и максимальное (hi=5) значения альтернатив. Затем вычисляется PC точки выхода PC'=PC+2*(hi-lo)+4 и записывается на P-стек. Затем с A-стека выбирается значение параметра i и подготавливается переход к нужной альтернативе: PC:=PC+2*(i-lo+1). В случае i>hi или i<lo PC остается неизменным. Теперь все готово к переходу: из кода выбирается двухбайтовое смещение delta, определяющее переход PC:=PC-delta. После выполнения группы операторов выбранной альтернативы выполняется команда XIT, которая осуществляет выход из CASE путем выполнения PC:=PC'. PC' берется командой XIT с P-стека.

2. ПРОЦЕДУРЫ

2.1. Описание и вызов примитивной процедуры

```
MODULE M;

PROCEDURE P; (* процедура 1 *)

      - - - - - - - - - - - - - - - - - -
BEGIN | RETURN END P;              RTN   |
      - - - - - - - - - - - - - - - - - -

BEGIN        (* процедура 0 *)
 - - - - - - - - - - - - - - - - - - - - -
 | P;                               CL1 | (2.1.1)
 - - - - - - - - - - - - - - - - - - -|- -
END M.                                |
                        вызов локальной процедуры 1
```

Примечание 2.1.1. CL1 маркирует P-стек следующим образом:



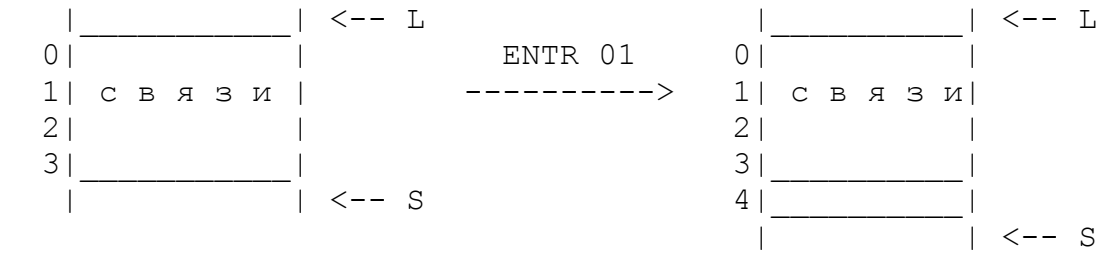
После маркировки в L засылается значение S, а в S значение S+4. По номеру процедуры (в данном случае - 1) из процедурной таблицы (см. Примечание 2.1.2) извлекается смещение до начала кода соответствующей процедуры и засылается в PC, после чего начинают исполняться команды тела процедуры. RTN берет из области связей процедуры значение PC и L в точке их вызова и засылает их в регистры PC и L. В регистр S засылается значение L в точке возврата (которое является старым значением S в точке вызова).

Примечание 2.1.2. Процедурная таблица - таблица, по номеру процедуры указывающая смещение начала кода процедуры относительно начала кода модуля. Нулевая процедура - инициализирующая часть модуля.

2.2. Работа с локалами процедуры

```
MODULE M;
PROCEDURE P; (* процедура 1 *)
                        число локальных переменных процедуры
                                          |
      - - - - - - - - - - - - - - - - - - |- - - --
VAR   |L4: INTEGER;                 ENTR 01      | (2.2.1)
BEGIN |L4:=0; RETURN                LI0 SLW4 RTN |
      - - - - - - - - - - - - - - - - - - |- - --
END P;                                    |
                                  первые четыре слова (0..3)
                                  заняты под область связей

BEGIN        (* процедура 0 *)
- - - - - - - - - - - - - - - - - - - - - - - - -
|P;                                 CL1    |
- - - - - - - - - - - - - - - - - - - - - - - - -
END M.
```

Примечание 2.2.1. ENTR 01 передвигает регистр S на одно слово:



После этого становится возможным работать с локальной переменной командами LLW, SLW (аналогичными командам LGW, SGW), адресующими данные относительно L-регистра.

2.3. Вложенные процедуры

```
MODULE M;


PROCEDURE p1;  (* процедура 1*)
  VAR p1L4:INTEGER;


  PROCEDURE p2; (* процедура  2*)
```

```
                                кладет      на     A-стек
                                L-регистр    охватывающей
  VAR p2L4:INTEGER;             процедуры p1
  BEGIN                              |
     - - - - - - - - - - - - - - - -|- - - - - - - - ---
     |p1L4:=12;                      GB1  LI12  SSW4    |
     |p2L4:=11;                      LI11 SLW4  RTN     |
     - - - - - - - - - - - - - - - - - - - - - - - - - - -
  END p1;                номер процедуры в процедурной таблице
BEGIN                                          |
  - - - - - - - - - - -- - - - - - - - - - - -|- - -
  |p1L4:=2;                          LI2  SLW4   |     |
  |p2;                               LLA 00  CI 02 RTN | (2.3.1)
  - - - - - - - - - - -- - - - - - - -|- -|- - - - -
END p1;                                |   |
                      Значение L-регистра |
                                          |
               команда вызова процедуры промежуточного уровня

BEGIN
- - - - - - - - - - - - - - - - - - - - -
|p1;                                CL1  |
- - - - - - - - - - - - - - - - - - - - -
END M.
```

2.4. Вызов внешней процедуры

```
DEFINITION MODULE N;

PROCEDURE proc1;

END N.


MODULE M;

FROM N IMPORT proc1;


                                      номер модуля
                                        |
PROCEDURE p1;                           | номер процедуры модуля
BEGIN                                   |  |
- - - - - - - - - - - - - - - - - - - -| -|- - -
|  proc1;                          CX 01 01 RTN |  (2.4.1)
- - - - - - - - - - - - - - - - - - - - - - - - - -
END p1;

END M.
```

Примечание 2.4.1. В области связей модуля элемент с номером 1

соответствует модулю N и после загрузки модуля в память ссылается на слово, содержащее адрес начала области глобальных данных модуля N, в первом слове которой содержится F-регистр модуля N, позволяющий добраться до его сегмента кода.

CX помещает в нулевое слово области связей процедуры G-регистр модуля, вызвавшего эту процедуру, и помечает, выставляя признак во втором локальном слове, что вызов был внешним. Команда RTN анализирует этот признак и в случае необходимости восстанавливает значение G-регистра.

2.5. Размещение мультизначений

```
MODULE M;

PROCEDURE p;

    - - - - - - - - - - - - - - - - - - - - - - - - - -
VAR |i: INTEGER;                    ENTR 02           |
    |A: ARRAY [0..15] OF INTEGER;  LIB 10 ALLOC SLW5 |(2.5.1)
    - - - - - - - - - - - - - - - - - - - - - - - - - -

BEGIN

  - - - - - - - - - - - - - - - - - - - - - - - - - - -
  |i:=0;                        LI0  SLW4   |        |
  |                                         |        |
  |                              A    i     |        |
  |                              |    |     |        |
  |A[i]:=1;                     LLW5 LLW4 LI1 SXW    | (2.5.2)
  - - - - - - - - - - - - - - - - - - - - - - - - - - - (2.5.3)
END p;

END M.
```

Примечание 2.5.1. ALLOC берет с A-стека размер мультизначения (массив или запись), кладет на A-стек значение регистра S и передвигает S на взятый размер, тем самым отводя нужное число слов на P-стеке для мультизначения. После этого адрес массива запоминается в нужном локальном слове.

Примечание 2.5.2. Индексация без контроля границ. По умолчанию контроль границ включен.

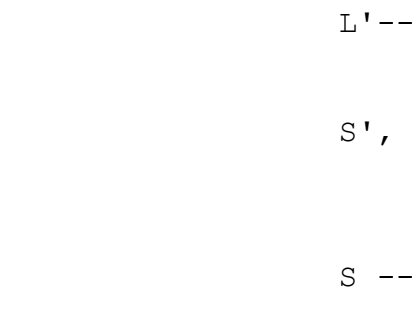
Примечание 2.5.3. При возврате из процедуры команда RTN переставляет S в L, и тем самым освобождает всю память, занятую локалами, в том числе и память, выделенную на P-стеке командой ALLOC.

2.6. Работа с процедурными значениями

```
MODULE M;

TYPE proc1=PROCEDURE (INTEGER);

PROCEDURE P(p1 : proc1);
(* процедура # 1 *)

                      сохранить A-стек на P-стеке
                         | загрузить процедурное значение P1
                         | из локального слова 4 процедуры P
                         |     |
BEGIN                    |     |  записать на P-стек значение
                         |     |     |            с  A-стека
  - - - - - - - - - - - -|- - -|- - -|- - - - - - - - - - - -
  |p1(1);              SLW4   LLW4  STOT  LI1  CF  RTN  |
  - - - - - - - - - - - - - - - - - - - - - - - -|- -|- - - - -
END P;                                           |   |
                                          параметр |
                                                 вызов формальной
                                                       процедуры

PROCEDURE p(w: INTEGER); (* процедура # 2 *)
BEGIN
  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  |                                  SLW4     RTN |
  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
END p;

VAR v: proc1;          номер модуля =0, т.к. проц. собственная
                                        | номер процедуры
BEGIN                                   |  |
  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  |v:=p;                              LPC 00 02  SGW2       |
  |v(5);                              LGW2  STOT  LI5  CF   |
  |P(v);                              LGW2  CL1             |
  |P(p);                              LPC 00 02  CL1        |
  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
END M.
```

```
                          |<--  о д н о   с л о в о -->|
                          |_____________________________|
Процедурное значение:     | 31..24 | 23 . . . . . .0 |
                          -----|-------------|----------
                               |             |
                            номер      24-разрядный адрес
                           процедуры   G-регистра модуля
```

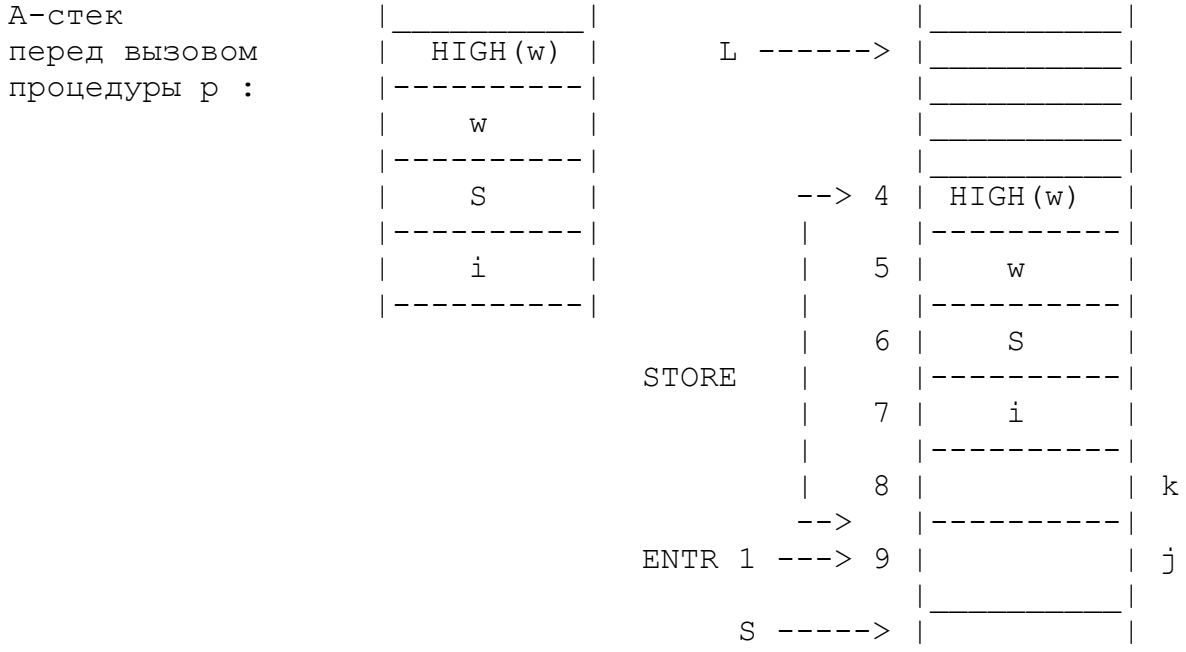
Процедурные связи при вызове процедуры командой CF (аналогично CX):



2.7. Передача параметров

```
MODULE M;

TYPE String=ARRAY [0..255] OF CHAR;

PROCEDURE P(i: INTEGER; S: String; VAR w: ARRAY OF CHAR);
  VAR k,j:INTEGER;
BEGIN
(* сохранение параметров и размещение переменных *)
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
|                                 STORE ENTR 01    |  (2.7.1)
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
(* копирование массива S, переданного по значению *)
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
|k:=HIGH(S);                       LIB FF  SLW8   |
|j:=HIGH(w);                       LLW4 SLW9 RTN  |  (2.7.2)
|                                   |    |        |
|                                  HIGH  j        |
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
END P;

VAR
    - - - - - - - - - - - - - - -- - - - - - - -  - -
    |str8:ARRAY[0..7] OF CHAR;  LI2 ALLOC SGW2    |
    |str :String;               LIB 40 ALLOC SGW3 |
    - - - - - - - - - - - - - - -- - - - - - - - -  -
BEGIN
                    загрузка на A-стек адреса константы 'abc'
                                 |
 - - - - - - - - -  - - - - - - -|- - - - - - - - - - - -
 |P(1,'abc',str8);        LI1 LSTA 0001 LGW2 LI7 CL1 |
 |                                                   |
 |str:='def';             LGW3 LSTA *ind* LI1 MOVE   | (2.7.3)
 |                                                   |
 |P(2,str,str);           LI2 LGW3 LGW3 LIB FF CL1   |
 - - - - - - - - -  - - - - - - - - - - - - - - - - - -
 END M.
```

Примечание 2.7.1. STORE записывает A-стек на P-стек, выделяя на P-стеке память под локалы: 4-е слово - HIGH(w), 5-е слово - w(адрес), 6-е слово - S (адрес), 7-е слово - i, 8-е слово - число параметров - используется для локала k. Команда ENTR 01 завершает отведение памяти; 9-е слово отводится для j :



Примечание 2.7.2. Здесь приведена группа команд, которая копирует переданный по значению массив S. Приводим три возможных вырианта кода:

```
I)  LIB 40        II)  LLW6          III)  LLW6
    ALLOC              LIB FF              LIB 3F
    COPT               CPCOP 06            PCOP 06
    LLW6
    LIB 40
    MOVE
    SLW6
```

Очевидна целесообразность введения в систему команд CPCOP и PCOP, предназначенных для размещения и копирования мультипараметров.

Примечание 2.7.3. LSTA *ind* по относительному смещению *ind* в строковом пуле грузит на A-стек адрес соответствующей строковой константы.

2.8. Вызов функции (на непустом стеке)

```
MODULE M;

PROCEDURE f(i,j: INTEGER): INTEGER;
BEGIN
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
| RETURN  i+j                STORE LLW5 LLW4 ADD RTN |
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
END f;

PROCEDURE p(i,j: INTEGER);
  VAR k:INTEGER;
BEGIN
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
| k:=i+j;                       STORE LLW5 LLW4 ADD SLW6 RTN |
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
END p;
VAR v: PROCEDURE(INTEGER,INTEGER): INTEGER;
BEGIN
- - - - - - - - - - - -- - - - - - - - - - - - - - - - - - - -
| p(1,f(2,3));   LI1     STORE LI2 LI3 CL1 LODFV CL2 |
| v:=f;          LPC 00 01 SGW2                      |
| p(1,v(2,3));   LI1 LGW2 STOFV LI2 LI3 CF LODFV CL2 |(2.8.1)
|                                                    |
- - - - - - - - - - - - - - - - - - - - - - - -  - - - - - - -
END M.
```

Примечание 2.8.1. Последовательность команд LI1 LI2 CL1 CL2 была бы неверна, поскольку CL1 в точке входа забирает со стека все значения (STORE), в том числе и 1, предназначающуюся не для нее (!).

3. ВЫРАЖЕНИЯ

3.1. Индексация словных массивов

```
MODULE M;

VAR x: ARRAY [0..3] OF INTEGER;
    i: INTEGER;

BEGIN
- - - - - - - - - - - - - - - - - - - - - -
|  x[i]:=1;    LGW2 LGW3 LI3 CHKZ LI1 SXW |-- с контролем
|  i:=x[1];    LGW2 LI1  LXW SGW3         |   границ
- - - - - - - - - - - - - - - - - - - - - -

- - - - - - - - - - - - - - - - - - -
|  x[i]:=1;      LGW2 LGW3 LI1 SXW  | -- без контроля границ
|  i:=x[1];      LGW2 LI1  LXW SGW3 |    (3.1.1)
- - - - - - - - - - - - - - - - - - -
END M.
```

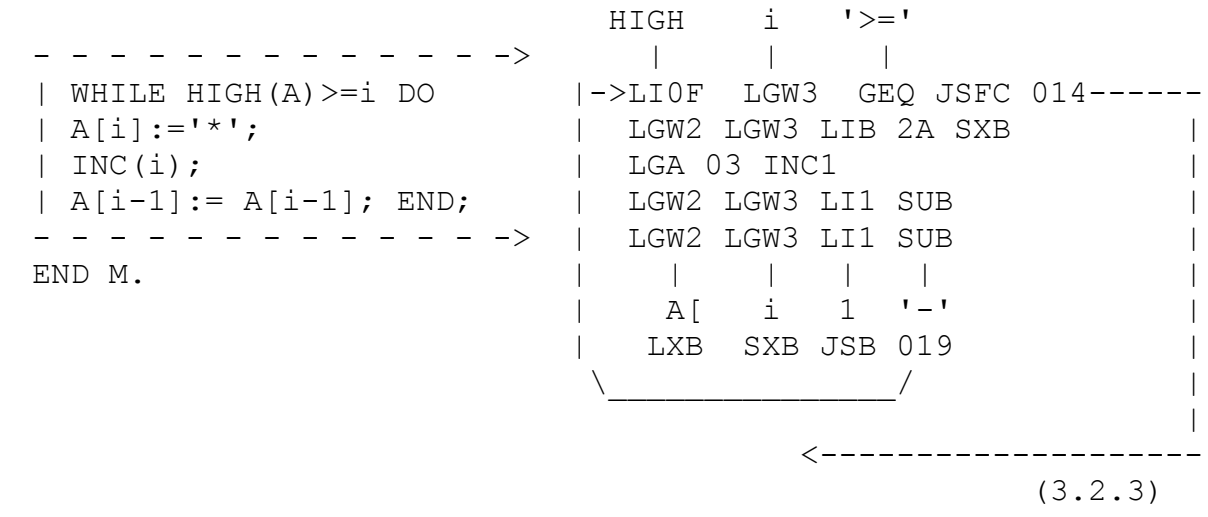
Примечание 3.1.1. Компилятор может использовать константную индексацию, например:

```
- - - - - - - - - - - - - - - - - - - - -
|    i:=x[1]        LGW2 LSW1     SGW2  |
- - - - - - - - - - - - - - - - - - - - -
```

и таким образом, контроль границ становится не нужен.

3.2. Индексация байтовых массивов

```
MODULE M;
    - - - - - - - - - - - - - - - - - - - - - -
VAR | A:ARRAY [0..0Fh] OF CHAR;  LI4 ALLOC SGW2-|--(3.2.1)
    - - - - - - - - - - - - - - - - - - - - - -
      i:INTEGER;                                  (3.2.2)

BEGIN
- - - - - - - - - - - - - - - - - - - - - - - -
|i:=0;                            LI0 SGW3 |
- - - - - - - - - - - - - - - - - - - - - - - -
```



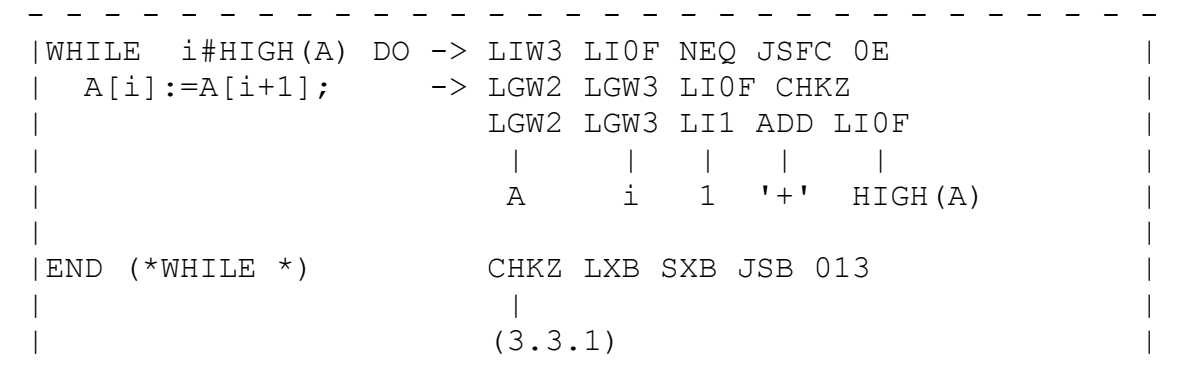
Примечание 3.2.1. В глобальном слове 2 адрес размещенного массива.

Примечание 3.2.2. Литерные массивы пакуются.

Примечание 3.2.3. LXB и SXB работают аналогично LXW и SXW, но читают и пишут соответствующий байт, а не слово. 0 <байтовый адрес> LXB и 0 < байтовый адрес > SXB осуществляют абсолютную байтовую адресацию памяти.

3.3. Индексация байтовых массивов с контролем границ

```
MODULE M;

VAR A:ARRAY [0..0Fh] OF CHAR;
    i:INTEGER;



BEGIN i:=0;
```



Примечание 3.3.1. CHKZ проверяет, лежит ли второй элемент A-стека между 0 и верхним элементом (задающим границу); если это так, то счеркивает границу, иначе возбуждает TRAP(4Ah).

3.4. Проверка принадлежности диапазону

```
MODULE M;
VAR i: INTEGER;
VAR x:[10h..20h];
BEGIN
   - - - - - -  - - - - - - - - - - - - - - - - - - - - - - - - -
  |x:=i;                          LGW2 LIB 10 LIB 20 CHK SGW3 |
  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - | - - - -
END M.                                                        |
                                                          (3.4.1)
```

Примечание 3.4.1. CHK проверяет принадлежность диапазону. Если бы в тексте было x:=13h, проверку произвел бы компилятор и породил код LIB13 SGW2.

3.5. Работа с объектами типа BITSET.

```
MODULE M;
VAR b1, b2:BITSET;
BEGIN
  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  |b1:={}; b2:={1};                LI0  SGW2  LI2  SGW3 |
  |b1:=b1+b2;                      LGW2  LGW3  OR  SGW3 |
  (*     *                                    AND
         /                                    XOR
         -                                    BIC      *)

  |         INCL(b1,2);            LGA 2  LI2  INCL     |
  |         EXCL(b1.2);            LGA 2  LI2  EXCL     |
 (* 2 IN b1                       LI2  LGW2   IN       *)
  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
END M.
```

3.6. Команды ANDJP и ORJP

```
MODULE M;
.
.
BEGIN
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 |IF FALSE AND TRUE THEN END;   LI0  ANDJP 1  LI1  JSFC  |
 |IF TRUE OR FALSE  THEN END;   LI1  ORJP 1   LI0  JSFC  |
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
END M.                                                  (3.6.1)
```

Примечание 3.6.1. Компилятор вместо этого смешного кода, оптимизируя, не породит ничего.

ИНДЕКС-ТАБЛИЦА СИСТЕМЫ КОМАНД