Ордена Ленина
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В. Келдыша
Академии наук СССР

С.А. Романенко

СИСТЕМА ПРОГРАММИРОВАНИЯ РЕФАЛ – 2 ДЛЯ ЕС ЭВМ. ИНТЕРФЕЙС РЕФАЛА И PL/I.

Москва

Ордена Ленина
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
им. М.В.Келдыша
Академии Наук СССР

С.А.Романенко

Система программирования Рефал-2 для ЕС ЭВМ. Интерфейс Рефала и PL/I.

Москва
1987

УДК 681.3.06

Входной язык системы программирования Рефал-2 для ЕС ЭВМ предназначен для обработки символьной информации, представленной в виде выражений, имеющих древовидную структуру. Основными изобразительными средствами Рефала являются сопоставление с образцом, подстановка и рекурсия. В работе описаны средства позволяющие вызывать из программ, написанных на языке PL/I, программы, написанные на рефале, и наоборот.

КЛЮЧЕВЫЕ СЛОВА И ФРАЗЫ: Рефал, обработка символьной информации, функциональное программирование, язык программирования.

С О Д Е Р Ж А Н И Е

В В Е Д Е Н И Е

В данном руководстве описаны средства, позволяющие вызывать из программ, написанных на PL/I, программы, написанные на рефале и наоборот [Р2ОВЯ 1987], [Р2КИП 1987].

Интерфейс между рефалом и PL/I основан на понятии п р о ц е с с а. Процессом называется совокупность из поля зрения и копилки.

В каждый момент времени работает либо PL/I-программа, либо рефал-программа. Когда управление принадлежит PL/I-программе, все процессы приостановлены. Когда работает рефал-программа, работает ровно один из процессов.

PL/I-программа может создавать и уничтожать процессы, запускать их и исследовать причины их остановки.

При обращении из PL/I к рефалу, из PL/I вызывается не рефал-функция, а процесс, т.е. поле зрения, в котором уже находятся обращения к рефал-функциям. Таким образом, содержимое поля зрения определяет, какие именно функции будут вызваны.

PL/I-программа обязана сформировать начальное поле зрения перед обращением к рефалу, а затем, когда рефал вернет ей управление, извлечь нужную информацию из поля зрения.

Если же рефал-программа обращается к PL/I-программе, то PL/I-программа должна сама извлечь нужную информацию из ведущего функционального терма и перед возвратом управления рефал-программе сформировать результат замены.

Такая схема взаимодействия между рефал-программой и программой на императивном (операторном) языке была впервые использована в реализации рефала в рамках мониторной системы "Дубна" БЭСМ-6 [КР 1975], [КПРТР 1975], [БР 1977].

В данной реализации интерфейс между рефалом и PL/I, однако, несколько отличается от интерфейса между рефалом и фортраном в мониторной системе "Дубна". Основное отличие заключается в том, что устранена необходимость возвращаться в середину подпрограммы, после того, как из нее произошел аварийный выход по исчерпанию списка свободной памяти. В данной реализации в случае нехватки памяти, все возвращается

к тому состоянию, которое было перед началом выполнения шага, поэтому повторный вход в подпрограмму ничем не отличается от первоначального. Это облегчает написание первичных функций на PL/I и упрощает их структуру.

Идея возвращать рефал-машину к началу шага в случае нехватки памяти была предложена Арк.В.Климовым и впервые использована А.А.Веденовым в его реализации рефала для ЕС ЭВМ. А.А.Веденовым было предложено также перенести часть информации из таблицы состояния текущего процесса в общий блок REFAL, что упрощает доступ к ней из первичных функций, написанных на PL/I. Эти предложения учтены в данной реализации рефала, поэтому автор считает своим долгом выразить благодарность Арк.В.Климову и А.А.Веденову за высказанные ими замечания и предложения.

I. ОБРАБОТКА ОШИБОК

Во многих случаях рефал-система обнаруживает фатальные ошибки, которые делают бессмысленным продолжение ее работы. В этих случаях печатается диагностическое сообщение и возбуждается состояние ERROR. Если в PL/I-программе не предусмотрена обработка этого состояния с помощью исполняемого предложения "ON", то работа PL/I—программы на этом завершается [ФО 1983].

Для выполнения вышеописаных действий удобно использовать следующую подпрограмму.

П о д п р о г р а м м а RFABE.

Назначение.

Печатает сообщение и возбуждает состояние ERROR.

Об'явление.

```
DECLARE RFABE ENTRY(CHAR(*));
```

Обращение.

```
CALL RFABE(TEXT);
```

Параметры.

TEXT - текст сообщения.

Использование.

Начиная с первой позиции печатается текст "***** РЕФАЛ-АВОСТ *****", а вслед за ним - текст сообщения. Затем возбуждается состояние ERROR.

Исходный текст.

```
RFABE:   PROCEDURE(TEXT) REORDER;
         DECLARE TEXT CHAR(*);
         DECLARE
            SYSPRINT  EXTERNAL FILE PRINT;
         PUT EDIT
            ('***** РЕФАЛ-АВОСТ ***** ',TEXT)(SKIP,A,A);
         SIGNAL ERROR;
         END RFABE;
```

2. ПРЕДСТАВЛЕНИЕ ВЫРАЖЕНИЙ В ПАМЯТИ МАШИНЫ

Во время работы рефал-программы поле зрения, копилка и ящики представлены в виде списков.

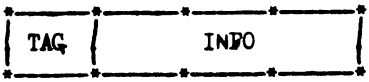
Минимальной, нерасчленимой единицей списка является з в е н о.

Каждое звено занимает три слова памяти, т.е. 12 байтов, выровненных на границу слова. Звено состоит из следующих полей

каждое из которых занимает одно слово.

Поля PREV и NEXT используются для связывания звеньев в линейную последовательность. Поле NEXT всегда содержит адрес следующего звена, а поле PREV - адрес предыдущего звена. При этом, адрес занимает только три младшие байта слова, а старший байт - обязательно нулевой.

Поле CODE состоит из двух подполей: TAG и INFO.



Поле TAG занимает старший байт слова CODE, а поле INFO - младшие три байта слова CODE. Их смысл зависит от того, какому об'екту рефала соответствует данное звено.

Если звено принадлежит полю зрения, копилке или содержимому ящика, оно может изображать один из следующих об'ектов: символ, структурную скобку или функциональную скобку.

Поле TAG содержит признак типа звена. Если его пять старших разрядов - нулевые, звено называется стандартным. Если же хотя бы один из пяти старших разрядов поля TAG отличен от нуля, звено именуется нестандартным.

Если звено стандартное, то поле TAG имеет следующие значения для об'ектов различных типов:

'00000000'B - символ-литера (об'ектный знак),
'00000010'B - символ-метка (имя функции),
'00000100'B - символ-число (макроцифра),
'00000110'B - символ-ссылка,

'00000001'B - левая структурная скобка "(",
'00000011'B - правая структурная скобка ")",
'00000101'B - левая функциональная скобка "<",
'00000111'B - правая функциональная скобка ">".

Ноль в младшем разряде поля TAG означает, что звено содержит символ, а единица - что звено содержит скобку.

С помощью первичных функций, написанных на PL/I или языке ассемблера, можно создавать нестандартные звенья, у которых старшие пять разрядов поля TAG имеют ненулевое значение. Если при этом младший бит поля TAG равен нулю, рефал-система рассматривает такие звенья как некоторые составные символы, отличные от символов-меток, символов-чисел и символов-ссылок. Поле INFO таких звеньев может содержать произвольную комбинацию из двадцати четырех битов. При выводе выражений на печать эти символы изображаются в виде

/FF#FFFFFF/

где FF - значение поля TAG, а FFFFFF - значение поля INFO, выраженные в шестнадцатиричной системе счисления.

Таким образом, помимо четырех стандартных типов символов, существует еще 124 "нестандартных" типа символов. Нестандартные символы невозможно изобразить в виде констант в рефал-программах. Кроме того, они являются специфической особенностью только некоторых реализаций, поэтому их не стоит использовать при создании мобильных рефал-программ.

При порождении нестандартных звеньев следует твердо помнить, что младший разряд поля TAG должен быть нулевым, ибо употребление нестандартных звеньев с четным значением поля TAG может вызвать непредсказуемые последствия.

Смысл поля INFO зависит от типа звена.

* Для структурной скобки поле INFO содержит адрес парной к ней скобки.

* Для символа-литеры поле INFO в младшем байте содержит код соответствующей литеры в коде ДКОИ, а старшие два байта равны нулю.

* Для символа-метки поле INFO содержит адрес точки входа в соответствующую функцию, написанную на рефале или PL/I.

* Для символа-числа поле INFO содержит целое неотрицательное число, соответствующее телу этого символа-числа.

* Для символа-ссылки поле INFO содержит адрес головы соответствующего ящика.

* Для символа нестандартного типа поле INFO содержит произвольную (с точки зрения рефал-системы) информацию.

О функциональных скобках будет сказано ниже.

3. ДОСТУП К ПОЛЯМ ЗВЕНА ИЗ ПРОГРАММЫ НА PL/I

Для того, чтобы извлекать или изменять содержимое звеньев в программах, написанных на PL/I, следует воспользоваться указателями и базированными структурами.

Структура звена описывается следующим образом:

```
DECLARE 1 LINKCB BASED ALIGNED,
   2 PREV     POINTER,
   2 NEXT     POINTER,
   2 CODE     BIT(32);
DECLARE 1 LINKTI BASED ALIGNED,
   2 CHAR8    CHAR(8),
   2 TAG      BIT(8),
   2 INFO     BIT(24);
DECLARE 1 LINKZC BASED ALIGNED,
   2 CHAR8    CHAR(8),
   2 TAGZZ    BIT(24),
   2 INFOC    CHAR(1);
DECLARE 1 LINKP BASED,
   2 CHAR8    CHAR(8),
   2 CODEP    POINTER;
DECLARE 1 LINKF BASED,
   2 CHAR8    CHAR(8),
   2 CODEF    FIXED BINARY(31);
```

Эти описания следует либо непосредственно вставить в текст программы на PL/I, либо включить их в программу с помощью макро-предложения

```
%INCLUDE DCLLINK;
```

Для повышения наглядности программ рекомендуется использовать символические имена для признаков типов. Например, вместо '00000101'B - писать TAGK. Имена признаков описываются следующим образом:

```
DECLARE (
   TAGO   INIT('00000000'B),
   TAGF   INIT('00000010'B),
   TAGN   INIT('00000100'B),
   TAGR   INIT('00000110'B),
   TAGLB  INIT('00000001'B),
   TAGRB  INIT('00000011'B),
   TAGK   INIT('00000101'B),
   TAGD   INIT('00000111'B)
      ) BIT(8) ALIGNED STATIC;
```

Эти описания следует либо непосредственно вставить в текст программы на PL/I, либо включить их в программу с помощью макро-предложения

```
%INCLUDE DCLTAG;
```

Можно, однако, не определять переменные, в которых хранятся значения признаков типов, а использовать средства макропроцессора для подстановки констант вместо символических имен признаков. Для этого перед началом программы следует поместить следующие предложения:

```
%DECLARE
   TAGO CHAR, TAGF CHAR, TAGN CHAR, TAGR CHAR,
   TAGLB CHAR, TAGRB CHAR, TAGK CHAR, TAGD CHAR;
%TAGO='''00000000''B';
%TAGF='''00000010''B';
%TAGN='''00000100''B';
%TAGR='''00000110''B';
%TAGLB='''00000001''B';
%TAGRB='''00000011''B';
%TAGK='''00000101''B';
%TAGD='''00000111''B';
```

Эти предложения можно вставить перед текстом программы либо непосредственно, либо с помощью макро-предложения

```
%INCLUDE DCLMACRO;
```

Пример 1.

Опишем подпрограмму, которая просматривает некоторое выражение и заменяет в нем все вхождения символа-литеры '+' на символ-литеру '-'.

Обращение к подпрограмме должно иметь вид:

```
CALL CHPM(P,Q);
```

где P - указатель на звено, предшествующее выражению, а Q - указатель на звено, следующее за выражением.

Подпрограмма может быть описана следующим образом:

```
CHPM:   PROCEDURE(P,Q);
        DECLARE (P,Q) POINTER;
        %INCLUDE DCLLINK;
        DECLARE  R  POINTER;
        R=P->NEXT;
        DO WHILE (R¬=Q);
           IF (R->TAGZZ=(24)'0'B) &
              (R->INFOC='+')
           THEN R->INFOC='-';
           R=R->NEXT;
        END;
        END CHPM;
```

Пример 2.

Опишем подпрограмму BMATCH, обращение к которой имеет вид:

```
CALL BMATCH(P,Q);
```

и которая просматривает выражение, заключенное между звеньями, на которые указывают P и Q, и все символы-литеры '(' и ')' заменяет на структурные скобки (и). Предполагается, что в исходном выражении символы-литеры '(' и ')' образуют правильную скобочную структуру.

```
PMATCH:  PROCEDURE(P,Q);
         DECLARE (P,Q)  POINTER;
         %INCLUDE  DCLLINK,DCLTAG;
         DECLARE
            NULL     BUILTIN;
         DECLARE (R,RI)  POINTER;
         DECLARE  LASTB  POINTER;
         LASTB=NULL;
         R=P->NEXT;
         DO WHILE (R¬=Q);
            IF
               (R->TAGZZ=(24)'Ø'B) &
               (R->INFOC='(')
            THEN DO;
               R->CODEP=LASTB;
               LASTB=R;
            END; ELSE IF
               (R->TAGZZ=(24)'Ø'B)  &
               (R->INFOC=')')
            THEN DO;
               R->CODEP=LASTB; R->TAG=TAGRB;
               RI=LASTB->CODEP;
               LASTB->CODEP=R; LASTB->TAG=TAGLB;
               LASTB=RI;
            END;
            R=R->NEXT;
         END;
         END BMATCH;
```

П о д п р о г р а м м а RFTPL.

Назначение.

Переставляет указанную часть списка (трансплантат) в другое место.

Об'явление.

```
DECLARE RFTPL
   ENTRY(POINTER, POINTER, POINTER);
```

Обращение.

```
CALL RFTPL(R,P,Q);
```

Параметры.

R - указатель на звено, после которого вставляется трансплантат;

P - указатель на звено, предшествующее трансплантату;

Q - указатель на звено, следующее за трансплантатом.

Использование.

Участок списка, заключенный между P и Q, исключается из списка, звенья P и Q сшиваются. После этого, вынутый участок списка вставляется после звена R.

Исходный текст.

```
RFTPL:   PROCEDURE(R,P,Q) REORDER;
         DECLARE (R,P,Q) POINTER;
         %INCLUDE DCLLINK;
         DECLARE (RI,PI,QI) POINTER;
         PI=P->NEXT;
         IF PI=Q THEN RETURN;
         QI=Q->PREV;
         P->NEXT=Q; Q->PREV=P;
         RI=R->NEXT;
         QI->NEXT=RI: RI->PREV=QI;
         R->NEXT=PI; PI->PREV=R;
         END RFTPL;
```

Ф у н к ц и я LLDUPL.

Назначение.

Позволяет отделить от указанного выражения с левого конца часть, совпадающую с другим указанным выражением.

Об'явление.

```
DECLARE LLDUPL
   ENTRY(POINTER, POINTER, POINTER, POINTER)
   RETURNS(BIT(1) ALIGNED);
```

Обращение.

```
LLDUPL(P,Q,U,V)
```

Параметры.

P - указатель на звено, предшествующее выражению-оригиналу;

Q - указатель на звено, следующее за выражением-оригиналом;

U - указатель на звено, предшествующее выражению, от которого должен быть отделен дубликат;

V - указатель на звено, следующее за выражением-дубликатом (возвращаемый параметр).

Использование.

Пусть между P и Q заключено выражение EX (именуемое оригиналом), а после U начинается некоторое выражение EY.

Если EY начинается с EX, т.е. его можно представить в виде EX EZ, то LLDUPL вырабатывает значение '1'B и устанавливает V на звено, следующее за EX, т.е. на звено, с которого начинается EZ.

Если же EY невозможно представить в виде EX EZ, то LLDUPL вырабатывает значение 'Ø'B, а U - остается без изменения.

Исходный текст.

```
LLDUPL:  PROCEDURE(P,Q,U,V)
              RETURNS(BIT(1) ALIGNED) REORDER;
         DECLARE  (P,Q,U,V)  POINTER;
         %INCLUDE  DCLLINK,DCLTAG;
         DECLARE  (X,Y)  POINTER;
         X=P->NEXT;
         Y=U->NEXT;
         DO WHILE(X¬=Q);
            IF X->CODE¬=Y->CODE  THEN
            IF (X->TAG¬=Y->TAG)|
               ((X->TAG¬=TAGLB)&(X->TAG¬=TAGRB))
            THEN RETURN('Ø'B);
            X=X->NEXT;
            Y=Y->NEXT;
         END;
         V=Y;
         RETURN('1'B);
         END LLDUPL;
```

4. ПРЕДСТАВЛЕНИЕ ФУНКЦИОНАЛЬНЫХ СКОБОК В СПИСКОВОЙ ПАМЯТИ

Функциональные скобки занимают по одному звену каждая. Имя функции хранится в виде символа-метки, сразу же вслед за знаком "<", и занимает отдельное звено.

Звено, соответствующее знаку "<", содержит в поле TAG признак '00000101'B. Звено, соответствующее знаку ">", содержит в поле TAG признак '00000111'B.

Знак ">" в поле INFO содержит адрес парного к нему знака "<". Знак "<" в поле INFO содержит адрес ">", который станет ведущим, после полного вычисления данного функционального терма. Если же такого ">" не существует, знак "<" в поле INFO содержит нуль.

Таким образом, знаки "<" и ">" связаны в список в том порядке, в котором они будут становиться ведущими.

5. ПРЕДСТАВЛЕНИЕ ЯЩИКОВ В СПИСКОВОЙ ПАМЯТИ

Каждому ящику (как статическому, так и динамическому) соответствует звено, именуемое г о л о в о й ящика. Это звено, как и всякое другое, обязательно выравнено по границе слова.

Если ящик динамический, то его именем является символ-ссылка. Символ-ссылка в поле TAG всегда содержит признак '00000110'B, а в поле INFO - адрес головы соответствующего ящика.

Если ящик статический, то его именем является символ-метка, который в поле TAG содержит признак '00000010'B, а в поле INFO - адрес байта, предшествующего голове соответствующего ящика.

Байт, предшествующий голове статического ящика, обязательно содержит константу '01001110'B, что дает возможность проверить, является ли символ-метка именем статического ящика.

Если к статическому ящику не было ни одного обращения, его голова содержит нуль. При первой же попытке что-либо прочитать из статического ящика или записать в него, голова инициализируется: в поля PREV и NEXT заносится адрес самой

головы, что равносильно записи в ящик пустого выражения. Только после этого выполняется операция над ящиком.

Содержимым ящика является некоторое выражение. Начало и конец этого выражения присоединены к голове ящика, т.е. поле NEXT головы ящика содержит адрес первого звена, а поле PREV - адрес последнего звена содержимого ящика. В то же время, первое звено содержимого ящика в поле PREV и последнее звено содержимого ящика в поле NEXT содержат адрес головы ящика. Таким образом, голова ящика вместе с его содержимым представляют собой двусвязный циклический список.

Если содержимое ящика пустое, то голова ящика в полях PREV и NEXT содержит свой собственный адрес.

В поле TAG головы ящика содержится '*00000000*'B. Это поле используется во время сборки мусора (см. п. 15), чтобы помечать ящики, которые нельзя выбрасывать.

Для сборки мусора необходимо иметь возможность просмотреть головы всех ящиков. Поэтому поля INFO в головах ящиков используются для того, чтобы связать все головы в односвязный список в порядке, обратном порядку их порождения. Каждая голова в поле TAG содержит адрес следующей головы. Если голова - последняя в списке, то в поле INFO она содержит нуль.

Головы статических и головы динамических ящиков связаны в два отдельных односвязных списка. Ссылки на эти списки содержатся в общем блоке REFAL (см. п. 8). Динамические ящики заносятся в список в момент создания, статические - в момент инициализации их головы.

6. ПЕЧАТЬ ВЫРАЖЕНИЙ

П о д п р о г р а м м а RFPEXM.

Назначение.

Печатает выражение в метакоде-Б.

Об'явление.

```
DECLARE RFPEXM
        ENTRY(CHAR(*), POINTER, POINTER);
```

Обращение.

```
CALL RFPEXM(TEXT,P,Q);
```

Параметры.

TEXT - текст, который печатается слева от выражения;

P - указатель на звено, предшествующее печатаемому выражению;

Q - указатель на звено, следующее за печатаемым выражением.

Использование.

Участок списка, заключенный между P и Q, следующим образом печатается в метакоде-Б.

Пусть L - длина текста TEXT. Тогда в первых L позициях печатается текст TEXT, а вслед за ним - выражение в метакоде-Б.

Если выражение не поместилось на одной строке, оно переносится на следующие строки. При этом, первые L позиций заполняются пробелами, а продолжение выражения печатается с (L+1)-й позиции.

Замечания.

(1) Предполагается, что длина печатной строки равна 120 литерам.

(2) Если длина текста text превышает 60, то печатаются только первые 60 литер, а остальные - игнорируются.

(3) Если символ-метка не помещается целиком на очередной строке, то он либо целиком переносится на следующую строку, либо по частям печатается на нескольких строках.

П о д п р о г р а м м а RFPEX.

Назначение.

Печатает выражение.

Об'явление.

```
DECLARE  RFPEX
   ENTRY(CHAR(*), POINTER, POINTER);
```

Обращение.

```
CALL RFPEX(TEXT,P,Q);
```

Параметры.

Те же, что и для подпрограммы RFPEXM.

Использование.

Аналогично подпрограмме RFPEXM. Разница состоит только в форме, в которой печатаются выражения. В отличие от RFPEXM, RFPEX не обрамляет цепочки символов-литер апострофами, а составные символы обрамляет апострофами вместо знаков "/".

7. ПРОЦЕССЫ

Данная реализация рефала позволяет создавать программы, различные части которых написаны на рефале и PL/I, и которые тесно взаимодействуют друг с другом.

Программа на PL/I может вызывать программы на рефале, которые, в свою очередь, могут вызывать программы на PL/I и т.д.

В каждый момент времени могут существовать несколько полей зрения и копилок. При этом каждому полю зрения соответствует одна копилка и наоборот.

Совокупность из поля зрения и связанной с ним копилки в дальнейшем именуется п р о ц е с с о м .

Программы на PL/I могут создавать и уничтожать процессы, запускать их и исследовать причины их остановки. Программы, написанные на рефале, не могут управлять процессами непосредственно, но могут делать это вызывая программы на PL/I.

Существует возможность запускать процесс на заданное число шагов. Запуская процесс каждый раз только на один шаг вперед, программа на PL/I может полностью контролировать его работу.

Для управления рефал-процессами предоставляется набор процедур (подпрограмм и функций), вызываемых из программ на PL/I. Эти процедуры хранят глобальную информацию в общем блоке REFAL.

Для каждого процесса имеется т а б л и ц а с о с т о я н и я п р о ц е с с а, в которой содержится вся необходимая информация о процессе.

8. ОБЩИЙ БЛОК REFAL.

Программы на PL/I, взаимодействующие с рефал-программами, используют общий блок REFAL, который должен быть описан следующим образом.

```
DECLARE 1 REFAL EXTERNAL,
   2 CRPREV    POINTER,
   2 CRNEXT    POINTER,
   2 UPSHOT    FIXED BINARY(31),
   2 PREVA     POINTER,
   2 NEXTA     POINTER,
   2 PREVR     POINTER,
   2 NEXTR     POINTER,
   2 CURRST    POINTER,
   2 FLHEAD    POINTER,
   2 SVAR      POINTER,
   2 DVAR      POINTER,
   2 STMNMB    FIXED BINARY(31),
   2 NOSTM     FIXED BINARY(31),
   2 TMMODE    FIXED BINARY(31),
   2 TMINTV    FLOAT DECIMAL(16);
```

Это описание может быть включено в программу на PL/I непосредственно, либо с помощью макро-предложения

```
%INCLUDE DCLREFAL;
```

Первоначальное заполнение общего блока REFAL происходит при вызове подпрограммы RFINIT, либо в момент создания первого процесса, либо в момент первого выделения пространства под списковую память.

Отдельные слова общего блока REFAL имеют следующее содержимое.

CRPREV – адрес последней таблицы состояния.

CRNEXT – адрес первой таблицы состояния.

UPSHOT – развязка (исход) вызова из рефал-программы программы написанной на PL/I. Может принимать целые значения 1, 2 и 3:

1 – вычисление окончено;
2 – отождествление невозможно;
3 – свободная память исчерпана.

PREVA – адрес звена, предшествующего аргументу функции, т.е. звена, содержащего имя функции.

NEXTA – адрес звена, следующего за аргументом, т.е. звена, содержащего знак ">".

PREVR – адрес звена, предшествующего результату замены. Этим звеном является звено, предшествующее знаку "<" перед началом шага.

NEXTR – адрес звена, следующего за результатом замены. Этим звеном является звено, содержащее знак "<".

CURRST – адрес текущей таблицы состояния, т.е. таблицы состояния того рефал-процесса, который вызвал работающую в данный момент программу на PL/I. Если же программа на PL/I вызвана не из рефал-программы, то CURRST=NULL.

FLHEAD – адрес звена, являющегося головой списка свободных звеньев.

SVAR – ссылка на первый элемент односвязного списка статических ящиков. Ящики заносятся в этот список в момент первого обращения и расположены в порядке, обратном к тому, в котором происходили первые обращения. Если не было ни одного обращения к статическим ящикам, SVAR=NULL.

DVAR – ссылка на первый элемент односвязного списка динамических ящиков. Ящики заносятся в этот список в момент создания и расположены в порядке, обратном к тому, в котором они создавались. Если нет ни одного динамического ящика, DVAR=NULL.

STMNMB – номер рефал-предложения, которое было применено при выполнении шага рефал-машины. Предложения в каждой функции нумеруются начиная с 1.

NOSTM – общее количество предложений в той рефал-функции, которая была вызвана при выполнении шага рефал-машины.

TMMODE – признак того, что следует измерить время центрального процессора, израсходованное между запуском и остановкой интерпретатора языка сборки. Может принимать целые значения 0 и 1.

Ø - время измерять не нужно:

1 - время измерять нужно.

TMINTV - время центрального процессора, израсходованное между запуском и остановкой интерпретатора языка сборки, выраженное в микросекундах. Устанавливается только если TMMODE=1.

Назначение и использование различных полей общего блока REFAL более подробно об'ясняется в следующих разделах.

9. ИНИЦИАЛИЗАЦИЯ И ТЕРМИНАЦИЯ РЕФАЛ-СИСТЕМЫ

В начале работы рефал-системы ее необходимо и н и ц и а л и з и р о в а т ь, т.е. привести в рабочее состояние, а в конце работы - т е р м и н и р о в а т ь, т.е. привести в нерабочее состояние и подготовить к последующей инициализации.

П о д п р о г р а м м а RFINIT.

Назначение.

Инициализирует рефал-систему.

Об'явление.

```
DECLARE  RFINIT  ENTRY();
```

Обращение.

```
CALL RFINIT;
```

Параметры.

Параметров нет.

Использование.

Заносятся начальные значения в общий блок REFAL и создается пустой список свободной памяти.

Исходный текст.

```
RFINIT:  PROCEDURE REORDER;
         %INCLUDE DCLREFAL,DCLLINK;
         DECLARE RF#INIT BIT(1) ALIGNED
           EXTERNAL INIT('1'B);
```

```
DECLARE 1 HD STATIC ALIGNED,
   2 (HDPREV,HDNEXT) POINTER,
   2 HDCODE  BIT(32);
DECLARE
   ADDR      BUILTIN,
   NULL      BUILTIN;
RF#INIT='Ø'B;
CRNEXT,CRPREV=ADDR(REFAL);
UPSHOT=1;
CURRST=NULL;
SVAR,DVAR=NULL;
FLHEAD=ADDR(HD);
FLHEAD->NEXT,FLHEAD->PREV=FLHEAD;
FLHEAD->CODE='Ø'B;
NOSTM=Ø;
STMNMB=Ø;
TMMODE=Ø;
TMINTV=Ø;
END RFINIT;
```

П о д п р о г р а м м а RFTERM.

Назначение.

Терминирует рефал-систему.

Об'явление.

```
DECLARE  RFTERM  ENTRY();
```

Обращение.

```
CALL RFTERM;
```

Параметры.

Параметров нет.

Использование.

Рефал-система подготавливается к последующей инициализации.

Исходный текст.

```
RFTERM:  PROCEDURE REORDER;
         DECLARE RF#INIT BIT(1) ALIGNED EXTERNAL;
         RF#INIT='1'B;
         END RFTERM;
```

10. ПРОСТРАНСТВО СПИСКОВОЙ ПАМЯТИ И СПИСОК СВОБОДНЫХ ЗВЕНЬЕВ

Под список выделяются один или несколько связных участков памяти с помощью подпрограммы RFLIST.

При этом все неиспользованные звенья связаны с помощью полей PREV и NEXT в двусвязный циклический список (список свободной памяти). В общем блоке REFAL в поле FLHEAD содержится ссылка на звено - голову этого списка.

При отведении под список нового участка памяти, новые звенья связываются в список и вставляются в конец списка свободной памяти.

П о д п р о г р а м м а RFLIST.

Назначение.

Отдает под список новый связный участок памяти.

Об'явление.

```
DECLARE  RFLIST
         ENTRY((*) FIXED BINARY(31));
```

Обращение.

```
CALL RFLIST(ARRAY);
```

Параметры.

ARRAY - массив слов, отдаваемый под списковую память.

Использование.

Слова, составляющие массив ARRAY, разбиваются на группы по 3 слова и из каждой такой группы создается звено, которое включается в список свободной памяти.

Замечания.

(1) Если длина массива ARRAY не кратна трем, то последние одно или два слова не используются.

(2) Если при обращении к RFLIST, общий блок REFAL еще не инициализирован, производится его инициализация.

Исходный текст.

```
RFLIST:  PROCEDURE(ARRAY) REORDER;
         DECLARE  ARRAY   (*) FIXED BINARY(31);
         %INCLUDE DCLREFAL,DCLLINK;
         DECLARE
            ADDR     BUILTIN,
            HBOUND   BUILTIN,
            LBOUND   BUILTIN;
         DECLARE
            RFINIT   ENTRY();
         DECLARE
            K    FIXED BINARY(15);
         DECLARE
            RF#INIT  BIT(1) ALIGNED EXTERNAL;
         DECLARE
            (P,Q)    POINTER;
         IF RF#INIT THEN CALL RFINIT;
         P=FLHEAD->PREV;
         DO K=LBOUND(ARRAY,1) TO HBOUND(ARRAY,1)-2 BY 3;
            Q=ADDR(ARRAY(K));
            P->NEXT=Q; Q->PREV=P;
            Q->CODE='0'B;
            P=Q;
         END;
         P->NEXT=FLHEAD; FLHEAD->PREV=P;
         END RFLIST;
```

Ф у н к ц и я LRQLK.

Назначение.

Проверяет, содержит ли список свободных звеньев указанное количество звеньев.

Об'явление.

```
DECLARE  LRQLK  ENTRY(FIXED BINARY(31))
              RETURNS(BIT(1) ALIGNED);
```

Обращение.

```
LRQLK(L)
```

Параметры.

L - количество запрашиваемых звеньев.

Использование.

Если список свободных звеньев содержит не меньше, чем L звеньев (не считая головы), функция вырабатывает значение '1'B, в противном случае - значение 'Ø'B.

Исходный текст.

```
LRQLK:   PROCEDURE(L) RETURNS(BIT(1) ALIGNED) REORDER;
         DECLARE  L   FIXED BINARY(31);
         %INCLUDE DCLREFAL,DCLLINK;
         DECLARE
            N  FIXED BINARY(31),
            P  POINTER;
         P=FLHEAD;
         DO N=1 TO L;
            P=P->NEXT;
            IF P=FLHEAD THEN RETURN('Ø'B);
         END;
         RETURN('1'B);
         END LRQLK;
```

П о д п р о г р а м м а RFDEL.

Назначение.

Удаляет указанную часть списка и заносит ее в список свободных звеньев.

Об'явление.

```
DECLARE  RFDEL
     ENTRY(POINTER, POINTER);
```

Обращение.

```
CALL RFDEL(P,Q);
```

Параметры.

P - указатель на звено, предшествующее удаляемой части списка;

Q - указатель на звено, следующее за удаляемой частью

списка.

Использование.

Участок списка, заключенный между звеньями P и Q, исключается из списка, звенья P и Q сшиваются, после чего вынутый участок списка вставляется в конец списка свободной памяти.

Исходный текст.

```
RFDEL:   PROCEDURE(P,Q) REORDER;
         DECLARE (P,Q) POINTER;
         %INCLUDE DCLREFAL,DCLLINK;
         DECLARE (PI,QI,R) POINTER;
         PI=P->NEXT;
         IF PI=Q THEN RETURN;
         QI=Q->PREV;
         R=FLHEAD->PREV;
         P->NEXT=Q; Q->PREV=P;
         QI->NEXT=FLHEAD; FLHEAD->PREV=QI;
         R->NEXT=PI; PI->PREV=R;
         END RFDEL;
```

Функция LCOPY.

Назначение.

Копирует указанное выражение и вставляет копию в указанное место.

Об'явление.

```
DECLARE LCOPY ENTRY(POINTER, POINTER,
     POINTER) RETURNS(BIT(1) ALIGNED);
```

Обращение.

```
LCOPY(R,P,Q)
```

Параметры.

R - указатель на звено, после которого вставляется копия;

P - указатель на звено, предшествующее копируемому выражению;

Q - указатель на звено, следующее за копируемым выражением.

Использование.

Если список свободной памяти содержит достаточное количество звеньев, выражение, заключенное между P и Q, копируется и вставляется после R. При этом функция вырабатывает значение '1'B. В противном случае, функция ничего не делает и вырабатывает значение '0'B.

Исходный текст.

```
LCOPY:   PROCEDURE(R,P,Q) RETURNS(BIT(1) ALIGNED) RECORDER;
         DECLARE  (R,P,Q) POINTER;
         %INCLUDE  DCLREFAL,DCLLINK,DCLTAG;
         DECLARE
            UNSPEC   BUILTIN;
         DECLARE (R1,F,F0,F1,LASTB) POINTER;
         F=FLHEAD;
         F0=P->NEXT;
         DO WHILE(F0¬=Q);
            F=F->NEXT;
            IF F=FLHEAD THEN RETURN('0'B);
            SELECT (F0->TAG);
               WHEN (TAGLB) DO;
                  F->CODE=UNSPEC(LASTB);
                  LASTB=F;
               END;
               WHEN (TAGRB) DO;
                  F->CODE=UNSPEC(LASTB)!TAGRB;
                  UNSPEC(F1)=LASTB->CODE;
                  LASTB->CODE=UNSPEC(F)!TAGLB;
                  LASTB=F1;
               END;
               OTHERWISE DO;
                  F->CODE=F0->CODE;
               END;
            END;
            F0=F0->NEXT;
         END;
         IF FLHEAD=F THEN RETURN('1'B);
         F0=FLHEAD->NEXT;
         F1=F->NEXT;
```

```
FLHEAD->NEXT=F1; F1->PREV=FLHEAD;
R1=R->NEXT;
F->NEXT=R1; R1->PREV=F;
R->NEXT=F0; F0->PREV=R;
RETURN('1'B);
END LCOPY;
```

Функция LINS.

Назначение.

Вставляет указанное число звеньев из списка свободной памяти после указанного звена.

Об'явление.

```
DECLARE LINS ENTRY(POINTER, FIXED BINARY(31))
     RETURNS(BIT(1) ALIGNED);
```

Обращение.

```
LINS(P,L)
```

Параметры.

P - указатель на звено, после которого вставляются звенья;

L - количество вставляемых звеньев.

Использование.

Если в списке свободных звеньев имеется не менее чем L звеньев (не считая головы списка), функция вставляет L звеньев после звена, на которое указывает P. В этом случае значением LINS является '1'B, а все вставленные звенья содержат символ-литеру ' '.

Если в списке свободных звеньев не набирается L звеньев, функция ничего не делает и вырабатывает значение '0'B.

Замечания.

(1) В результате работы LINS ни P, ни L не меняются.

(2) Если L<1, LINS ничего не делает и вырабатывает значение '1'B.

(3) Поля CODE во всех вставленных звеньях содержат значение LOW(3)!!' ', т.е. символ-литеру ' '.

Исходный текст.

```
LINS:   PROCEDURE(P,L) RETURNS(BIT(1) ALIGNED) RECORDER;
        DECLARE
           P  POINTER,
           L  FIXED BINARY(31);
        %INCLUDE DCLREFAL,DCLLIN%;
        DECLARE
           N   FIXED BINARY(31),
           (PI,Q,QI,R) POINTER;
        IF L<1 THEN RETURN('1'B);
        QI=FLHEAD;
        DO N=1 TO L;
           QI=QI->NEXT;
           IF QI=FLHEAD THEN RETURN('0'B);
           QI->TAGZZ=(24)'0'B; QI->INFOC=' ';
        END;
        R=QI->NEXT;
        Q=FLHEAD->NEXT;
        FLHEAD->NEXT=R; R->PREV=FLHEAD;
        PI=P->NEXT;
        QI->NEXT=PI; PI->PREV=QI;
        P->NEXT=Q; Q->PREV=P;
        RETURN('1'B);
        END LINS;
```

II. ТАБЛИЦА СОСТОЯНИЯ ПРОЦЕССА

Каждому процессу соответствует таблица состояния процесса (STATUS TABLE), которая должна быть описана следующим образом.

```
DECLARE 1 ST,
   2 STPREV   POINTER,
   2 STNEXT   POINTER,
   2 STATE    FIXED BINARY(31),
   2 DOT      POINTER,
   2 STEP     FIXED BINARY(31),
   2 STOP     FIXED BINARY(31),
```

```
2 VIEW     POINTER,
2 STORE    POINTER;
```

Описание полей таблицы состояния может быть включено в программу на PL/I с помощью макро-оператора следующим образом:

```
DECLARE I ST %INCLUDE DCLST;
```

Отдельные поля таблицы состояния имеют следующее содержимое.

STPREV - адрес предыдущей таблицы состояния.
STNEXT - адрес следующей таблицы состояния.
STATE - состояние процесса - одно из целых чисел 1, 2, 3, 4, имеющих следующий смысл:
1 - процесс остановился в результате того, что в поле зрения не осталось ни одного знака "<" или из-за того, что перед началом шага оказалось выполнено условие ST.STEP=ST.STOP.
2 - процесс остановился из-за того, что отождествление невозможно.
3 - процесс остановился из-за того, что список свободной памяти содержит слишком мало звеньев, вследствие чего невозможно сформировать результат замены ведущего функционального терма.
4 - процесс находится в активном состоянии, т.е. в середине выполнения шага. В частности это может означать, что процесс вызвал программу на PL/I и ожидает, когда эта программа вернет управление.
DOT - адрес ведущей точки.
В состоянии 1 содержит адрес ведущей точки, если она есть, а если в поле зрения нет ни одного знака ">", DOT=NULL.
В состоянии 2 или 3 содержит адрес ведущей точки.
В состоянии 4 значение DOT не определено.
STEP - счетчик числа шагов. Показывает количество уже завершенных шагов.
STOP - предельный номер шага. Если перед началом

выполнения очередного шага оказывается, что STEP=STOP, то очередной шаг выполняться не будет, а процесс остановится в состоянии 1.

VIEW - адрес головы поля зрения.

STORE - адрес головы копилки.

Все таблицы состояния связаны в двусвязный циклический список, головой которого является общий блок REFAL. Для этого используются поля CRPREV и CRNEXT общего блока REFAL и поля STPREV и STNEXT таблиц состояния процессов.

Поле зрения представляет собой двусвязный циклический список. Головой этого списка является звено, которое в поле NEXT содержит адрес первого звена поля зрения, а в поле PREV - адрес последнего звена поля зрения. Точно так же устроена и копилка.

Функция LEXIST.

Назначение.

Позволяет узнать, является ли ее параметр таблицей состояния какого-нибудь из процессов.

Об'явление.

```
DECLARE LEXIST ENTRY(*)
     RETURNS(BIT(1) ALIGNED);
```

Обращение.

```
LEXIST(ST)
```

Параметры.

ST - таблица состояния процесса.

Использование.

Функция просматривает список таблиц состояния, головой которого является общий блок REFAL, и вырабатывает значение '1'B, если найдет ST в этом списке. В противном случае вырабатывается значение '0'B.

Исходный текст.

```
LEXIST: PROCEDURE(ST) RETURNS(BIT(1) ALIGNED) REORDER;
        %INCLUDE DCLREFAL,DCLLINK;
        DECLARE 1 ST %INCLUDE DCLST;
```

```
DECLARE
   ADDR     BUILTIN;
DECLARE P  POINTER;
P=ADDR(REFAL);
DO UNTIL(P=ADDR(REFAL));
   P=P->NEXT;
   IF P=ADDR(ST) THEN RETURN('1'B);
END;
RETURN('0'B);
END LEXIST;
```

12. СОЗДАНИЕ И УНИЧТОЖЕНИЕ ПРОЦЕССОВ

Функция LCRE.

Назначение.

Создает процесс.

Об'явление.

```
DECLARE  LCRE  ENTRY(*)
     RETURNS(BIT(1) ALIGNED);
```

Обращение.

```
LCRE(ST)
```

Параметры.

ST – таблица состояния процесса.

Использование.

Если список свободных звеньев содержит достаточное количество звеньев, LCRE создает новый процесс и вырабатывает значение '1'B, в противном случае – ничего не делает и вырабатывает значение '0'B.

Замечания.

(1) Если до обращения к LCRE общий блок REFAL не был инициализирован, LCRE предварительно инициализирует его.

(2) У только что созданного процесса поле зрения и копилка – пустые.

Исходный текст.

```
LCRE:    PROCEDURE(ST) RETURNS(BIT(1) ALIGNED) REORDER;
         DECLARE 1 ST %INCLUDE DCLST;
         %INCLUDE DCLREFAL,DCLLINK;
         DECLARE
            ADDR     BUILTIN,
            NULL     BUILTIN;
         DECLARE
            RFINIT   ENTRY(),
            LEXIST   ENTRY(*) RETURNS(BIT(1) ALIGNED),
            RFABE    ENTRY(CHAR(*));
         DECLARE
            RF#INIT  BIT(1)  EXTERNAL;
         DECLARE
            (P,Q,R,FLHEAD1) POINTER;
         IF RF#INIT THEN CALL RFINIT;
         IF LEXIST(ST) THEN
            CALL RFABE('RFCRE:  ПРОЦЕСС УЖЕ СУЩЕСТВУЕТ');
         VIEW=FLHEAD->NEXT;
         IF VIEW=FLHEAD THEN RETURN('0'B);
         STORE=VIEW->NEXT;
         IF STORE=FLHEAD THEN RETURN('0'B);
         FLHEAD1=STORE->NEXT;
         FLHEAD->NEXT=FLHEAD1; FLHEAD1->PREV=FLHEAD;
         VIEW->NEXT,VIEW->PREV=VIEW;
         STORE->NEXT,STORE->PREV=STORE;
         Q=CRPREV;
         STNEXT=ADDR(REFAL);
         CRPREV=ADDR(ST);
         Q->NEXT=ADDR(ST);
         STPREV=Q;
         STATE=1;
         DOT=NULL;
         STEP=0;
         STOP=-1;
         RETURN('1'B);
         END LCRE;
```

П о д п р о г р а м м а RFCANC.

Назначение.

Уничтожает процесс.

Об'явление.

```
DECLARE  RFCANC  ENTRY(*)
     RETURNS(BIT(1) ALIGNED);
```

Обращение.

```
CALL RFCANC(ST);
```

Параметры.

ST - таблица состояния процесса.

Использование.

В результате обращения к RFCANC, процесс, имеющий таблицу состояния ST, уничтожается. При этом память, которая была занята под поле зрения и копилку, освобождается. Освободившиеся звенья присоединяются в конец списка свободных звеньев.

Замечания.

(1) Можно уничтожать процесс, только если он находится в состоянии 1, 2 или 3. Процесс, находящийся в состоянии 4 уничтожать нельзя.

(2) Если при обращении к RFCANC, общий блок REFAL еще не инициализирован, производится его инициализация.

Исходный текст.

```
RFCANC:  PROCEDURE(ST) REORDER;
         DECLARE 1 ST %INCLUDE DCLST;
         %INCLUDE DCLREFAL,DCLLINK;
         DECLARE
            RFINIT   ENTRY(),
            LEXIST   ENTRY(*) RETURNS(BIT(1) ALIGNED),
            RFABE    ENTRY(CHAR(*));
         DECLARE
            RF#INIT  BIT(1) ALIGNED  EXTERNAL;
         DECLARE
            (FLHEAD1,VIEW1,STORE1) POINTER;
         IF RF#INIT THEN CALL RFINIT;
         IF  LEXIST(ST) THEN CALL RFABE
           ('RFCANC: ПРОЦЕСС УЖЕ НЕ СУЩЕСТВУЕТ');
```

```
IF STATE=4 THEN CALL RFABE
   ('RFCANC: ПРОЦЕСС ЕЩЕ РАБОТАЕТ');
STPREV->NEXT=STNEXT; STNEXT->PREV=STPREV;
FLHEADI=FLHEAD->PREV;
VIEWI=VIEW->PREV;
STOREI=STORE->PREV;
FLHEADI->NEXT=VIEW;
VIEW->PREV=FLHEADI;
VIEWI->NEXT=STORE;
STORE->PREV=VIEWI;
STOREI->NEXT=FLHEAD;
FLHEAD->PREV=STOREI;
END RFCANC;
```

13. ЗАПУСК ПРОЦЕССОВ

П о д п р о г р а м м а RFRUN.

Назначение.

Запускает процесс и ждет пока он остановится.

Об'явление.

```
DECLARE RFRUN ENTRY(*)
      OPTIONS(ASSEMBLER INTER);
```

Обращение.

```
CALL RFRUN(ST)
```

Параметры.

ST - таблица состояния процесса.

Использование.

Подпрограмма служит для того, чтобы запустить процесс, имеющий таблицу состояния ST.

После обращения к RFRUN процесс начинает работать, пока либо в поле зрения не останется знаков "<", либо будет невозможно выполнить синтаксическое отождествление, либо окажется, что ST.STEP=ST.STOP, либо в списке свободной памяти окажется недостаточное количество звеньев для формирования результата замены.

Замечания.

(1) После останова процесса выполняется оператор, следующий за вызовом подпрограммы RFRUN.

(2) Если при обращении к RFRUN, ST.DOT=NULL, т.е. в поле зрения нет ни одного знака "<", то после обращения к RFRUN все остается без изменения, за исключением того, что процесс переходит в состояние 1.

(3) К RFRUN можно обращаться рекурсивно.

(4) Если при обращении к RFRUN, ST.STATE=4, то RFRUN ничего не делает и после возврата из RFRUN процесс остается в состоянии 4.

(5) RFRUN написана на языке ассемблера, а не на PL/I, поэтому в ее об'явлении не следует забывать указывать OPTIONS(ASSEMBLER INTER).

(6) При исполнении тех рефал-функций, при компиляции которых была задана опция "C", в STMNMB заносится номер примененного предложения, а в NOSTM - количество предложений в функции. Если же при компиляции рефал-функции опция "C" не была задана, или функция написана не на рефале, STMNMB и NOSTM не изменяются.

(7) Если при обращении к RFRUN TMMODE=1, то перед началом работы RFRUN снимает показания с таймера центрального процессора и устанавливает TMMODE=0, а в конце работы снова снимает показания с таймера и устанавливает TMMODE=1. Затем RFRUN вычисляет время, протекшее между двумя опросами таймера, переводит его в микросекунды и заносит в TMINTV. Если же в момент обращения к RFRUN TMMODE=0, то измерение времени не делается и TMINTV не изменяется.

(8) Таймер центрального процессора имеется не на всех моделях ЕС ЭВМ. Кроме того, его опрос выполняется с помощью привилегированных команд SPT и STPT, что допустимо только если установлен режим расширенного управления (ECMODE).

Функция LINSKD.

Назначение.

Вставляет в поле зрения "<", ">" и имя функции.

Об'явление.

```
DECLARE LINSKD ENTRY(*, ENTRY())
        RETURNS(BIT(1) ALIGNED);
```

Обращение.

```
LINSKD(ST,F)
```

Параметры.

ST - таблица состояния процесса;

F - имя функции.

Использование.

Функция проверяет, что ST.DOT=NULL, т.е. что в поле зрения нет ни одного знака ">".

Затем, если в списке свободной памяти содержится достаточное количество звеньев, она вставляет перед содержимым поля зрения "<F ", а после содержимого поля зрения - ">". Таким образом, если поле зрения содержало выражение EX, оно приобретает вид "<F EX>".

После этого LINSKD завершает работу, причем ее значением является '1'B. Если же звеньев в списке свободных звеньев недостаточно, LINSKD ничего не делает и вырабатывает значение '0'B.

Исходный текст.

```
LINSKD:  PROCEDURE(ST,F) RETURNS(BIT(1) ALIGNED) RECORDER;
         DECLARE 1 ST %INCLUDE DCLST;
         DECLARE F   ENTRY();
         %INCLUDE DCLREFAL,DCLLINK,DCLTAG;
         DECLARE
            BIT      BUILTIN,
            NULL     BUILTIN,
            UNSPEC   BUILTIN;
         DECLARE
            LEXIST   ENTRY(*) RETURNS(BIT(1) ALIGNED),
            RFABE    ENTRY(CHAR(*)),
            LRQLK    ENTRY(FIXED BINARY(31))
                        RETURNS(BIT(1) ALIGNED),
```

```
   LINS     ENTRY(POINTER, FIXED BINARY(31))
               RETURNS(BIT(1) ALIGNED);
DECLARE (P,Q,R) POINTER;
IF ¬LEXIST(ST) THEN CALL RFABE
   ('LINSKD: ПРОЦЕСС ЕЩЕ НЕ СУЩЕСТВУЕТ');
IF DOT¬=NULL THEN CALL RFABE
   ('LINSKD: УЖЕ ЕСТЬ ЗНАКИ "<" В ПОЛЕ ЗРЕНИЯ');
IF  LRQLK(3) THEN RETURN('0'B);
IF LINS(VIEW,2) THEN;
Q=VIEW->PREV;
IF LINS(Q,1) THEN;
P=VIEW->NEXT;
R=P->NEXT;
Q=VIEW->PREV;
P->CODE=TAGK;
Q->CODE=UNSPEC(P)!TAGD;
R->CODE=BIT(UNSPEC(F),32)!TAGF;
DOT=Q;
RETURN('1'B);
END LINSKD;
```

14. ПРИМЕРЫ УПРАВЛЕНИЯ ПРОЦЕССАМИ

Следующая программа создает процесс, запускает его, а затем печатает причину его остановки. Затем печатается поле зрения и копилка в метакоде-Б и процесс уничтожается.

Перед началом каждого шага печатается номер шага и ведущий функциональный терм. В конце каждого шага печатается результат замены ведущего функционального терма. Все это достигается тем, что программа на PL/I запускает рефал-процесс каждый раз только на один шаг.

```
EXMPL1:  PROCEDURE OPTIONS(MAIN);
         DECLARE  ARR(3000) FIXED BINARY(31);
         DECLARE  JOB ENTRY();
         %INCLUDE DCLLINK;
         DECLARE 1 ST %INCLUDE DCLST;
         DECLARE
            SYSPRINT  EXTERNAL FILE PRINT;
```

```
DECLARE
   NULL     BUILTIN,
   UNSPEC   BUILTIN;
DECLARE
   RFINIT   ENTRY(),
   RFLIST   ENTRY((*) FIXED BINARY(31)),
   LCRE     ENTRY(*) RETURNS(BIT(1) ALIGNED),
   LINSKD   ENTRY(*,ENTRY())
               RETURNS(BIT(1) ALIGNED),
   RFRUN    ENTRY(*) OPTIONS(ASSEMBLER INTER),
   RFPEXM   ENTRY(CHAR(*),POINTER,POINTER),
   RFCANC   ENTRY(*),
   RFTERM   ENTRY();
DECLARE
   (PREVK,NEXTD,PK) POINTER;

CALL RFINIT;
CALL RFLIST(ARR);
IF  LCRE(ST) THEN GOTO LACK;
IF  LINSKD(ST,JOB) THEN DO;
   CALL RFCANC(ST);
   GOTO LACK;
END;
DO WHILE ((STATE=1)&(DOT¬=NULL));
   STOP=STEP+1;
   UNSPEC(PK)=(8)'0'B!!(DOT->INFO);
   PREVK=PK->PREV; NEXTD=DOT->NEXT;
   PUT EDIT('ШАГ:',STOP)(SKIP,A,F(10));
   CALL RFPEXM(' BT: ',PREVK,NEXTD);
   CALL RFRUN(ST);
   IF STATE=1 THEN
      CALL RFPEXM(' P3: ',PREVK,NEXTD);
END;
SELECT (STATE);
   WHEN(1) PUT EDIT
      ('ВЫЧИСЛЕНИЕ ОКОНЧЕНО')(SKIP,A);
   WHEN(2) PUT EDIT
      ('ОТОЖДЕСТВЛЕНИЕ НЕВОЗМОЖНО')(SKIP,A);
   WHEN(3) PUT EDIT
      ('СВОБОДНАЯ ПАМЯТЬ ИСЧЕРПАНА')(SKIP,A);
```

```
        END;
        CALL RFPEXM('ПОЛЕ ЗРЕНИЯ: ',VIEW,VIEW);
        CALL RFPEXM('КОПИЛКА: ',STORE,STORE);
        CALL RFCANC(ST);
        CALL RFTERM;
        RETURN;
LACK:   PUT EDIT
           ('НЕ ХВАТАЕТ ПАМЯТИ ДЛЯ ИНИЦИАЛИЗАЦИИ')
           (SKIP,A);
        CALL RFTERM;
        END EXMPL1;
```

Предполагается, что существует рефал-модуль, в котором определена как входная точка метка JOB. Сборка мусора не предусмотрена. Память под список выделяется в массиве ARR.

Теперь рассмотрим программу, которая создает два одновременно существующих процесса и заставляет их работать одновременно, т.е. делая шаги поочередно. Таким образом, первый процесс делает шаг и останавливается, затем второй процесс делает шаг и останавливается и т.д.

Если один из процессов заканчивается раньше другого, он дожидается окончания второго процесса. Это достигается за счет того, что RFRUN ничего не делает, если ST.DOT=NULL.

```
EXMPL2: PROCEDURE OPTIONS(MAIN);
        DECLARE ARR(3000) FIXED BINARY(31);
        DECLARE (FUNC1,FUNC2) ENTRY();
        DECLARE 1 ST1 %INCLUDE DCLST;
        DECLAPE 1 ST2 %INCLUDE DCLST;
        DECLARE
           NULL     BUILTIN;
        DECLARE
           RFLIST   ENTRY((*) FIXED BINARY(31)),
           LCRE     ENTRY(*) RETURNS(BIT(1) ALIGNED),
           LINSKD   ENTRY(*,ENTRY())
                       RETURNS(BIT(1) ALIGNED),
           RFRUN    ENTRY(*) OPTIONS(ASSEMBLER INTER),
           RFCANC   ENTRY(*),
           RFTERM   ENTRY();
        CALL RFLIST(ARR);
```

```
IF LCRE(ST1) THEN;
IF LCRE(ST2) THEN;
IF LINSKD(ST1,FUNC1) THEN;
IF LINSKD(ST2,FUNC2) THEN;
DO WHILE ((ST1.DOT1=NULL)&(ST2.DOT1=NULL));
   ST1.STOP=ST1.STEP+1;
   ST2.STOP=ST2.STEP+1;
   CALL RFRUN(ST1);
   CALL RFRUN(ST2);
END;
CALL RFCANC(ST1)
CALL RFCANC(ST2);
CALL RFTERM;
END EXMPL2;
```

В этой программе предполагается, что список свободной памяти достаточно велик, и что ни один из процессов не может остановиться в состоянии 2 или 3. При желании в программу нетрудно добавить соответствующие проверки.

15. СБОРКА МУСОРА

В тех случаях, когда рефал-программа использует динамические ящики и символы-ссылки, причем некоторые ящики могут становиться ненужными, следует предусмотреть сборку мусора в те моменты, когда исчерпывается список свободных звеньев.

Функция LGCL.

Назначение.

Производит сборку мусора.

Об'явление.

```
DECLARE LGCL ENTRY()
        RETURNS(BIT(1) ALIGNED);
```

Обращение.

```
LGCL()
```

Параметры.

Параметров нет.

Использование.

В результате обращения производится сборка мусора. Сначала помечаются головы всех динамических ящиков, до которых можно добраться из какого-нибудь поля зрения, копилки или статического ящика. Затем все ящики, оставшиеся непомеченными, уничтожаются, а освободившиеся звенья присоединяются к началу списка свободных звеньев.

Если в результате сборки мусора высвободилось хоть одно звено, функция вырабатывает значение 'I'B, в противном случае - значение 'Ø'B.

Исходный текст.

```
LGCL:   PROCEDURE RETURNS(BIT(I) ALIGNED) REORDER;
        DECLARE
           ADDR     BUILTIN,
           NULL     BUILTIN,
           UNSPEC   BUILTIN;

        %INCLUDE DCLREFAL, DCLLINK, DCLTAG;
        DECLARE I ST BASED %INCLUDE DCLST;

        DECLARE
           WAS#COLL BIT(I) ALIGNED,
           PZEPO    POINTER,
           (P,Q,R)  POINTER,
           FLHEADI  POINTER;

        DECLARE I HDVAR,
           2 HDPREV   POINTER,
           2 HDNEXT   POINTER,
           2 HDCODEP  POINTER;

        IF DVAR=NULL THEN RETURN('Ø'B);
        UNSPEC(PZEPO)=(32)'Ø'B;
        WAS#COLL='Ø'B;
```

```
/*
   ОТМЕЧАЕМ ЯЩИКИ, ДОСТИЖИМЫЕ
   ИЗ ПОЛЕЙ ЗРЕНИЯ И КОПИЛОК.
*/
P=CRNEXT;
DO WHILE (P¬=ADDR(REFAL));
   CALL MARK(P->VIEW);
   CALL MARK(P->STORE);
   P=P->NEXT;
END;
/*
   ОТМЕЧАЕМ ЯЩИКИ, ДОСТИЖИМЫЕ
   ИЗ СТАТИЧЕСКИХ ЯЩИКОВ.
*/
IF SVAR¬=NULL THEN DO;
   P=SVAR;
   DO UNTIL(P=PZERO);
      CALL MARK(P);
      P=P->CODEP;
   END;
END;

/* УДАЛЕНИЕ МУСОРА */
HDCODEP=DVAR;
P=ADDR(HDVAR);
Q=DVAR;
DO UNTIL(Q=PZERO);
   IF Q->TAG THEN DO;
      /* ОСТАВЛЯЕМ ЯЩИК */
      Q->TAG=(8)'0'B;
      P=Q;
   END; ELSE DO;
      /* УДАЛЯЕМ ЯЩИК */
      WASCOLL='1'B;
      P->CODEP=Q->CODEP;
      R=Q->PREV;
      FLHEADI=FLHEAD->NEXT;
      R->NEXT=FLHEADI; FLHEADI->PREV=R;
      FLHEAD->NEXT=Q; Q->PREV=FLHEAD;
   END;
```

```
            Q=P->CODEP;
          END;
          IF HDCODEP=PZEPO
          THEN DVAR=NULL;
          ELSE DVAR=HDCODEP;
          RETURN(WAS#COLL);

MARK:     PROCEDURE(ROOT);
          DECLARE ROOT  POINTER;
          DECLARE (H,P,Q,R) POINTER;
          H,P=ROOT;
MRK:      IF P->NEXT=H THEN GOTO UP;
          P=P->NEXT;
          IF P->TAG¬=TAGR THEN GOTO MRK;
          UNSPEC(Q)=(8)'Ø'B!!(P->INFO);
          IF Q->TAG THEN GOTO MRK;
          Q->TAG=(8)'1'B;
          P->CODEP=H;
          Q->PREV=P;
          H,P=Q;
          GOTO MRK;
UP:       IF H=ROOT THEN RETURN;
          Q=H->PREV;
          H->PREV=P;
          R=H;
          H=Q->CODEP;
          Q->CODEP=R; Q->TAG=TAGR;
          P=Q;
          GOTO MRK;
          END MARK;

          END LGCL;
```

П р и м е р .

Можно исправить программу EXMPL1 так, чтобы она собирала мусор в случае, если процесс остановился в состоянии 3. Для этого достатсчно добавить об'явление функции LGCL, а строчку

```
          CALL RFRUN(ST);
```

заменить на последовательность операторов

```
AGAIN:   CALL RFRUN(ST);
         IF STATE=3 THEN
         IF LGCL() THEN GOTO AGAIN;
```

16. ДИНАМИЧЕСКИЙ ЗАХВАТ И ОСВОБОЖДЕНИЕ ПАМЯТИ

П о д п р о г р а м м а RFQMAIN.

Назначение.

Позволяет узнать размер максимального куска памяти, который можно захватить.

Об'явление.

```
DECLARE  RFQMAIN  ENTRY(FIXED BINARY(31))
                  OPTIONS(ASSEMBLER INTER);
```

Обращение.

```
CALL RFQMAIN(L);
```

Параметры.

L - длина (в словах) максимального куска памяти, который можно захватить (возвращаемый параметр).

Использование.

Подпрограмма обращается к операционной системе с помощью условной макрокоманды GETMAIN, захватывает максимальный кусок свободной памяти, определяет его длину и тут же освобождает его. Длина куска (измеряемая в четырехбайтовых словах) выдается в L.

П о д п р о г р а м м а RFGMAIN.

Назначение.

Пытается захватить кусок свободной памяти указанного размера.

Об'явление.

```
DECLARE  RFGMAIN
         ENTRY(FIXED BINARY(31), POINTER)
         OPTIONS(ASSEMBLER INTER);
```

Обращение.

```
CALL RFGMAIN(L,P);
```

Параметры.

L - длина (в словах) запрашиваемого куска памяти;

P - указатель на начало захваченного куска памяти (возвращаемый параметр).

Использование.

Подпрограмма пытается захватить L слов памяти с помощью макрокоманды GETMAIN. Если это удается, в P заносится адрес начала захваченного куска, в противном случае в P заносится значение NULL.

П о д п р о г р а м м а RFFMAIN.

Назначение.

Освобождает кусок памяти указанного размера.

Об'явление.

```
DECLARE  RFFMAIN
      ENTRY(FIXED BINARY(31), POINTER)
      OPTIONS(ASSEMBLER INTER);
```

Обращение.

```
CALL RFFMAIN(L,P);
```

Параметры.

L - длина (в словах) освобождаемого куска памяти;

P - указатель на начало освобождаемого куска памяти.

Использование.

Подпрограмма освобождает кусок памяти длиной в L слов, адрес начала которого находится в P, посредством макрокоманды FREEMAIN.

17. ДИНАМИЧЕСКОЕ УПРАВЛЕНИЕ РАЗМЕРОМ СПИСКОВОЙ ПАМЯТИ

Во многих случаях трудно заранее предсказать, сколько памяти потребуется отвести под список. Поэтому возникает необходимость в динамическом захвате памяти.

Алгоритм захвата памяти должен удовлетворять следующим противоречивым требованиям:

* Чтобы уменьшить накладные расходы, выгодно захватывать память пореже, но большими кусками.

* Чтобы не тратить под список больше памяти, чем необходимо, выгодно захватывать память небольшими кусками по мере необходимости.

* Чтобы уменьшить накладные расходы на сборку мусора, выгодно отвести под список как можно больше памяти, тогда сборка мусора будет происходить реже, и при каждой сборке мусора будет освобождаться больше памяти.

* Если работа происходит в условиях виртуальной страничной памяти, крайне невыгодно занимать под список больше памяти, чем необходимо, ибо это может привести к значительным накладным расходам на листание страниц.

Чтобы удовлетворить этим противоречивым требованиям, в данной реализации рефала предоставляется алгоритм управления списковой памятью, который можно настраивать, задавая значения пяти параметров: MINSZ, LIMSZ, INCRSZ, COLLSZ и REMSZ.

Эти параметры имеют следующий смысл (все размеры задаются в двенадцатибайтовых звеньях).

MINSZ. Минимальный размер списка. В начале работы рефал-программы сразу захватывается под список MINSZ звеньев.

LIMSZ. Ограничение на максимальный размер списка. В процессе работы рефал-программы не разрешается тратить под список более чем LIMSZ звеньев, даже если еще имеется свободная память.

INCRSZ. Количество звеньев, которое добавляется к списку при каждом захвате дополнительной памяти.

COLLSZ. Количество звеньев, которое должно освобождаться после очередной сборки мусора. Если освободилось меньше, чем COLLSZ звеньев, это означает, что к списку следует добавить INCRSZ дополнительных звеньев.

REMSZ. Размер памяти (в звеньях), которая должна остаться не захваченной под список и, таким образом, может использоваться для нужд других программ и операционной системы (например, для создания буферов ввода-вывода и выполнения операций по открытию и закрытию файлов).

Алгоритм управления памятью работает следующим образом. В самом начале, с помощью подпрограммы RFQMAIN, определяется размер доступной свободной памяти. Этот размер, выраженный в количестве звеньев, заносится в переменную AVAILSZ. Затем вычисляется величина MAXSZ, которая представляет собой максимальную величину, до которой разрешается расти списку в процессе работы. MAXSZ, конечно не может превышать размер доступной памяти. Кроме того, следует оставить REMSZ звеньев не занятыми, да еще обеспечить, чтобы MAXSZ не превышало LIMSZ. Поэтому MAXSZ вычисляется по формуле:

```
MAXSZ=MIN(MAX(AVAILSZ-REMSZ,0),LIMSZ);
```

Далее, если MAXSZ<MINSZ, алгоритм считает, что доступная память исчерпана, ибо ее не хватает даже на создание начального списка размером MINSZ. Если же MAXSZ>=MINSZ, под список захватывается MINSZ звеньев.

Во время дальнейшей работы, алгоритм постоянно помнит величину CURRSZ - текущий размер списка. Всякий раз, когда возникает нехватка списковой памяти, делается попытка собрать мусор. Если в результате сборки мусора освободится не менее, чем COLLSZ звеньев, считается, что памяти под список достаточно и работа продолжается. Если же освободится меньше, чем COLLSZ звеньев, алгоритм пытается захватить дополнительную память. Если CURRSZ=MAXSZ, то список уже достиг максимального размера и алгоритм об'являет, что свободная память исчерпана. В противном случае, к списку добавляется дополнительно MIN(INCRSZ,MAXSZ-CURRSZ) звеньев,

и работа продолжается.

Кроме вышеупомянутых переменных, алгоритм распределения памяти использует еще переменную LASTWB - указатель на последний кусок свободной памяти, захваченный им посредством подпрограммы PFGMAIN.

Все вышеупомянутые переменные и константы хранятся в общем блоке REFALSC, который имеет следующую структуру.

```
DECLARE 1 REFALSC EXTERNAL,
   2 MINSZ    FIXED BINARY(31),
   2 LIMSZ    FIXED BINARY(31),
   2 INCRSZ   FIXED BINARY(31),
   2 COLLSZ   FIXED BINARY(31),
   2 REMSZ    FIXED BINARY(31),
   2 LASTWB   POINTER,
   2 CURRSZ   FIXED BINARY(31),
   2 MAXSZ    FIXED BINARY(31),
   2 AVAILSZ  FIXED BINARY(31);
```

Величинам MINSZ, LIMSZ, INCRSZ, COLLSZ, REMSZ присваиваются во время загрузки рефал-системы следующие начальные значения:

```
MINSZ       500
LIMSZ    2000000
INCRSZ      200
COLLSZ      200
REMSZ      1000
```

Это сделано в описании блока REFALSC внутри функции LINCRM (описанной ниже) с помощью атрибутов INITIAL. Эти значения, однако, можно изменить перед началом работы рефал-системы (до первого обращения к LINCRM) и, таким образом, настроить по-своему алгоритм распределения памяти.

Алгоритм распределения памяти реализован с помощью описанных ниже процедур LINCRM и RFTERMM.

Функция LINCRM.

Назначение.

Пытается увеличить размер списка свободной памяти с помощью сборки мусора и захвата дополнительной памяти.

Об'явление.

```
DECLARE  LINCRM ENTRY()
     RETURNS(BIT(1) ALIGNED);
```

Обращение.

```
LINCRM()
```

Параметры.

Параметров нет.

Использование.

Пытается увеличить размер списка свободной памяти с помощью сборки мусора и захвата дополнительной памяти. Если это удается - вырабатывает значение '1'B, в противном случае - значение '0'B.

Замечания.

(1) Если первое обращение к LINCRM происходит до инициализации рефал-системы, и при этом удается создать начальную списковую память, производится предварительная инициализация рефал-системы.

Исходный текст.

```
LINCRM:  PROCEDURE RETURNS(BIT(1) ALIGNED) REORDER;

         DECLARE
            MAX      BUILTIN,
            MIN      BUILTIN,
            NULL     BUILTIN;

         DECLARE 1 REFALSC EXTERNAL,
            2 MINSZ     FIXED BINARY(31) INIT(500),
            2 LIMSZ     FIXED BINARY(31) INIT(2000000),
            2 INCRSZ    FIXED BINARY(31) INIT(200),
            2 COLLSZ    FIXED BINARY(31) INIT(200),
            2 REMSZ     FIXED BINARY(31) INIT(1000),
            2 LASTWB    POINTER          INIT(NULL),
            2 CURRSZ    FIXED BINARY(31),
```

```
   2 MAXSZ     FIXED BINARY(31),
   2 AVAILSZ   FIXED BINARY(31);

DECLARE
   ALLOC#LENGTH    FIXED BINARY(31),
   NEW#B           POINTER STATIC;

DECLARE 1 M#BLOCK BASED(NEW#B),
   2 LENGTH#B   FIXED BINARY(31),
   2 NEXT#B     POINTER,
   2 ARRAY#B
          (ALLOC#LENGTH REFER(M#BLOCK.LENGTH#B))
          FIXED BINARY(31);

%INCLUDE DCLREFAL,DCLLINK;

DECLARE
   RFLIST   ENTRY((*) FIXED BINARY(31)),
   RFTERM   ENTRY(),
   LGCL     ENTRY() RETURNS(BIT(1) ALIGNED),
   RFQMAIN  ENTRY(FIXED BINARY(31))
               OPTIONS(ASSEMBLER INTER),
   RFGMAIN  ENTRY(FIXED BINARY(31), POINTER)
               OPTIONS(ASSEMBLER INTER),
   RFFMAIN  ENTRY(FIXED BINARY(31), POINTER)
               OPTIONS(ASSEMBLER INTER);

DECLARE
   (L#INCR, W#INCR, TOTAL#INCR STATIC)
             FIXED BINARY(31);
DECLARE
   N           FIXED BINARY(31),
   FIRST#FREE  POINTER,
   P           POINTER,
   WAS#COLL    BIT(1) ALIGNED STATIC;

/* LINCRM: ENTRY */
IF LAST#B=NULL THEN DO;
   INCRSZ=MIN(INCRSZ,10922);
     CALL RFQMAIN(TOTAL#INCR);
```

```
            AVAILSZ=TOTAL#INCR/3;
            MAXSZ=MIN(MAX(AVAILSZ-REMSZ,0),LIMSZ);
            CURRSZ=0;
            IF MAXSZ<MINSZ THEN GOTO FALSE;
            L#INCR=MINSZ;
         END; ELSE DO;
            FIRST#FREE=FLHEAD->NEXT;
            WAS#COLL=LGCL();
            N=Q;
            P=FLHEAD->NEXT;
            DO WHILE( (P¬=FIRST#FREE)&(N¬=COLLSZ) );
               N=N+1;
               P=P->NEXT;
            END;
            IF N=COLLSZ THEN GOTO TRUE;
            IF CURRSZ=MAXSZ THEN GOTO FALSE;
            L#INCR=MIN(MAXSZ-CURRSZ,INCRSZ);
         END;
         W#INCR=3*L#INCR;
         TOTAL#INCR=W#INCR+2;
         CALL RFQMAIN(TOTAL#INCR,NEW#B);
         IF NEW#B=NULL THEN GOTO FALSE;
         LENGTH#B=W#INCR;
         NEXT#B=LAST#B;
         LAST#B=NEW#B;
         CURRSZ=CURRSZ+L#INCR;
         CALL RFLIST(ARRAY#B);
TRUE:    RETURN('1'B);
FALSE:   RETURN('0'B);

RFTERMM: ENTRY;
         DO WHILE (LAST#B¬=NULL);
            NEW#B=LAST#B;
            LAST#B=NEXT#B;
            TOTAL#INCR=LENGTH#B+2;
            CALL RFFMAIN(TOTAL#INCR,NEW#B);
         END;
         CALL RFTERM;
         RETURN;
         END LINCRM;
```

П о д п р о г р а м м а RFTERMM.

Назначение.

Освобождает всю память, захваченную в результате обращений к LINCRM и терминирует рефал-систему.

Об'явление.

```
DECLARE RFTERMM ENTRY();
```

Обращение.

```
CALL RFTERMM;
```

Параметры.

Параметров нет.

Использование.

Освобождает всю память, которая оказалась захвачена в результате обращений к LINCRM, а затем - терминирует рефал-систему, обратившись к RFTERM.

Исходный текст.

См. исходный текст функции LINCRM.

П р и м е р .

Можно исправить программу EXMPL1 так, чтобы она использовала сборку мусора и динамический захват памяти под список. Для этого достаточно строчку

```
CALL RFLIST(ARR);
```

заменить на

```
IF LINCRM() THEN GOTO LACK;
```

строчку

```
CALL RFRUN(ST);
```

заменить на

```
AGAIN:   CALL RFRUN(ST);
         IF STATE=3 THEN
         IF LINCRM() THEN GOTO AGAIN;
```

а строчки

```
         CALL RFCANC(ST);
         CALL RFTERM;
```

заменить на

```
         CALL RFCANC(ST);
         CALL RFTERMM;
```

Кроме этого, придется сделать очевидные исправления в описаниях: удалить об'явление массива ARR и добавить об'явления процедур LINCRM и RFTERMM.

18. ВЫЗОВ ПРОГРАММ НА PL/I ИЗ ПРОГРАММ, НАПИСАННЫХ НА РЕФАЛЕ

Рефал-программа может вызывать процедуры, написанные на PL/I. Вызываемая процедура с точки зрения PL/I всегда должна быть подпрограммой без параметров [Р2ОБФ 1986].

Обращение к программе на PL/I делается следующим образом.

Пусть нужно вызвать PL/I-программу PLIPROC. Тогда в рефал-модуле метку PLIPROC следует описать как внешнюю следующим образом:

```
         EXTRN   PLIPROC
```

После этого, как только станет ведущим функциональный терм вида

```
         <PLIPROC EX>
```

вызовется PL/I-процедура PLIPROC.

Если эта процедура ничего не изменяет в поле зрения (например, если она не знает, что ее вызвали из

рефал-программы), то результатом замены будет "пусто".

Если же эта процедура написана в расчете на то, что ее будет вызывать рефал-программа, то результатом замены будет то, что она сформирует между звеньями, на которые указывают PREVR и NEXTR из общего блока REFAL.

19. НАПИСАНИЕ ПЕРВИЧНЫХ ФУНКЦИЙ НА PL/I

Всякая первичная функция, написанная на PL/I, представляет собой подпрограмму без параметров. Эта процедура должна быть, кроме того, процедурой нулевого уровня, т.е. не содержаться внутри каких-либо других процедур.

В тот момент, когда процедура на PL/I получает управление, вызвавший ее процесс находится в состоянии 4, а его таблица состояния является текущей, т.е. ее адрес находится в слове CURRST общего блока REFAL. В словах PREVA и NEXTA общего блока REFAL находятся адреса звеньев, между которыми находится аргумент функции, т.е. содержимое ведущего функционального терма (исключая имя функции, стоящее сразу после "<").

Если аргумент функции пуст, то выполнено

```
(PREVA->NEXT=NEXTA)&(NEXTA->PREV=PREVA)
```

В словах PREVR и NEXTR общего блока REFAL находятся адреса звеньев, между которыми подпрограмма может сформировать результат замены. В момент вызова подпрограммы результат замены пуст, т.е. выполнено

```
(PREVR->NEXT=NEXTR)&(NEXTR->PREV=PREVR)
```

Таким образом, если программа на PL/I не изменяет поле зрения, результатом замены автоматически будет "пусто".

П р е д у п р е ж д е н и е .

Подпрограмма на PL/I не должна изменять PREVA, NEXTA, PREVR и NEXTR, а также содержимое звеньев на которые указывают PREVR, NEXTR и NEXTA, за исключением

полей PREVA->NEXT, NEXTA->PREV, PREVR->NEXT и NEXTR->PREV. Звенья, находящиеся между PREVA (включительно) и NEXTA (исключительно), можно использовать для формирования результата замены.

Поле NEXTR->INFO содержит адрес знака ">", который станет ведущим после окончания данного шага. Первичная функция может использовать эту информацию для порождения функциональных скобок в результате замены. При этом нужно будет только надлежащим образом скорректировать NEXTR->INFO.

Слово UPSHOT общего блока REFAL служит для того, чтобы сообщить рефал-системе, чем завершилась работа первичной функции.

* Если при выходе из процедуры UPSHOT=1, это означает, что шаг выполнен.

* Если при выходе UPSHOT=2, это означает, что аргумент не принадлежит области определения первичной функции.

* Если при выходе UPSHOT=3, это означает, что список свободных звеньев содержит недостаточное количество звеньев, чтобы можно было сформировать результат замены.

Перед вызовом первичной функции устанавливается UPSHOT=1, поэтому приходится устанавливать UPSHOT только в тех случаях, когда требуется присвоить ей значение 2 или 3.

Если в качестве аргумента первичной функции допускаются не любые выражения, то, прежде чем что-либо изменять в поле зрения, следует произвести синтаксический и семантический контроль аргумента. Если обнаружится, что аргумент не годится, следует установить UPSHOT=2 и выйти из процедуры оператором RETURN (или передав управление на END).

Если аргумент допустим, первичная функция начинает формировать результат замены, используя звенья из списка свободной памяти и переставляя в результат замены куски списка из аргумента (заключенные между NEXTR и NEXTA).

Может оказаться, что список свободной памяти содержит недостаточное количество звеньев и их не хватает для

формирования результата замены. В этом случае следует установить UPSHOT=3 и выйти из процедуры оператором RETURN. После этого все звенья, которые к этому моменту уже были вставлены между PREVR и NEXTR будут возвращены в список свободной памяти, процесс остановится в состоянии 3 и управление вернется в программу, запустившую данный процесс с помощью REFRUN. Эта программа должна либо дать дополнительную память под список, либо выполнить сборку мусора, либо уничтожить какие-либо выражения и освободившиеся звенья присоединить к списку свободной памяти. После этого она может перезапустить процесс.

П р е д у п р е ж д е н и е .

При формировании результата замены можно брать звенья не только из списка свободной памяти, но и из аргумента. Однако, портить аргумент можно только тогда, когда первичная функция заведомо знает, что она успешно сможет завершить шаг. Поэтому рекомендуется сначала взять все необходимые звенья из списка свободной памяти и переставить их в результат замены, либо убедиться, что свободных звеньев заведомо хватает и уже после этого что-то брать из аргумента.

Если первичная функция вставляет в результат замены знаки "<" и ">", она может изменять NEXTR->INFO только после того, как убедится, что шаг может быть успешно завершен.

20. ПРИМЕРЫ ПЕРВИЧНЫХ ФУНКЦИЙ НА PL/I

П р и м е р I.

Рассмотрим функцию CPFM, которая просматривает свой аргумент и каждый символ-литеру '+' заменяет на символ-литеру '-' на всех уровнях скобочной структуры. Эта функция может иметь, например, следующее описание на рефале.

```
CPFM    '+'  EA =  '-'          <CPFM EA>
        SX   EA =  SX           <CPFM EA>
        (EX) EA =  (<CPFM EX>)  <CPFM EA>
```

Описание на PL/I имеет вид

```
CPFM:   PROCEDURE;
        %INCLUDE  DCLREFAL,DCLLINK;
        DECLARE  RFTPL  ENTRY(POINTER, POINTER, POINTER);
        DECLARE  P  POINTER;
        CALL RFTPL(PREVR,PREVA,NEXTA);
        P=PREVR->NEXT;
        DO WHILE (P¬=NEXTR);
           IF (P->TAGZZ=(24)'0'B)  &
              (P->INFOC='+')
           THEN P->INFOC='-';
           P=P->NEXT;
        END;
        END CPFM;
```

Пример 2.

Опишем на PL/I первичную функцию CREL, обращение к которой имеет вид

```
        <CREL SX SY>
```

где SX и SY - символы-литеры. Результатом замены является выражение

```
        SZ SX SY
```

где SZ='<', если код SX меньше кода SY, SZ='=', если SX=SY, и SZ='>', если код SX больше кода SY.

```
CREL:   PROCEDURE;
        %INCLUDE DCLREFAL, DCLLINK;
        DECLARE
           LINS     ENTRY(POINTER, FIXED BINARY(31))
                      RETURNS(BIT(1) ALIGNED),
```

```
         RFTPL    ENTRY(POINTER, POINTER, POINTER);

      DECLARE
         (PX,PY,PZ) POINTER;
      PX=PREVA->NEXT;
      IF PX=NEXTA THEN GOTO FAIL;
      PY=PX->NEXT;
      IF PY=NEXTA THEN GOTO FAIL;
      IF (PY->NEXT¬=NEXTA) !
         (PX->TAGZZ¬=(24)'0'B) !
         (PY->TAGZZ¬=(24)'0'B)
      THEN GOTO FAIL;
      IF  LINS(PREVR,1) THEN GOTO LACK;
      PZ=PREVR->NEXT;
      CALL RFTPL(PZ,PREVA,NEXTA);
      PZ->TAGZZ=(24)'0'B;
      SELECT;
         WHEN (PX->INFOC < PY->INFOC) PZ->INFOC='<';
         WHEN (PX->INFOC > PY->INFOC) PZ->INFOC='>';
         OTHERWISE                    PZ->INFOC='=';
      END;
      GOTO DONE;
FAIL: UPSHOT=2; GOTO DONE;
LACK: UPSHOT=3; DONE:
      END CREL;
```

Можно немного упростить эту процедуру, если не брать одно звено из свободной памяти, а использовать звено, которое содержит имя функции (на которое указывает PREVA).

```
CREL: PROCEDURE;
      %INCLUDE DCLREFAL, DCLLINK;
      DECLARE
         LINS     ENTRY(POINTER, FIXED BINARY(31))
                     RETURNS(BIT(1) ALIGNED),
         RFTPL    ENTRY(POINTER, POINTER, POINTER);
      DECLARE
         (PX,PY,PZ) POINTER;
      PX=PREVA->NEXT;
      IF PX=NEXTA THEN GOTO FAIL;
```

```
        PY=PX->NEXT;
        IF PY=NEXTA THEN GOTO FAIL;
        IF (PY->NEXT¬=NEXTA) !
           (PX->TAGZZ¬=(24)'0'B) !
           (PY->TAGZZ¬=(24)'0'B)
        THEN GOTO FAIL;
        PZ=PREVA;
        CALL RFTPL(PREVR,NEXTR,NEXTA);
        PZ->TAGZZ=(24)'0'B;
        SELECT;
           WHEN (PX->INFOC < PY->INFOC) PZ->INFOC='<';
           WHEN (PX->INFOC > PY->INFOC) PZ->INFOC='>';
           OTHERWISE                    PZ->INFOC='=';
        END;
        RETURN;
FAIL:   UPSHOT=2;
        END CREL;
```

Пример 3.

Чтобы разобраться в том, как программа, написанная на PL/I, может порождать знаки "<" и ">" в результате замены, опишем на PL/I функцию, эквивалентную следующей рефал-функции

```
TWOKD    EX '+' EY = <FUNC1 EX> <FUNC2 EY>
```

Эта же функция следующим образом описывается на PL/I.

```
TWOKD:  PROCEDURE;
        %INCLUDE DCLREFAL,DCLLINK,DCLTAG;
        DECLARE
           UNSPEC   BUILTIN;
        DECLARE
           RFTPL    ENTRY(POINTER, POINTER, POINTER),
           LINS     ENTRY(POINTER, FIXED BINARY(31))
                       RETURNS(BIT(1) ALIGNED);
        DECLARE (FUNC1,FUNC2) ENTRY();
        DECLARE
           (P,PK1,PF1,PD1,PK2,PF2,PD2) POINTER;
```

```
P=PREVA;
DO UNTIL ( (P->TAGZZ=(24)'Ø'B) &
           (P->INFOC='+') );
   P=P->NEXT;
   IF P=NEXTA THEN DO;
      UPSHOT=2; RETURN;
   END;
END;
IF ¬LINS(PREVR,6) THEN DO;
   UPSHOT=3; RETURN;
END;
PK1=PREVR->NEXT; PF1=PK1->NEXT; PD1=PF1->NEXT;
PK2=PD1->NEXT; PF2=PK2->NEXT; PD2=PF2->NEXT;
CALL RFTPL(PF1,PREVA,P);
CALL RFTPL(PF2,P,NEXTA);
PF1->CODE=UNSPEC(FUNC1); PF1->TAG=TAGF;
PF2->CODE=UNSPEC(FUNC2); PF2->TAG=TAGF;
PD1->CODEP=PK1; PD1->TAG=TAGD;
PK1->CODEP=PD2; PK1->TAG=TAGK;
PD2->CODEP=PK2; PD2->TAG=TAGD;
PK2->CODE=NEXTR->CODE;
NEXTR->CODEP=PD1; NEXTR->TAG=TAGK;
END TWOKD;
```

Пример 4.

В [Р2ОБФ 1986] описана функция APPLY, обращение к которой из рефал-программы имеет вид:

```
<APPLY EX>
```

где EX - произвольное выражение.

Выполнение этого терма происходит следующим образом. Создаются новое поле зрения и копилка. В поле зрения помещается функциональный терм

```
< EX>
```

а в новую копилку переносится содержимое старой копилки. После этого делается попытка вычислить функциональный терм в

новом поле зрения.

Возможны три исхода этой попытки: нормальный останов (N), останов "отождествление невозможно" (R) и останов "свободная память исчерпана" (S).

В случае N результатом замены будет выражение

```
'N' EY
```

где EY - результат вычисления функционального терма

```
< EX>
```

В случае R результатом замены будет выражение

```
'R' EY
```

где EY - это содержимое того функционального терма, при вычислении которого произошел авост.

В случае S результатом замены будет выражение

```
'S'
```

При любом исходе содержимое новой копилки переносится обратно в старую копилку.

Процедура описывается на PL/I следующим образом.

```
APPLY:   PROCEDURE RECURSIVE REORDER;
         %INCLUDE  DCLREFAL,DCLLINK,DCLTAG;
         DECLARE
            1 ST   %INCLUDE DCLST;
         DECLARE
            1 UPST BASED(CURRST) %INCLUDE DCLST;
         DECLARE
            NULL     BUILTIN,
            UNSPEC   BUILTIN;
         DECLARE
            LCRE     ENTRY(*)  RETURNS(BIT(1) ALIGNED),
            RFCANC   ENTRY(*),
            RFRUN    ENTRY(*) OPTIONS(ASSEMBLER INTER),
            LINCRM   ENTRY() RETURNS(BIT(1) ALIGNED),
```

```
   LINS     ENTRY(POINTER, FIXED BINARY(31))
               RETURNS(BIT(1) ALIGNED),
   RFTPL    ENTRY(POINTER, POINTER, POINTER);
DECLARE  (PX,PK,PD)  POINTER;
IF ¬LINS(PREVR,1) THEN GOTO LACK;
PX=PREVR->NEXT;
IF ¬LCRE(ST) THEN GOTO LACK;
IF ¬LINS(ST.VIEW,2) THEN DO;
   CALL RFCANC(ST);
   GOTO LACK;
END;
PK=ST.VIEW->NEXT;
PD=PK->NEXT;
PK->INFO=(24)'0'B; PK->TAG=TAGK;
PD->CODEP=PK; PD->TAG=TAGD;
ST.DOT=PD;
CALL RFTPL(PK,PREVA,NEXTA);
CALL RFTPL(ST.STORE,UPST.STORE,UPST.STORE);
ST.STOP=-1;
DO UNTIL ((ST.STATE¬=1)|(ST.DOT=NULL));
   CALL RFRUN(ST);
   IF ST.STATE=3 THEN IF LINCRM() THEN ST.STATE=1;
END;
CALL RFTPL(UPST.STORE,ST.STORE,ST.STORE);
SELECT (ST.STATE);
   WHEN (1) DO;
      PX->INFOC='N';
      CALL RFTPL(PX,ST.VIEW,ST.VIEW);
   END;
   WHEN (2) DO;
      PX->INFOC='R';
      PD=ST.DOT;
      UNSPEC(PK)=(8)'0'B||(PD->INFO);
      CALL RFTPL(PX,PK,PD);
   END;
   WHEN (3) DO;
      PX->INFOC='S';
   END;
END;
```

```
        CALL RFCANC(ST);
        RETURN;
LACK:   UPSHOT=3;
        END APPLY;
```

Пример 5.

В [Р20ВЯ 1987] описаны первичные функции для работы со статическими и динамическими ящиками. Они имеют следующее описание на PL/I.

```
NEW:    PROCEDURE REORDER;
        %INCLUDE DCLREFAL,DCLLINK,DCLTAG;
        DECLARE
           ADDR     BUILTIN,
           UNSPEC   BUILTIN;
        DECLARE
           LINS     ENTRY(POINTER, FIXED BINARY(31))
                       RETURNS(BIT(1) ALIGNED),
           RFTPL    ENTRY(POINTER, POINTER, POINTER),
           LCOPY    ENTRY(POINTER, POINTER, POINTER)
                       RETURNS(BIT(1) ALIGNED),
           RFDEL    ENTRY(POINTER, POINTER);
        DECLARE  1 FUNC#SWAP BASED ALIGNED,
           2 SWAP     BIT(8),
           2 HEAD     CHAR(12);
        DECLARE
           N#SWAP   BIT(8) ALIGNED
                       INIT('01001110'B) STATIC;
        DECLARE
           (P,Q,R)  POINTER STATIC;
        DECLARE
           EMP      BIT(1) ALIGNED STATIC;
        DECLARE
           RET      LABEL(GTR1,RDR1,PTR1,WTR1,SWR1);

        /* NEW:  ENTRY */
        IF  LINS(PREVR,1) THEN GOTO LACK;
        R=PREVR->NEXT;
        R->CODEP=PREVA; R->TAG=TAGR;
```

```
          P=NEXTA->PREV;
          P->NEXT=PREVA; PREVA->PREV=P;
          NEXTR->NEXT=NEXTA; NEXTA->PREV=NEXTR;
          PREVA->CODEP=DVAR; PREVA->TAG=(8)'0'B;
          DVAR=PREVA;
          GOTO DONE;
GTR:      ENTRY;
          EMP='1'B;
          RET=GTRI; GOTO ENTER;
GTRI:     CALL RFTPL(PREVR,P,P);
          GOTO DONE;
RDR:      ENTRY;
          EMP='1'B;
          RET=RDRI; GOTO ENTER;
RDRI:     IF ¬LCOPY(PREVR,P,P) THEN GOTO LACK;
          GOTO DONE;
PTR:      ENTRY;
          EMP='0'B;
          RET=PTRI; GOTO ENTER;
PTRI:     Q=P->PREV;
          CALL RFTPL(Q,R,NEXTA);
          GOTO DONE;
WTR:      ENTRY;
          EMP='0'B;
          RET=WTRI; GOTO ENTER;
WTRI:     CALL RFDEL(P,P);
          CALL RFTPL(P,R,NEXTA);
          GOTO DONE;
SWR:      ENTRY;
          EMP='0'B;
          RET=SWRI; GOTO ENTER;
SWRI:     CALL RFTPL(PREVR,P,P);
          CALL RFTPL(P,R,NEXTA);
          GOTO DONE;
ENTER:    R=PREVA->NEXT;
          IF R=NEXTA THEN GOTO FAIL;
          IF EMP & (R->NEXT¬=NEXTA) THEN GOTO FAIL;
```

```
            SELECT(R->TAG);
               WHEN(TAGR) DO;
                  UNSPEC(P)=(8)'Ø'B!!(R->INFO);
               END;
               WHEN(TAGF) DO;
                  UNSPEC(P)=(8)'Ø'B!!(R->INFO);
                  IF P->SWAP¬=NWSWAP THEN GOTO FAIL;
                  P=ADDR(P->HEAD);
                  IF UNSPEC(P->PREV)=(32)'Ø'B THEN DO;
                     P->PREV,P->NEXT=P;
                     P->CODEP=SVAR; P->TAG=(8)'Ø'B;
                     SVAR=P;
                  END;
               END;
               OTHERWISE GOTO FAIL;
            END;
            GOTO RET;
FAIL:       UPSHOT=2;
            GOTO DONE;
LACK:       UPSHOT=3;  DONE:
            END NEW;
```

21. НАПИСАНИЕ ПЕРВИЧНЫХ ФУНКЦИЙ НА ЯЗЫКЕ АССЕМБЛЕРА

Можно писать первичные функции не на PL/I, а на языке ассемблера. При этом нужно только выполнять соглашения о связях, принятые в системе программирования PL/I. Эти соглашения подробно описаны в документации по оптимизирующему компилятору PL/I, к которой необходимо обращаться за полной информацией. В большинстве случаев, однако, будет достаточно приводимых ниже сведений.

Следует различать два основных случая: вызов нерекурсивной ассемблерной программы не вызывающей процедуры, написанные на PL/I, и вызов ассемблерной программы, которая может вызывать другие процедуры, написанные на PL/I. Ниже эти два случая рассматриваются отдельно.

Когда PL/I-программа вызывает ассемблерную программу, ассемблерная программа должна выполнять обычные соглашения о связях, принятые в ОС ЕС и сохранять значения регистров в

области сохранения, выделенной PL/I-программой.

Если ассемблерная программа - не рекурсивна и не вызывает процедуры, написанные на PL/I, она может иметь следующий вид:

```
         ENTRY FUNC
         DC    C' FUNC'
         DC    AL1(4)
FUNC     EQU   *
         STM   14,11,12(13)  ЗАПОМНИЛИ РЕГИСТРЫ
         BALR  11,0          УСТАНОВИЛИ БАЗУ
         USING *,11
         ST    13,SAVEAREA+4 УСТАНОВИЛИ НОВУЮ
         LA    13,SAVEAREA   ОБЛАСТЬ СОХРАНЕНИЯ
         . . . . .
         . . . . .
         L     13,4(13)      ВОССТАНОВЛЕНИЕ РЕГИСТРОВ
         LM    14,11,12(13)
         BR    14            И ВОЗВРАТ.
SAVEAREA DC    20F'0'        80 БАЙТОВ НА ОБЛ. СОХРАНЕНИЯ
*
REFAL    COM
CRPREV   DS    F
NEXT     DS    F
UPSHOT   DS    F   СОСТОЯНИЕ ПРИ ВОЗВРАТЕ ИЗ PL1
PREVA    DS    F   АДРЕС ЗВЕНА ПЕРЕД АРГУМЕНТОМ
NEXTA    DS    F   АДРЕС ЗВЕНА ПОСЛЕ АРГУМЕНТА
PREVR    DS    F   АДРЕС ЗВЕНА ПЕРЕД РЕЗУЛЬТАТОМ
NEXTR    DS    F   АДРЕС ЗВЕНА ПОСЛЕ РЕЗУЛЬТАТА
CURRST   DS    F   АДРЕС ТЕКУЩЕЙ ТАБЛИЦЫ СОСТОЯНИЯ
FLHEAD   DS    F   АДРЕС ГОЛОВЫ СПИСКА СВОБОДНОЙ ПАМЯТИ
SVAR     DS    F   АДРЕС СПИСКА СТАТИЧЕСКИХ ЯЩИКОВ
DVAR     DS    F   АДРЕС СПИСКА ДИНАМИЧЕСКИХ ЯЩИКОВ
STMNMB   DS    F   НОМЕР СРАБОТАВШЕГО ПРЕДЛОЖЕНИЯ
NOSTM    DS    F   КОЛИЧЕСТВО ПРЕДЛОЖЕНИЙ В ФУНКЦИИ
TMMODE   DS    F   РЕЖИМ ИСПОЛЬЗОВАНИЯ ТАЙМЕРА
TMINTV   DS    D   ИНТЕРВАЛ ЗАМЕРЕННЫЙ ТАЙМЕРОМ
```

З а м е ч а н и я.

(1) Ассемблерная подпрограмма не должна изменять значение регистра 12, т.к. оно используется обработчиком программных прерываний в PL/I-системе.

(2) Перед началом подпрограммы следует поместить имя подпрограммы и длину имени. Если длина имени четна, следует добавить перед именем один пробел, чтобы обеспечить выравнивание на границу полуслова. Имя функции используется при печати составных символов-меток.

(3) Первой исполняемой командой должна быть команда STM, т.к. именно по коду этой команды рефал-система распознает, что функция написана не на рефале.

(4) В момент вызова подпрограммы регистры 1 и 5 содержат нулевое значение, т.к. рефал-система предполагает, что вызываемая процедура является подпрограммой без параметров и описана на нулевом уровне вложенности.

Теперь рассмотрим более сложный случай ассемблерной программы, которая может вызывать другие подпрограммы и функции, написанные на PL/I.

Такая подпрограмма должна размещать область сохранения и свои локальные переменные в стеке, поддерживаемом PL/I-системой и иметь следующий вид:

```
         ENTRY FUNC
         DC    C' FUNC'
         DC    AL1(4)
FUNC     EQU   *
         STM   14,11,12(13)
         BALR  11,0
         USING *,11
         LA    0,DSALNGTH-DSA
         L     1,76(13)        R1:=АДРЕС ПЕРВОГО СВОБ. БАЙТА
         ALR   0,1
         CL    0,12(12)        ХВАТАЕТ ЛИ ПАМЯТИ?
         BNH   ENOUGH
         L     15,116(12)      НА ПОДПР. ПЕРЕПОЛНЕНИЯ
         BALR  14,15
```

```
ENOUGH   EQU   *
         ST    0,76(1)          ЗАП. АДР. НАЧ. СВОБ. ПАМЯТИ
         ST    13,4(1)          СВЯЗАЛИ ДВЕ DSA
         ST    1,8(13)
         MVC   72(4,1),72(13)   АДРЕС РАБ. ПР-ВА
         LR    13,1
         MVI   0(13),X'80'      УСТ. ФЛАГОВ В DSA, ЧТОБЫ
         MVI   86(13),X'91'     ОБЕСПЕЧИТЬ ОБРАБОТКУ ОШИБОК
         MVI   87(13),X'C0'     PL/I-ОМ.
         USING DSA,13
         . . . . .
         . . . . .
         L     13,4(13)         ВОЗВРАТ В PL/I.
         LM    14,11,12(13)
         BR    14
*
REFAL    COM
CRPREV   DS    F
NEXT     DS    F
UPSHOT   DS    F   СОСТОЯНИЕ ПРИ ВОЗВРАТЕ ИЗ PL1
PREVA    DS    F   АДРЕС ЗВЕНА ПЕРЕД АРГУМЕНТОМ
NEXTA    DS    F   АДРЕС ЗВЕНА ПОСЛЕ АРГУМЕНТА
PREVR    DS    F   АДРЕС ЗВЕНА ПЕРЕД РЕЗУЛЬТАТОМ
NEXTR    DS    F   АДРЕС ЗВЕНА ПОСЛЕ РЕЗУЛЬТАТА
CURRST   DS    F   АДРЕС ТЕКУЩЕЙ ТАБЛИЦЫ СОСТОЯНИЯ
FLHEAD   DS    F   АДРЕС ГОЛОВЫ СПИСКА СВОБОДНОЙ ПАМЯТИ
SVAR     DS    F   АДРЕС СПИСКА СТАТИЧЕСКИХ ЯЩИКОВ
DVAR     DS    F   АДРЕС СПИСКА ДИНАМИЧЕСКИХ ЯЩИКОВ
STMNMB   DS    F   НОМЕР СРАБОТАВШЕГО ПРЕДЛОЖЕНИЯ
NOSTM    DS    F   КОЛИЧЕСТВО ПРЕДЛОЖЕНИЙ В ФУНКЦИИ
TMMODE   DS    F   РЕЖИМ ИСПОЛЬЗОВАНИЯ ТАЙМЕРА
TMINTV   DS    D   ИНТЕРВАЛ ЗАМЕРЕННЫЙ ТАЙМЕРОМ
*
* ДИНАМИЧЕСКАЯ ОБЛАСТЬ ПАМЯТИ,
* ВЫДЕЛЯЕМАЯ В СТЕКЕ PL1
*
```

```
DSA       DSECT
          DS    11D      ОБЛАСТЬ СОХРАНЕНИЯ.
          DS    . . .    ОПИСАНИЕ ЛОКАЛЬНЫХ
          DS    . . .    ПЕРЕМЕННЫХ.
DSALNGTH  DS    0D       ДЛИНА - КРАТНА 8.
```

22. РАБОТА С ТАЙМЕРОМ ЧЕРЕЗ МАКРОКОМАНДЫ ОС ЕС

В этом разделе описаны подпрограммы, с помощью которых можно устанавливать, опрашивать и сбрасывать таймер с помощью макрокоманд STIMER и TTIMER. Эти макрокоманды правильно исполняются как в том случае, когда программа работает под управлением ОС ЕС, так и в том случае, когда программа работает под управлением CMS или ЩО.

П о д п р о г р а м м а RFOSTMS.

Назначение.

Подпрограмма заносит в таймер интервал времени, выраженный в единицах таймера.

Об'явление.

```
DECLARE  RFOSTMS
         ENTRY(FIXED BINARY(31))
         OPTIONS(ASSEMBLER INTER);
```

Обращение.

```
CALL RFOSTMS(T);
```

Параметры.

T - интервал времени, выраженный в единицах таймера.

Использование.

Подпрограмма устанавливает в таймер с помощью макрокоманды STIMER интервал времени T.

Замечания.

(1) Одна единица таймера составляет 26 микросекунд.

Исходный текст.

```
*
* ПОДПРОГРАММЫ ДЛЯ РАБОТЫ С ТАЙМЕРОМ ОС ЕС.
*
RF@OSTM  CSECT
         PRINT NOGEN
         ENTRY RFOSTMS,RFOSTMG,RFOSTMC
*
         DC    C'RFOSTMS',AL1(7)
RFOSTMS  STM   14,3,12(13)
         BALR  BASE,0
         USING *,BASE
         ST    13,SA+4
         LA    13,SA
         L     T,0(1)
*
         L     0,0(T)
         ST    0,INTVL
         STIMER TASK,TUINTVL=INTVL
         B     DONE
*
         DC    C'RFOSTMG',AL1(7)
RFOSTMG  STM   14,3,12(13)
         BALR  BASE,0
         USING *,BASE
         ST    13,SA+4
         LA    13,SA
         L     T,0(1)
*
         TTIMER
         ST    0,0(T)
         B     DONE
*
         DC    C'RFOSTMC',AL1(7)
RFOSTMC  STM   14,3,12(13)
         BALR  BASE,0
         USING *,BASE
         ST    13,SA+4
         LA    13,SA
```

```
         L     T,Ø(1)
*
         TTIMER CANCEL
         ST    Ø,Ø(T)
         B     DONE
*
DONE     L     13,4(13)
         LM    14,3,12(13)
         BR    14
**
** ПЕРЕМЕННЫЕ И КОНСТАНТЫ.
**
INTVL    DS    F
SA       DS    2ØF'Ø'
**
** РЕГИСТРЫ.
**
T        EQU   2
BASE     EQU   3
*
         END
```

П о д п р о г р а м м а RFOSTMG.

Назначение.

Подпрограмма читает текущие показания таймера, выраженные в единицах таймера.

Об'явление.

```
DECLARE  RFOSTMG
         ENTRY(FIXED BINARY(31))
         OPTIONS(ASSEMBLER INTER);
```

Обращение.

```
CALL RFOSTMG(T);
```

Параметры.

T - интервал времени, выраженный в единицах таймера (возвращаемый параметр).

Использование.

Подпрограмма с помощью макрокоманды TTIMER читает в T время, оставшееся до конца временного интервала.

Замечания.

(1) Одна единица таймера составляет 26 микросекунд.

Исходный текст.

См. описание подпрограммы RFOSTMS.

П о д п р о г р а м м а RFOSTMC.

Назначение.

Подпрограмма читает текущие показания таймера, выраженные в единицах таймера.

Об'явление.

```
DECLARE  RFOSTMC
         ENTRY(FIXED BINARY(31))
         OPTIONS(ASSEMBLER INTER);
```

Обращение.

```
CALL RFOSTMC(T);
```

Параметры.

T – интервал времени, выраженный в единицах таймера (возвращаемый параметр).

Использование.

Подпрограмма с помощью макрокоманды TTIMER читает в T время, оставшееся до конца временного интервала, и сбрасывает таймер.

Замечания.

(1) Одна единица таймера составляет 26 микросекунд.

Исходный текст.

См. описание подпрограммы RFOSTMS.

23. РАБОТА С ТАЙМЕРОМ ЦЕНТРАЛЬНОГО ПРОЦЕССОРА

Если программа работает под управлением CMS или ЩО и при этом для виртуальной машины установлен режим расширенного управления (ECMODE), можно устанавливать и опрашивать таймер центрального процессора с помощью привилегированных команд SPT и STPT. В этом разделе описаны подпрограммы, которые позволяют производить измерения времени с помощью таймера центрального процессора [CBM 1985].

П о д п р о г р а м м а RFTMMAX.

Назначение.

Подпрограмма устанавливает в таймер максимально возможное значение.

Об'явление.

```
DECLARE  RFTMMAX ENTRY()
      OPTIONS(ASSEMBLER INTER);
```

Обращение.

```
CALL RFTMMAX;
```

Параметры.

Параметров нет.

Использование.

Подпрограмма с помощью команды SPT заносит в таймер максимально возможное значение.

Исходный текст.

```
*
* ПОДПРОГРАММЫ ДЛЯ РАБОТЫ С ТАЙМЕРОМ НА МАШИНАХ "РЯД-2".
*
RF@TM    CSECT
         PRINT NOGEN
         ENTRY RFTMMAX,RFTMGET
*
         DC    C'RFTMMAX',AL1(7)
RFTMMAX  STM   14,3,12(13)
         BALR  BASE,0
         USING *,BASE
```

```
         ST    I3,SA+4
         LA    I3,SA
*
* В ТАЙМЕР - МАКСИМАЛЬНОЕ ЗНАЧЕНИЕ.
*
         SPT   FIXMAXT
         B     DONE
*
         DC    C'RFTMGET',AL1(7)
RFTMGET  STM   I4,3,I2(I3)
         BALR  BASE,Ø
         USING *,BASE
         ST    I3,SA+4
         LA    I3,SA
         L     X,Ø(I)
*
* ТЕКУЩЕЕ ПОКАЗАНИЕ ТАЙМЕРА.
* ПЕРЕВОДИМ ИЗ FIXED ВО FLOAT.
*
         STPT  FIXCURRT
         MVC   FLCURRT+1(7),FIXCURRT
         MVI   FLCURRT,X'4D'
*
* НАХОДИМ ПРИРАЩЕНИЕ ВРЕМЕНИ.
*
         LD    Ø,FLMAXT
         SD    Ø,FLCURRT
         STD   Ø,Ø(X)
         B     DONE
*
DONE     L     I3,4(I3)
         LM    I4,3,I2(I3)
         BR    I4
**
** ПЕРЕМЕННЫЕ И КОНСТАНТЫ.
**
FIXMAXT  DS    ØD
         DC    X'8FFFFFFFFFFFF000'
FLMAXT   DS    ØD
         DC    X'4D8FFFFFFFFFFFFØ'
```

```
FIXCURRT DS    D
FLCURRT  DS    D
SA       DS    2ØF'Ø'
**
** РЕГИСТРЫ.
**
X        EQU   2
BASE     EQU   3
*
         END
```

П о д п р о г р а м м а RFTMGET.

Назначение.

Подпрограмма позволяет узнать, сколько времени прошло после последнего обращения к подпрограмме RFTMMAX.

Об'явление.

```
DECLARE  RFTMGET
         ENTRY(FLOAT DECIMAL(16))
         OPTIONS(ASSEMBLER INTER);
```

Обращение.

```
CALL RFTMGET(T);
```

Параметры.

T - интервал времени, прошедший после последнего обращения к RFTMMAX, выраженный в микросекундах (возвращаемый параметр).

Использование.

Подпрограмма с помощью команды STPT снимает с таймера его текущие показания и вычисляет время, прошедшее после последнего обращения к подпрограмме RFTMMAX.

Исходный текст.

См. описание подпрограммы RFTMMAX.

ЛИТЕРАТУРА

[БР 1977]

Базисный рефал и его реализация на вычислительных машинах. – М.: ЦНИПИАСС, 1977, вып. V–40. – 258 с.

[КПРТР 1975]

Ан.В.Климов, Л.В.Проворов, С.А.Романенко, Е.В.Травкина. Рефал в мониторной системе "Дубна" БЭСМ–6. Входной язык компилятора и запуск программ. – М.: ИПМ АН СССР, 1975, препринт N 8. – 71 с.

[КР 1975]

Ан.В.Климов, С.А.Романенко. Рефал в мониторной системе "Дубна" БЭСМ–6. Интерфейс рефала и фортрана. – М.: ИПМ АН СССР, 1975. – 86 с.

[Р2КИП 1987]

С.А.Романенко. Система программирования Рефал–2 для ЕС ЭВМ. Компиляция и исполнение рефал–программ под управлением ПДО СВМ. – М.: ИПМ им.М.В.Келдыша АН СССР, 1987. – 35 с.

[Р2ОБФ 1986]

Ан.В.Климов, С.А.Романенко. Система программирования Рефал–2 для ЕС ЭВМ. Описание библиотеки функций. – М.: ИПМ им.М.В.Келдыша АН СССР, 1986, препринт N 200. – 38 с.

[Р2ОВЯ 1987]

Анд.В.Климов, С.А.Романенко. Система программирования Рефал–2 для ЕС ЭВМ. Описание входного языка. – М.: ИПМ им.М.В.Келдыша АН СССР, 1987. – 52 с.

[СВМ 1985]

Система виртуальных машин для ЕС ЭВМ. Справочник / Под ред. Э.В.Ковалевича. – М.: Финансы и статистика, 1985. – 360 с.

[ФО 1983]

Г.Д.Фролов, В.Ю.Олонин. Практический курс программирования на языке PL/I. - М.: Наука. Главная редакция физико-математической литературы, 1983. - 384 с.

АЛФАВИТНЫЙ УКАЗАТЕЛЬ ПРОЦЕДУР

С.А. Романенко " Система программирования Рефал- 2 для ЕС ЭВМ. Интерфейс Рефала и PL/1.
Редактор Вс.С. Штаркман. Корректор А.В. Климов.

Подписано к печати 8.07.87г. № Т-16147. Заказ № 310.
Формат бумаги 60X90 1/16. Тираж 300 экз.
Объём 3,5 уч.-изд.л. Цена 30 коп.

055 (02)2

Отпечатано на ротапринтах в Институте прикладной математики АН СССР
Москва, Миусская пл. 4.

Ссылки на издание рекомендуется делать по следующей форме: и.о., фамилия, название, препринт Ин. прикл. матем. им. М.В. Келдыша АН СССР, год, №.

Распространение: препринты института продаются в магазинах Академкниги г. Москвы, а также распространяются через Библиотеку АН СССР в порядке обмена.

Адрес: СССР, 125047, Москва-47, Миусская пл. 4, Институт прикладной математики им. М.В. Келдыша АН СССР, ОНТИ.

The references should be typed by the following form: initials, name, title, preprint, Inst.Appl.Mathem., the USSR Academy of Sciences, year, N(number).

Distribution. The preprints of the Keldysh Institute of Applied Mathematics, the USSR Academy of Sciences are sold in the bookstores "Academkniga", Moscow and are distributed by the USSR Academy of Sciences Library as an exchange.

Adress: USSR, I25047, Moscow A-47, Miusskaya Sq.4, the Keldysh Institute of Applied Mathematics, Ac.of Sc., the USSR, Information Bureau.

Цена 30 коп.