Oldies... But Goldies!

СИСТЕМА PL/1-КТ

ОПИСАНИЕ ЯЗЫКА, КОМПИЛЯТОРА И УТИЛИТ

Руководство по программированию на языке PL/1 для 64-разрядной Windows

Королев 2021


ПРОГРАММИРОВАНИЕ
НА ЯЗЫКЕ
ПЛ/1

Оглавление

ФИЛОСОФИЯ

Обычно книги начинаются с введения, а не с какой-то «философии». В данном случае настоящее введение расположено в следующем разделе, а этот раздел адресован тем, кто хотел бы понять, о чем вообще речь и почему это средство программирования еще используется.

Для собственно знакомства с реализацией, этот раздел можно смело пропустить.

В 1966 году в США вышла статья Ландина «Следующие 700 языков программирования», где он рассматривал языки до 1966 года и рассуждал о следующих 700. Спустя 55 лет легко можно написать подобную статью и назвать ее «Следующие 7000 языков программирования» потому, что языки продолжают множиться до бесконечности.

Причем это объективный процесс, поскольку отрасли ИТ развиваются, а области их применения расширяются. Значит, требуется как развитие существующих средств программирования, так и создание новых.

И вот, на фоне этого бесконечного потока новых языков некоторые «старые» языки остаются востребованными и даже «переживают» новые. Правда, как правило, и «старые» языки тоже меняются, то есть развиваются.

Но это объективно необходимое развитие не проходит гладко, иначе не появлялись бы статьи с заголовками вроде «ООП - катастрофа на триллион долларов». Потому, что все время хотелось найти одну идею, которая сразу же решила бы проблемы программирования. То это был отказ от GOTO, то ООП, то абстрактные типы, то функциональное программирование, то доказательство правильности программ. И каждый раз это приводило к завышенным ожиданиям, которые, по большому счету, не оправдывались. На этот процесс к тому же накладывалась мода, реклама, маркетинговые войны и желание корпораций взять индустрию инструментальных средств программирования под свой контроль.

С точки зрения моды и рекламы языку PL/1 не повезло: в свое время Эдсгер В. Дейкстра сыграл роль яростного аятоллы и издал фетву, в которой он объявил PL/1 «смертельной болезнью». Эти ядовитые шипы религиозного пыла повредили статусу многих языков и помогли более примитивным языкам, но с гораздо более простыми компиляторами, такими как Паскаль и Си, получить и место, и аудиторию.

Собственно, та самая длинная цитата из выступления Дейкстры:

…«я должен упомянуть PL/1, язык программирования, документация которого обладает устрашающими размерами и сложностью. Использование PL/1 больше всего напоминает полет на самолете с 7000 кнопок, переключателей

и рычагов в кабине. Я совершенно не представляю себе, как мы можем удерживать растущие программы в голове, когда из-за своей полнейшей вычурности язык программирования - наш основной инструмент, не так ли! - ускользает из-под контроля нашего интеллекта.

И если мне понадобится описать влияние, которое PL/1 может оказывать на своих пользователей, ближайшее сравнение, которое приходит мне в голову, - это наркотик. Я помню лекцию в защиту PL/1, прочитанную на симпозиуме по языкам высокого уровня человеком, который представился одним из его преданных пользователей. Но после похвал в адрес PL/1 в течение часа он умудрился попросить добавить к нему около пятидесяти новых "возможностей", не предполагая, что главный источник его проблем кроется в том, что в нем уже и так слишком уж много "возможностей". Выступающий продемонстрировал все неутешительные признаки пагубной привычки, сводящейся к тому, что он впал в состояние умственного застоя и может теперь только просить еще, еще, еще... Если Фортран называют детским расстройством, то PL/1, с его тенденциями роста подобно опасной опухоли, может оказаться смертельной болезнью».

И вот прошло полвека. А число «кнопок, переключателей и рычагов в кабине» воздушных лайнеров все не уменьшается. Кстати, странное сравнение кабины самолета с языком программирования. Логичнее было бы сравнивать его, к примеру, с набором обычных инструментов. Тут ведь тоже не наблюдается снижения их количества в одном наборе, нужно знать и помнить какой инструмент для чего, хотя можно было бы «не увеличивать сложность» и в пределе все свести, например, к одному молотку.

В результате и этой почти религиозной критики PL/1 не стал массовым языком, постепенно оброс мифами и анекдотами, причем, сейчас эти мифы о его невероятной сложности часто повторяют люди, никогда не имевшие дело с PL/1. А постоянное набирание сложности всеми современными, используемыми на практике языками, наоборот, не замечается – «это другое».

Часто PL/1 продолжают ассоциировать исключительно с IBM 360 (или с ЕС ЭВМ) и с какими-нибудь давно забытыми DD-операторами и индексно-последовательными файлами. Отчасти виновата в этом и сама IBM, изначально не планировавшая переносить свой компилятор в другие среды. А когда все-таки сделала это, оказалось, что поезд давно ушел. Но, к счастью, реализацией PL/1 занимались не только в IBM и одна из «не родных» реализаций для x86 оказалась весьма удачной. Поэтому за весь период развития персональных компьютеров язык никуда не пропадал и даже продолжал развиваться.

Существует большое число задач, которые требуют решения путем составления соответствующих программ. Но для них совсем необязательны, а часто и бессмысленны классы и методы, наследование и инкапсуляция, кортежи и монады. Например, это класс «инженерных» задач. И для этого класса

существует ряд своих, удобных средств, начиная с того же Фортрана. В этом ряду находится и PL/1.

Реализация этого языка очень хорошо «ложится» на систему команд x86, поскольку при ее создании в конце 70-х аппаратная поддержка языков высокого уровня подразумевала в первую очередь поддержку именно его возможностей. Например, сравнение строк в лексикографическом порядке, работа с битами, точные вычисления в двоично-десятичном коде нашли прямое отражение в инструкциях процессора, что способствовало повышению эффективности программ на PL/1 в целом.

И сам факт весьма успешного многолетнего использования этого языка в задачах космической отрасли и на современных компьютерах свидетельствует о том, что его недостатки сильно преувеличены или даже выдуманы. PL/1 был и остается отличным средством, хотя, да, не без недостатков. Но с недостатками можно бороться путем развития и совершенствования, как языка, так и компилятора, что, собственно, и происходило на протяжении многих лет.

В свое время вышла прекрасная книга Л.Ф. Штернберга «Ошибки программирования и приемы работы на языке ПЛ/1» («Машиностроение» Москва, 1993), посвященная конструктивной (а не огульной) критике языка. В какой-то мере, она явилась и планом доработок.

ВВЕДЕНИЕ

В данном руководстве описана система программирования на расширенном подмножестве языка PL/1 для среды Windows и архитектуры процессоров x86-64. Сам язык претерпел некоторые изменения и расширения, относительно первоначального стандарта (стандарт X3.74, подмножество общего назначения «G»).

Этот язык, благодаря усилиям фирмы IBM, с середины 60-х создавался как универсальный язык для всех (!) задач, причем на этот, невыполнимый, как сегодня ясно, проект было потрачено много сил и средств. Впрочем, IBM вообще не собиралась конкурировать с другими языками, считая, что ее PL/1 и IBM 360 – это и вершина, и конец развития средств программирования.

Разумеется, никакой вершины и конца не случилось. Поэтому в 1980 году американский специалист д-р Фрайбургхаус задался целью скорректировать язык PL/1 и сделать из него компактное подмножество, оставив удачные находки языка и убрав неудачные или просто громоздкие.

Он провел ряд семинаров и совещаний по данному вопросу в Калифорнии, пригласив на них ведущего специалиста по микропроцессорам Гарри Килдэла. Дело в том, что Фрайбургхаус, услышав, что Килдэл разработал язык PL/M, ошибочно решил, что тот уже создал компилятор PL/1 для микрокомпьютеров и хотел услышать его мнение по поводу сокращений в языке.

Но у Килдэла не было компилятора PL/1, а его язык PL/M был, по сути, разновидностью ассемблера. Получилось наоборот, послушав Фрайбургхауса о сокращениях в языке, Килдэл решил создать компилятор для Intel 8080 и для этого нового языка, который ему понравился (а PL/1 от IBM не нравился). Через полтора года такой компилятор был создан и получил название PL/I-80.

Все это совпало с началом эры персональных компьютеров IBM, и компилятор был доработан для Intel 8086, получив название PL/I-86.

Вполне вероятно, что это программное средство получило бы самое широкое распространение в мире вместе с персональными компьютерами IBM. Но поскольку раньше произошла странная и мутная история по поводу операционной системы CP/M-86 между Килдэлом с одной стороны, и Билом Гейтсом и IBM с другой, то ОС CP/M-86, а заодно и компилятор PL/I-86, не получили рекламы и не входили в официальный состав системного ПО IBM-PC/XT. Хотя компилятор Килдэла, работавший и под CP/M-86 и под MS DOS, продавался небольшим тиражом, он не мог не уступить примитивным Бейсику и Си, поскольку не получил миллионной аудитории. А кроме этого давил авторитет Э. Дейкстры, эмоционально критиковавшего язык.

Тем не менее, в 1985 году компилятор PL/I-86 попал и в Советский Союз, благодаря институту Радио, имевшему большой задел ПО на этом языке и активно использовавшему математику комплексных чисел. Увы, данное

подмножество языка не имело комплексных чисел и поэтому компилятор в институте не получил должного применения. Однако, благодаря личным связям, компилятор попал в другую организацию – в НПО «Энергия», где, наконец, оказался весьма кстати. Он стал использоваться для решения инженерных задач, связанных с разработкой системы «Энергия-Буран». Часть программистов перешла с ЕС ЭВМ, часть с БЭСМ-6. Использование IBM-PC/XT и компилятора PL/I-86 резко повысило эффективность и скорость работы, причем легко удалось сохранить и существенную долю имеющегося задела ПО и не только на PL/1, но даже и на БЭСМ-Алгол. Хотя при этом число пользователей PL/I-86 на предприятии все же было небольшим.

По мере получения опыта и решения все более сложных задач пришло понимание, что для дальнейшего развития (и использования возрастающих возможностей компьютеров) необходимо самостоятельно совершенствовать как компилятор, так и сам язык, поскольку его авторской поддержки уже не было. Это позволило избежать главной ошибки копирования IBM 360 – отказа развивать далее самостоятельно системное ПО, используя имеющееся как начальную базу. В данном случае уже имевшийся компилятор PL/I-86, позволил, с одной стороны, не создавать свое инструментальное ПО с нуля, а с другой – стал той самой стартовой площадкой дальнейшего развития языка и воплощения этого развития в компиляторе.

Развитие заключалось как в возврате некоторых возможностей «полного» стандарта (параллельность, комплексные числа), так и в новых особенностях (оператор обмена, типы как физическая размерность). Кроме этого был убран ряд мелких раздражающих недостатков, вроде коэффициента повторения фрагмента строки, путающегося в стандарте с коэффициентом повторения при инициализации переменных.

При этом изменения вносились не просто как умозрительные улучшения, а по мере проявления этих недостатков языка и компилятора в конкретных задачах. Процесс совершенствования растянулся на многие годы, даже десятилетия, отойдя от начальной среды 16-разрядной MS DOS и придя к современной 64-разрядной Windows 10. Сам язык облегчал подобный переход, поскольку изначально был задуман «на вырост». Например, увеличение разрядности среды не потребовало введения в язык новых типов, так как изначально имеющиеся типы вроде FIXED(15) или BIT(16) просто увеличивали свой допустимый предел до FIXED(63) или до BIT(64).

Все проекты, реализованные в течение многих лет данным средством, окончились успехом и отмечены высоким качеством реализации. Поэтому стало возможным предложить данную систему программирования широкому кругу пользователей как один из давно отработанных отечественных бесплатных инструментов, пригодных для решения различных задач путем составления соответствующих программ на языке PL/1.

В основном, реализация соответствует разработанному Американским и Международным институтами стандартов подмножеству языка "G" ("American National Standard X3.74 PL/1 General Purpose Subset"), в котором устранены серьезные недостатки языка (типа использования неописанных переменных).

В этой реализации сделан ряд доработок по сравнению со стандартом языка, например, допустимы псевдографические знаки внутри исходного текста, можно определять длину файла с помощью функции LENGTH, можно отдельно транслировать BEGIN-блоки и т.п. Все эти доработки не выходят за пределы первоначальных выразительных средств языка и улучшают надежность и понятность программ. Естественно, проведена полная русификация, и можно использовать кириллицу совершенно свободно, даже для ключевых слов языка. Сделаны также доработки, необходимые для взаимодействия с Windows, в частности введено ключевое слово IMPORT для процедур Win API из динамически подключаемых библиотек.

Добавлены возможность писать произвольные коды с помощью функции UNSPEC и возможность прямого доступа к регистрам процессора. Хотя эти доработки формально делают язык машинно-зависимым, практика показывает, что они совершенно необходимы в сложных проектах, где нужна оптимизация и доступ к аппаратуре.

Все отличия от стандартов приведены в разделах 24-25 и упоминаются в примечаниях.

Это руководство является справочником, а не учебником по Windows и по языку PL/1. Поэтому предполагается, что читатель имеет некоторые знания по программированию и математике. Однако спецификация языка в руководстве дается полно, с формальным описанием всех понятий.

ОБОЗНАЧЕНИЯ

- В данном руководстве большинство латинских обозначений и ключевых слов набрано большими буквами.
- Примеры программ обычно набраны малыми буквами.
- Квадратные скобки в описаниях обозначают необязательный элемент.
- Три точки обозначают пропуск одинаковых элементов или пропуск фрагментов программ в примерах.
- Символ "кружок" выделяет структурные понятия языка или примечания.
- В конце руководства имеется алфавитный указатель английских обозначений со ссылками на номера страниц, где они встречаются.

Последняя редакция 10.09.2021. г. Королев

1. КРАТКИЕ СВЕДЕНИЯ О РЕАЛИЗАЦИИ ЯЗЫКА PL/1

- Система программирования, включающая компилятор, редактор связей, редактор библиотеки, транслятор с ассемблера, встроенный отладчик, системные библиотеки, файлы стандартных описаний, словом, все, кроме редактора исходного текста, находится в единственном файле PLINK64.EXE.

- Какая именно утилита будет запущена, определяется видом командной строки, обычно по указанию расширения файла:
  .PL1 или .PLI - исходная программа на языке PL/1;
  .A86 - исходная программа на ассемблере RASM-KT;
  .OBJ - объектный модуль, полученный в результате трансляции;
  .L86 - библиотека объектных модулей.

- Для работы требуется лишь переписать файл PLINK64.EXE в папку, путь к которой ищется по умолчанию, и затем, используя текстовый редактор какой-либо общеизвестной программы типа FAR или TOTAL COMMANDER, создавать исходные тексты и запускать PLINK64 из командной строки. Никакая настройка операционной системы не требуется, хотя и можно режимы компиляции задавать через реестр Windows.

- Исходный текст программы на PL/1 может быть создан с помощью любого текстового редактора в «кириллице MS DOS», длина строки - до 245 символов. Если строка окажется длиннее, при просмотре компилятор обрежет ее. Следует отметить, что максимальный размер файла с исходным текстом (без учета вложенности оператором %INCLUDE) не должен превышать мегабайта.

- В результате работы компилятор создаст файл с тем же именем, как и у файла с исходным текстом и расширением .OBJ, представляющий собой объектный модуль в формате OMF (Intel Technical Specification 121748-001) с доработкой для x86-64. Файл помещается в текущую папку на диске. После обработки всех файлов типа .OBJ редактором связей, создается файл с расширением .EXE (максимальный размер команд в нем до 16 Мбайт), который и является выполняемой 64-х разрядной программой в операционной системе Windows. Если программа состоит только из одного главного модуля – редактор связей запускается автоматически.

- Выполняемая программа не обязательно должна создавать «окна» Windows, использовать механизм «обратных вызовов» или вообще обращаться к Win API. По умолчанию вывод идет в стандартное окно-консоль. Таким образом, любимый всеми пример простейшей программы в виде:

```
P:PROC MAIN;
PUT (‘Hello, world!’);
END;
```

  выдаст соответствующий текст на экран.

  С другой стороны, ни что не мешает использовать всю богатую номенклатуру подпрограмм Win API, включая, например, DirectX или GDI+ и создавать программы с развитой графикой.

- С помощью встроенного редактора библиотеки LIB-KT файлы типа .OBJ могут быть объединены в библиотеки и использоваться при формировании выполняемых файлов. Свои собственные библиотеки можно с помощью оператора %INCLUDE включить в список вызываемых по умолчанию библиотек.

- При необходимости отдельные процедуры или даже целый драйвер для Windows можно написать на ассемблере – для этого имеется транслятор с ассемблера RASM-KT.

- Встроенный в каждый EXE-файл интерактивный отладчик SID-KT позволяет выполнять созданную программу по командам или подпрограммам, или по строкам исходного текста, сохраняя при этом связь между именами меток и переменных в программе и их адресами, а при режиме расширенной отладки имея также и характеристики объектов в терминах PL/1.

2. СТРУКТУРА ПРОГРАММЫ PL/1

2.1. Общее построение программы

Каждая программа на PL/1 состоит из следующих частей:

- Структурных операторов
- Операторов описания
- Выполняемых операторов

Структурные операторы определяют наличие логических частей в программе и, благодаря этому, определяют общую структуру программы. Когда программа выполняется, управление передается от одной такой логической части к другой. Логические части сами могут содержать в себе другие логические части, т.е. они могут быть вложенными. Структурные операторы также определяют иерархический порядок программы: какие логические части являются главными по отношению к другим.

Операторы описания определяют среду каждой логической части. Среда – это просто имена и атрибуты переменных, которые используются или инициируются в данной логической части. Операторы описания определяют контекст переменной, что позволяет правильно использовать ее в данной логической части программы. По сути, описания – это информация для компилятора, без нее невозможно правильно и эффективно перевести исходную программу в выполняемые команды. В языке имеется развитой принцип умолчания, позволяющий сократить и упростить описания.

Выполняемые операторы - это операторы, которые собственно и выполняют некоторые действия в программе, в то время как структурные операторы и операторы описания создают только среду и контекст для выполняемых операторов. Все выполняемые операторы можно отнести к одной из следующих категорий:

- Операторы присваивания, которые присваивают значение выражения или константы заданной переменной.
- Операторы передачи управления, которые передают управление между логическими частями программы.
- Операторы прерываний, которые позволяют программе перехватить и обработать ошибки или определенные ситуации во время выполнения программы.
- Операторы ввода/вывода, которые управляют потоком данных в/из

внешних устройств обмена.
- Операторы управления памятью, которые распределяют память.
- Препроцессорные операторы, которые выполняются во время компиляции и управляют исходным текстом программы.
- Пустые операторы, которые не выполняют действий, но служат метками или заполнителями («держателями места») в программе.

Все операторы в PL/1 состоят из необязательной метки, за которым следует ключевое слово (см. раздел 2.12.) и тело оператора, оканчивающееся точкой с запятой. Только у оператора присваивания нет ключевого слова, а у оператора описания не может быть метки. Следующие разделы детально описывают каждый тип оператора. Раздел 14. содержит полный список всех конструкций PL/1 в алфавитном порядке.

2.2. Блоки

PL/1 - блочно-структурированный язык. Это означает, что каждая логически отдельная часть программы, состоящая из одного или более операторов, составляет блок. Блок - это набор операторов, в которых используются описанные переменные. Внутри блока можно описать переменные и для некоторых переменных выделить, а затем освободить память. Можно вкладывать блоки друг в друга, но их границы не могут пересекаться.

Есть два типа блоков: BEGIN-блоки и PROCEDURE-блоки. В BEGIN-блоке последовательность операторов ограничивается ключевыми словами BEGIN и END. PROCEDURE-блок ограничен ключевыми словами PROCEDURE и END. Исходный текст PL/1-программы должен представлять собой один PROCEDURE-блок (если это главный модуль) или, возможно, один BEGIN-блок с меткой (если это не главный модуль). BEGIN-блок имеет вид:

```
[метка:] BEGIN;
оператор-1;
...
оператор-n;
END [метка];
```

где оператор-1 ... оператор-n - любые операторы PL/1, составляющие тело блока.

BEGIN-блоки могут содержать вложенные PROCEDURE-блоки и вложенные BEGIN-блоки. В данной реализации необязательная метка после END не является признаком автоматического закрытия всех открытых блоков,

как это реализовано в «полном» стандарте PL/1. Максимальный уровень вложенности блоков в данной реализации – 31.

***Замечание**: метка после END необязательна, но если она стоит, то должна соответствовать метке перед соответствующим BEGIN.*

PROCEDURE-блок имеет вид:

```
имя_процедуры: заголовок процедуры;
оператор-1;
...
оператор-n;
END [имя_процедуры];
```

где имя_процедуры - идентификатор процедуры и оператор-1 ... оператор-n - любые операторы PL/1, составляющие тело процедуры. Раздел 2.9. описывает заголовок процедуры.

Главная процедура имеет вид:

```
имя_процедуры: PROCEDURE [ (входная_строка) ] MAIN;
...
операторы или блоки
...
END [имя_процедуры];
```

***Замечание:** имя после END необязательно, но если оно стоит, то должно соответствовать имени перед соответствующим PROCEDURE.*

Главное отличие между BEGIN-блоком и PROCEDURE-блоком в том, как в них передается управление при выполнении программы. Управление передается в BEGIN-блок в той точке, где он описан, в этот момент блок становится активным. Когда управление выходит из блока, оно передается на первый выполняемый оператор, стоящий после END данного блока.

В случае же процедуры, PL/1 пропускает описание PROCEDURE-блока при выполнении программы и передает в него управление только в точке вызова (см. раздел 2.6.). Следующий пример иллюстрирует понятие блока:

Пример BEGIN- и PROCEDURE- блоков

```
A:
   PROCEDURE MAIN;

            BEGIN;


            END;

            BEGIN;
                        BEGIN;

                        END;
            END;
END A;
```

2.3. Внешние и внутренние блоки

Каждый блок характеризуется как внешний или внутренний по отношению к окружающим его другим блокам. Внутренняя процедура - это один из блоков, входящий в обрамляющий ее блок. Внешняя процедура расположена отдельно от всех других блоков. Такая процедура не входит как вложенная ни в один блок. Одна из внешних процедур является главной - с нее начнется выполнение программы. В следующем примере (а) блоки P1, P2 и P3 - внешние, но BEGIN-блок - внутренний по отношению к P3. В примере (б) P1 - внешний блок, а P2, P3 и BEGIN – внутренние блоки.

```
P1:
   PROCEDURE MAIN;


END P1;

P2:
   PROCEDURE;


END P2;

P3:
   PROCEDURE;

          BEGIN;


          END;
END P3;
```

Пример а

```
P1:
   PROCEDURE MAIN;


    P2:
       PROCEDURE;


    END P2;

    P3:
       PROCEDURE;

              BEGIN;


              END;
    END P3;
END P1;
```

Пример б

2.4. Область действия переменных

Область действия переменной определяется блоком, в котором эта переменная описана. Переменная может быть внешней или внутренней по отношению к тому блоку, где она используется.

Если Вы описали переменную в блоке, то можете ссылаться на нее в любом блоке, входящем в данный блок. Переменная может быть также локальной для некоторого блока и тогда на нее нельзя ссылаться из внешнего блока (где нет ее описания).

Следующий пример иллюстрирует понятие области действия:

```
P1:
  procedure;
   declare
    (a, b) fixed binary(7);
     a=2;    /*  a локальна в P1 */
     b=3;    /*  b локальна в P1 */
   P2:
      procedure;
       declare
       b fixed binary (7);
       b=2;   /* b локальна в P2 */
       a=a*b; /* b - это b из P2, но не из P1 */
   end P2;
     put list(a, b);
end P1;
```

PL/1 создаст новую переменную «b» в блоке P2 потому, что она описывается в этом блоке. Оператор PUT LIST расположен вне блока P2, и поэтому не может вывести переменную, описанную внутри P2. Переменная «a» внутри блока P2 является внешней по отношению к этому блоку, и блок может изменять ее значение. Таким образом, результат вывода будет 4 и 3.

В данной реализации все локальные данные внутри PROCEDURE-блока могут быть также обнулены, например, при входе, одним единственным оператором с использованием идентификатора этой процедуры:

```
test:proc(a,b);
dcl (a,b,c,d)      float;
dcl (x1,x2,x3)     fixed;
test=0;             // обнуление всех локальных переменных
…
end test;
```

При каждом входе в процедуру test ее локальные переменные c, d, x1, x2, x3 будут обнуляться.

Любая переменная, описанная как EXTERNAL (внешняя), известна во всех блоках, где она также описана как EXTERNAL и во всех вложенных блоках, за исключением тех, где она переопределена без атрибута EXTERNAL. Два описания переменной с одинаковым именем означают разные области памяти для этих переменных, за исключением случая, когда оба описания имеют атрибут EXTERNAL.

```
P1:
  procedure;
   declare
    z fixed binary external;
     ...
  P2:
     procedure;
      declare
      z fixed binary external;
      ...
   P3:
      procedure;
      begin;
       declare
       z float binary; /* без атрибута EXTERNAL */
      end;
   end P3;
  end P2;
end P1;
```

В этом примере переменная «z» в P1 и P2 - это одна и та же внешняя переменная, но переменная внутри P3 - это локальная переменная, не имеющая ничего общего с внешней переменной «z».

```
P1:
  procedure options(main);
   declare
    x float binary;
     ...
  begin;
      declare
      x fixed;
      ...
  end;
  P2:
     procedure;
      declare
      x character(10) varying;
      ...
  end P2;
end P1;
```

В этом примере область действия «x» ограничивается каждым блоком, внутри которого она описана. Хотя во всех описаниях имена одинаковы, компилятор обрабатывает их как совершенно разные переменные с разными типами и выделяет для них разные области памяти.

2.5. Процедуры

В PL/1 имеется два типа процедур: процедуры и процедуры-функции. Оба типа выполняют определенные действия в программе и логически отделяются от остальной программы. Оба типа могут выполнять некоторую последовательность действий один или несколько раз, при этом сама последовательность действий описывается в тексте программы только один раз.

При вызове процедуры (оператором CALL) указывается необязательный список данных (аргументов), с которыми она должна работать. Процедура манипулирует с этими данными, возможно, меняет их, и возвращает вызывающей программе. После окончания работы процедуры, управление передается на оператор, расположенный непосредственно после оператора вызова процедуры.

Процедура-функция также манипулирует с указанными данными и затем возвращает еще и единственное значение. Вызов процедуры-функции производится указанием ее имени и списка аргументов в выражении. Управление передается функции в точке ее вызова в выражении, а затем возвращается вместе со значением функции и выражение продолжает вычисляться.

Замечание: *в данной реализации, если процедура-функция возвращает значение типа FIXED BINARY или BIT, ее можно указывать и в операторе CALL. В этом случае возвращаемое функцией значение нигде не используется и просто пропадает.*

2.6. Оператор CALL

Общий вид оператора вызова процедуры CALL:

```
CALL имя_процедуры [(индекс-1, ... индекс-n)] [(список_аргументов)];
```

индекс-1 ... индекс-n необходимы, если имя_процедуры это - индексированная переменная типа ENTRY VARIABLE (см. раздел 3.3.2.). Список_аргументов определяет набор данных, передаваемых процедуре для обработки.

Примеры вызова процедур:

```
call print_header;
call compute(base_pay,overtime);
```

Примеры вызова функций:

```
point=3.14/sin(A);
put list(sum(X,Y));
```

Если этот оператор имеет атрибут TASK, то указанная процедура запускается в параллельном режиме, например:

```
CALL P1 TASK;
```

В данной реализации возможно также применение атрибута STOP для остановки запущенной ранее с атрибутом TASK процедуры:

```
CALL P1 STOP;
```

Этот оператор, если он указан внутри процедуры с этим же именем, эквивалентен отсутствующему в данной реализации оператору EXIT.

После запуска в параллельном режиме также начинает иметь смысл оператор WAIT (см. раздел 14).

Замечание: *в данной реализации оператор CALL для вызова процедур необязателен, т.е. вместо оператора типа CALL F (X,Y); можно писать просто F(X,Y); В тех редких случаях, когда имя процедуры совпадает с ключевым словом языка, по-прежнему нужно использовать ключевое слово CALL. Например, если Вы назвали процедуру словом "IF" (Вы с ума сошли!), то конечно нельзя написать IF(X,Y); но можно CALL IF(X,Y);*

2.7. Оператор RETURN

Оператор RETURN возвращает управление в точку, расположенную непосредственно за вызовом данной процедуры. Он также возвращает значение, если это процедура-функция.

Оператор RETURN имеет вид:

```
RETURN [(возвращаемое_значение)];
```

где возвращаемое_значение - значение функции, передаваемое в точку вызова. При необходимости PL/1 преобразует возвращаемое значение к виду, определяемому атрибутами, указанными в операторе RETURNS заголовка процедуры (см. раздел 2.9.).

RETURN оканчивает процедуру. Если RETURN встречается в главной процедуре, то управление возвращается в операционную систему. Если главная процедура имеет атрибут RETURNS, то возвращаемое значение выдается как код ответа операционной системе. Компилятор выдаст предупреждение, если в процедуре-функции формально возможен выход без возврата значения.

Примеры:

```
return;
return(X**2);
return(F (A,(B)));
```

2.8. Формальные и фактические параметры

Данные, которые передаются процедуре для обработки, называются фактическими параметрами, в то время как данные, описанные в заголовке процедуры, называются формальными параметрами. При вызове процедуры, PL/1 ставит в соответствие каждому формальному параметру свой фактический, например:

```
/* Вызов с фактическими параметрами */
CALL COMPUTE(A+(B+C),R/2,3.14);

/* Описание с формальными параметрами */
COMPUTE:PROCEDURE(X,Y,Z);
DCL (X,Y,Z) FLOAT;
...
END COMPUTE;
```

Когда Вы передаете фактический параметр по имени, формальный и фактический параметры разделяют общую память и любое изменение этого параметра внутри процедуры приводит к соответствующему изменению фактического параметра в вызывающей программе.

Когда Вы передаете фактический параметр по значению, формальный и фактический параметры не разделяют общую память. В этом случае PL/1 копирует фактический параметр в другую область памяти и любое изменение этого параметра внутри процедуры не влияет на значение фактического параметра.

Следующий пример иллюстрирует передачу параметров:

```
┌ A:
│   procedure;
│    declare
│      ACTUAL fixed binary,
│      DUMMY  fixed binary;
│
│    call X (ACTUAL);
│    call X ((DUMMY));
│
│    ┌ X:
│    │    procedure (FORMAL);
│    │     declare
│    │     FORMAL fixed binary;
│    │     FORMAL=3;
│    └ end X;
└ end A;
```

Параметр ACTUAL передается по имени, поэтому оператор присваивания внутри процедуры X меняет значение ACTUAL на 3. Параметр DUMMY передается по значению, поэтому он копируется во внутреннюю переменную процедуры X.

Параметры передаются по имени, когда атрибуты фактического и формального параметров совпадают. Параметры передаются по значению, если фактический параметр это:

- константа
- имя процедуры
- выражение, состоящее из ссылок на переменные и операторы
- переменная, заключенная в круглые скобки
- вызов функции
- переменная, имеющая тип, отличный от типа формального параметра

В последнем случае производится преобразование к типу формального параметра, с учетом заданной точности. Следующий пример иллюстрирует это:

```
┌ A:
│   procedure;
│    declare
│      X character(7),
│     (Y,Z) fixed binary;
│
│    call p(X,(Y),Z);
│
│   ┌ p:
│   │    procedure (A,B,C);
│   │     declare
│   │      A character(7),
│   │      B fixed binary,
│   │      C float binary;
│   │
│   │     A='Digital';
│   │     B=100;
│   │     C=2.5E2;
│   └ end p;
└ end A;
```

Оператор CALL вызывает процедуру с тремя параметрами X, Y и Z, соответствующим трем формальным параметрам A, B и C. Первый параметр передается по имени, так как полностью соответствует формальному параметру. Второй параметр передается по значению, так как заключен в круглые скобки. Третий параметр преобразуется к типу FLOAT и передается по значению.

В данной реализации перед каждым фактическим параметром можно поставить знак «*», указывающий компилятору, что этот параметр будет выходным. Если из-за несоответствия типов или рекурсии создается копия этого параметра (и, следовательно, он не может быть в данном случае выходным) - компилятор выдаст предупреждение. Знак «*» несовместим с заключением параметра в круглые скобки. Пример: call test(*x1, *x2, (x3));

2.9. Заголовок процедуры

В PL/1 Вы можете описать процедуру с помощью оператора PROCEDURE в любом месте программы. Однако, для лучшей читаемости программы, желательно размещать все описания вместе в начале или в конце программы.

Оператор PROCEDURE обозначает входную точку в процедуру, определяет начало блока процедуры и описывает список формальных параметров (и возвращаемое значение для функции). Процедура может состоять из одного или нескольких операторов, включая соответствующий оператор END конца процедуры. Оператор END также обозначает выходную точку из процедуры,

хотя внутри процедуры может быть один или несколько операторов RETURN.

Общий вид оператора PROCEDURE:

```
имя_процедуры: PROCEDURE [(список_параметров)]
        [OPTIONS(опция, ...)]  [RETURNS(список_атрибутов)]
        [RECURSIVE] [EXTERNAL] [MAIN] [IMPORT]
```

список_параметров - это список формальных параметров, которые необходимо описать внутри тела процедуры (в ее блоке). Параметры могут быть:

- скалярной переменной
- массивом
- верхним уровнем структуры

но не могут иметь атрибутов памяти

- STATIC
- AUTOMATIC
- BASED
- CONTROLLED
- DEFINED
- EXTERNAL

Если параметр в описании заключен в скобки – передача такого параметра становится возможна только по значению.

Конструкция **OPTIONS** определяет опции MAIN или STACK.

MAIN определяет данную процедуру как первую, с которой начнется выполнение программы. MAIN можно указывать и без OPTIONS.

STACK(n) устанавливает размер стека в заголовке EXE-файла (в поле SIZE_OF_STACK_RESERVE заголовка PE-формата) в n байт памяти. По умолчанию под стек отводится 100000H байт. Это стек, который выделяет Windows при начале работы программы.

Атрибут **RETURNS** для процедур-функций задает атрибуты возвращаемого значения. В данной реализации это слово можно опускать, т.е. вместо:

```
p:proc(a, b) returns(float);
```

можно писать просто:

p:proc(a, b) float;

Атрибут **RECURSIVE** обозначает, что процедура при выполнении может обращаться сама к себе явно или неявно.

Атрибут **EXTERNAL** определяет процедуру как внешнюю, отдельно компилируемую. Данный атрибут позволяет поместить процедуру в отдельный файл. Группу внешних процедур можно также поместить в отдельный модуль и связать общими глобальными данными. В соответствии со стандартом подмножества «G», можно компилировать каждую внешнюю процедуру отдельно, повторяя в каждой описание общих глобальных данных. Затем с помощью редактора связей все откомпилированные модули соединяются в единый загрузочный модуль.

В данной реализации после атрибута EXTERNAL может быть еще дополнтельно указано «внешнее» имя в скобках и в кавычках. Это сделано для возможности создания динамически подключаемых библиотек на PL/1 в случае, когда нужна связь с языком, где прописные и строчные буквы идентификаторов различны, например:

```
FIND_FIRST_FILE:PROC (NAME) EXTERNAL (‘FindFirstFile’) RETURNS (BIT);
```

В этом примере, если LINK-KT создает DLL, включающую данную процедуру, ее точка входа будет иметь название FindFirstFile.

Следующий пример показывает использование атрибута EXTERNAL:

```
module:
  begin;
   declare
    1 global_data static,
     2 a_field character(20) varying initial(''),
     2 b_field fixed initial(0),
     2 c_field float initial(0);
   set_a:
      procedure (c) external;
       declare c character(20) varying;
       a_field=c;
   end set_a;
   set_b:
      procedure (x) external;
       declare x fixed;
       b_field=x;
   end set_b;
   set_c:
```

```
    procedure (y) external;
     declare y float;
     c_field=y;
  end set_a;
  sum:
     procedure returns(float) external;
     return(b_field+c_field);
  end sum;
  display:
     procedure external;
     put skip list(a_field,b_field,c_field);
  end display;
end module;
```

В этом модуле пять внешних программ set_a, set_b, set_c, sum и display. Процедура, которая их использует, может быть, например, такой:

```
call_ext:
   procedure main;
    declare
     set_a entry (character(20) varying),
     set_b entry (fixed),
     set_c entry (float),
     sum returns (float),
     display entry;
   call set_a('Johnson,J');
   call set_b(25);
   call set_c(5.50);
   put skip list(sum());
   call display;
end call_ext;
```

Эти два модуля, после раздельной компиляции и объединения в один, образуют выполняемую программу. Заметим, что в данной реализации можно компилировать и отдельный BEGIN-блок, который в этом случае является классическим модулем.

Атрибут **IMPORT** (не входит в подмножество «G») необходим для процедур (типа главной оконной процедуры), которые будет вызывать сама Windows. Этот атрибут заставляет компилятор оформлять передачу параметров процедуры через стек и через регистры RCX, RDX, R8 и R9. Данный атрибут указывается также при использовании подпрограмм Win API.

2.10. Организация программы на нижнем уровне

Организация исходного текста PL/1 программы включает набор допустимых символов, правила написания идентификаторов (как ключевых слов, так и имен), операторов, констант, разделителей и комментариев.

PL/1 имеет свободный формат. Исходный текст состоит из символов, ограниченных символом возврата каретки. Можно начинать строки с произвольной позиции, однако исходный текст становится более читаемым и понятным, если соблюдаются следующие правила:

- на одной строке находится только один оператор;
- используется отступ для обозначения каждого вложенного блока или DO-группы.

Исходный текст может быть создан с помощью любого текстового редактора. В данной реализации используется «кириллица MS DOS».

Замечание: *Все файлы с текстами на PL/1 должны иметь расширение .PL1 или .PLI, иначе, при вызове PLINK64.EXE придется явно указывать специальный знак вызова компилятора (звездочка перед именем файла).*

2.11. Набор символов

Набор символов для PL/1 включает как строчные, так и прописные латинские и русские буквы, цифры, а также ряд специальных символов, краткое описание использования которых приведено ниже:

=	равенство (присваивание и логическое сравнение)
.	точка, используется для составных имен
,	запятая, разделитель
+ - * /	арифметические знаки
'	апостроф, разделитель строк
:	двоеточие (метка и разделитель для устройств ввода/вывода)
_	подчеркивание, символ в идентификаторах
?	вопросительный знак, символ в идентификаторах
#	знак номера, символ в идентификаторах
@	коммерческий знак «эт», символ в идентификаторах
%	знак процента (префикс для INCLUDE и REPLACE)
( )	круглые скобки (разделители для выражений и аргументов)
>	знак больше, чем
<	знак меньше, чем
~	тильда (логическое отрицание)
^	крышка (логическое отрицание)
!	восклицательный знак (логическое ИЛИ, сцепление строк)
|	вертикальная черта (логическое ИЛИ, сцепление строк)
\	обратная косая черта (логическое ИЛИ, сцепление строк)
&	амперсанд (логическое И)
$	доллар, символ в идентификаторах

2.12. Идентификаторы

Идентификатор - это строка от одного до тридцати одного символа, состоящая из букв, цифр и знака подчеркивания. Первый символ должен быть буквой. За букву считаются также символы "@", "$", "#", "?" и "_". Буквы всегда переводятся компилятором в верхний регистр, поэтому идентификаторы, отличающиеся только регистром букв, это один и тот же идентификатор. Русские и латинские буквы, одинаковые по написанию, считаются одними и теми же буквами.

Замечание: *использовать вопросительный знак в начале имен не рекомендуется, так как он используется в именах служебных переменных, вставляемых компилятором, и для доступа к регистрам процессора.*

Каждый идентификатор в исходном тексте на PL/1 должен быть или ключевым словом, или описанным именем. Ключевые слова - это идентификаторы, которые имеют специальный смысл в языке PL/1, когда они используются в специфичном контексте. Ключевые слова используются как

имена встроенных функций, операторов языка и атрибуты данных, ниже приведен полный список ключевых слов:

A, T *	DO	KEYFROM	RETURN
ALIGNED	E *	KEYTO	RETURNS
ALLOCATE	EDIT	LABEL	REVERT
AUTO, AUTOMATIC	ELSE	LEAVE	REWRITE
B, Б *	END	LIKE	SEQUENTIAL
B1, Б1 *	ENDFILE	LINE	SET
B1, Б1 *	ENDPAGE	LINESIZE	SIGNAL
B2, Б2 *	ENTRY	LIST	SKIP
B3, Б3 *	ENV, ENVIRONMENT	MAIN	STACK
B4, Б4 *	ERROR	NULL	STATIC
BASED	EXT, EXTERNAL	ON	STOP
BEGIN	F, Ч *	OPEN	STREAM
BIN, BINARY	FILE	OPTIONS	STRING
BIT	FIXED	OUTPUT	STRINGRANGE, STRG
BUILTIN	FIXEDOVERFLOW, FOFL	OVERFLOW, OFL	SUBSCRIPTRANGE, SUBRG
BY	FLOAT	P, Ш *	TAB
CALL	FORMAT	PAGE	TASK
CHAR, CHARACTER	FREE	PAGESIZE	THEN
CLOSE	FROM	PARAMETER	TITLE
COL, COLUMN	GET	POINTER, PTR	TO
COMPLEX, CPLX	GO	PRINT	UFL, UNDERFLOW
CONTINUE	GOTO	PROC, PROCEDURE	UNAL, UNALIGNED
CONV, CONVERSION	IF	PUT	UNDEFINEDFILE, UNDF
CTL	IMPORT	R, У *	UNSPEC
DATA	INIT, INITIAL	READ	UPDATE
DCL, DECLARE	INPUT	RECORD	VALUE
DEC, DECIMAL	INT, INTERNAL	RECURSIVE	VAR, VARYING
DEF, DEFINED	INTO	REPEAT	VARIABLE
DIM, DIMENSION	KEY	REPLACE	WHILE
DIRECT	KEYED	RESIGNAL	WRITE
			X, П *
			ZDIV, ZERODIVIDE

Замечание: ** - означает, что данное ключевое однобуквенное слово (и его аналог на кириллице) используются только внутри форматов операторов GET/ PUT EDIT. Вне формата, так могут называться обычные идентификаторы.*

Описанные имена - это идентификаторы, которые описаны в операторах DECLARE. Ключевое слово может совпадать с описанным именем. Смысл идентификатора определяется по тому, как и где он используется. Компилятор различает их по окружающему контексту. Например, INDEX является именем встроенной функции PL/1, однако в контексте описания:

```
declare index fixed binary;
```

это уже описанное имя, а не встроенная функция.

2.13. Список сокращений ключевых слов

Для удобства можно использовать следующие аббревиатуры:

AUTOMATIC	AUTO
BINARY	BIN
CHARACTER	CHAR
COLUMN	COL
COMPLEX	CPLX
DECIMAL	DEC
DECLARE	DCL
DEFINED	DEF
DIMENSION	DIM
ENVIRONMENT	ENV
EXTERNAL	EXT
FIXEDOVERFLOW	FOFL
INITIAL	INIT
INTERNAL	INT
OVERFLOW	OFL
POINTER	PTR
PROCEDURE	PROC
STRINGRANGE	STRG
SUBSCRIPTRANGE	SUBRG
UNALIGNED	UNAL
UNDEFINEDFILE	UNDF
UNDERFLOW	UFL
VARYING	VAR
ZERODIVIDE	ZDIV

2.14. Константы

Константы - это объекты, записанные в текстовом виде, которые не меняются во время выполнения программы. В PL/1 есть следующие базовые константы:

- арифметические (например, 3674.799)
- символьные строки (например, 'Привет!')
- битовые строки (например, '000100101'B)

2.15. Ограничители и разделители

Объекты программы, такие как идентификаторы, должны быть отделены друг от друга. В PL/1 используются различные символы как ограничители и разделители.

Обычно ограничители оканчивают один или несколько объектов, в то время как разделители отмечают конец очередного объекта и начало следующего. В PL/1 каждый идентификатор или арифметическая константа должны быть отделены друг от друга одним или несколькими ограничителями, или разделителями. Ограничителями могут служить пробелы или операторы, или некоторые специальные символы.

2.15.1. Пробелы

В PL/1 используются собственно пробелы и знаки табуляции (CTRL+I). Компилятор игнорирует символы возврата каретки, конца строки или их последовательности в середине строковой константы. Например, оператор присваивания:

```
string='ЕСЛИ ВЫ ХОТИТЕ ЗАПИСАТЬ ОЧЕНЬ ДЛИННУЮ СТРОКУ, ПОДОБНУЮ ЭТОЙ, PL/1
  ПОЗВОЛИТ ПЕРЕНЕСТИ ЧАСТЬ ЕЕ НА ДРУГУЮ СТРОКУ';
```

игнорирует символ конца строки, однако все пробелы после слова «PL/1» и до слова «ПОЗВОЛИТ», будут включены в символьную строку.

2.15.2. Арифметические и логические операторы

Имеется четыре типа операторов. Оператор, состоящий из двух символов, называется составным и эти два символа уже не должны разделяться пробелами или табуляцией.

Арифметические операторы

+	сложение или унарный плюс
-	вычитание или унарный минус
*	умножение
/	деление
**	возведение в степень

Операторы отношения

>	больше, чем
^> или ~>	не больше, чем
>=	больше или равно
=	равно
^= или ~=	не равно
<=	меньше или равно
<	меньше, чем
^< или ~<	не меньше, чем

Операторы для битовых строк

^ или ~	логическое НЕ
&	логическое И
| или \ или !	логическое ИЛИ

Оператор для битовых и символьных строк

|| или \\ или !! обозначают конкатенацию (сцепление строк).

2.15.3. Специальные символы

Ниже приведены специальные символы, которые также используются как ограничители и разделители. Следующие разделы описывают конкретные примеры их использования.

:	признак ENTRY- и LABEL-констант
;	конец оператора
,	конец элемента в списке
.	разделитель в составных именах
->	специальный составной разделитель, отделяющий имя указателя
=	признак присваивания и условие равенства
<=>	знак оператора обмена
()	разделители начала и конца списков имен, выражений, индексов т.д.

2.15.4. Комментарии

Комментарии позволяют вносить любой текст в программу. Компилятор их просто игнорирует, так что их можно помещать в любое место, где допустимы ограничители. Комментарий начинается парой символов /* и оканчивается обратной парой */, например:

```
get list(name); /* прочитать имя */
```

Внутри комментария не должен встречаться другой комментарий, однако это правило можно обойти, если использовать ключ компиляции "H" (см. раздел 17.1).

Наряду с обычными комментариями, можно использовать и однострочные комментарии, они начинаются с символов "//" и действуют только до конца строки. Пример: x=1; // установка счетчика

Из-за однострочного комментария нельзя ставить обычный комментарий сразу после деления, нужно вставить хотя бы пробел:

```
x=y//* ошибочный комментарий */z;
x=y/  /* правильный комментарий */ z;
```

В данной реализации все символы псевдографики (но не внутри символьных констант) рассматриваются как пробелы, поэтому их также можно использовать везде в тексте программы как своеобразные комментарии.

```
DCL  X  FIXED;
```

2.16. Препроцессорные операторы

PL/1 позволяет модифицировать исходный текст программы или включать в программу исходный текст из других файлов во время компиляции с помощью препроцессорных операторов. Препроцессорные операторы обозначаются символом "%", за которым следуют ключевые слова INCLUDE или REPLACE.

2.16.1. Оператор %INCLUDE

Оператор %INCLUDE копирует исходный текст из другого внешнего файла во время компиляции. Оператор может использоваться, например, для включения общего описания структур или форматов. Оператор имеет вид:

```
%INCLUDE 'имя_файла';
```

где имя_файла - это символьная строка, обозначающая имя копируемого файла, она должна быть записана в стандартном виде:
[путь] имя [.расширение].

Если путь не указан, файл ищется в текущей папке (откуда читался исходный текст). Когда компилятор доходит до оператора %INCLUDE, он открывает указанный файл, читает и компилирует его. Когда компилятор доходит до конца этого файла, он закрывает его и продолжает читать текущий исходный текст.

Если в качестве диска указана буква «Z:», то содержимое INCLUDE ищется, как встроенный «ресурс» файла PLINK64.EXE. Таким образом, стандартные описания не нужно хранить отдельно от PLINK64.EXE.

Пример использования оператора %INCLUDE:

```
f: procedure;
   declare a fixed binary;
   %include 'c:\lib\struc.lib';
   declare c float;
  ...
end f;
```

компилятор вставит исходный текст из файла c:\lib\struc.lib в точку, где встретился оператор %INCLUDE.

Замечание: *PL/1 не позволяет вкладывать %INCLUDE друг в друга*.

Если расширение файла .L86 (и только в этом случае), файл считается библиотекой, которую, как и PL1LIB.L86, надо будет вызвать при редактировании связей. Имя этой библиотеки включается в объектный модуль (в файл .OBJ). Такие библиотеки должны быть обязательно указаны в главной программе и их должно быть не более 15. Путь у библиотеки отбрасывается.

Для более короткой записи, вместо пары операторов:

```
%INCLUDE 'Z:SERVICE.DCL';
%INCLUDE 'SERVICE.L86';
```

можно писать один эквивалентный им оператор
%%INCLUDE 'Z:SERVICE.DCL'; Два символа "%%" показывают здесь, что нужна библиотека <имя>.L86.

Если внутри имени файла в %INCLUDE встречается символ "?", он заменяется на символ, который можно установить через системный реестр Windows или указать в командной строке после символа «%». Таким образом, появляется возможность примитивной «условной компиляции», поскольку можно включать в компилируемый текст разные файлы, не меняя текста самой программы. Полная длина имени в %INCLUDE не должна превышать 128 символов.

2.16.2. Оператор %REPLACE

Оператор %REPLACE позволяет давать имена константам в программе. Оператор имеет вид:

```
%REPLACE идентификатор BY константа;
```

Каждый раз, когда компилятор встречает в исходном тексте указанный идентификатор, он заменяет идентификатор символьным текстом, обозначаемой константой. Константа может быть арифметической константой (со знаком или без), битовой строкой или символьной строкой, другим идентификатором или даже ключевым словом.

Можно записать несколько конструкций REPLACE используя один оператор %REPLACE, при этом элементы разделяются запятыми.

Например, оператор: %replace true by '1'b; заменяет все вхождения имени true на константу '1'b и поэтому оператор do while(true); будет заменен на оператор do while('1'b);

PL/1 требует, чтобы все операторы %REPLACE стояли в самом внешнем блоке и до начала описания вложенных блоков (таким образом, в точке, где встретилось в программе данное имя, REPLACE для него уже должен быть известен).

Замечание: *для удобства чтения и отладки программы, лучше всего поместить %REPLACE сразу за заголовком процедуры.*

Используя оператор %REPLACE, можно вводить в язык русские ключевые слова:

```
%replace если     by if,
          тогда   by then,
          иначе   by else;
    ...
если счетчик>5 тогда граница=1; иначе граница=2;
    ...
```

2.16.3. Русские предопределенные ключевые слова

В данной реализации имеется таблица уже готовых «предопределенных» русских ключевых слов, которая вводится с помощью неявного оператора %REPLACE в самом внешнем блоке и позволяет писать исходный текст программы, вообще не пользуясь английскими словами и аббревиатурами. Ввод такой таблицы в программу можно отменить, используя ключ трансляции «M». Это сделано для случая, когда уже есть текст программы, где «русские» имена используются как идентификаторы переменных, а не как ключевые слова.

Использование русских ключевых слов никак не влияет на работу компилятора и результат компиляции. Исходный текст программы при этом обычно удлиняется, однако делается понятнее, «естественней» и даже похожим на текст инструкции на русском языке.

Значения всех операторов %REPLACE (и предопределенных и введенных программистом) можно посмотреть, задав ключ компиляции «S» (и, возможно «D»). Ниже приведена таблица всех русских предопределенных ключевых слов. Для удобства также логические константы ’1’B и ’0’B предопределены как слова ДА и НЕТ.

```
И                       = '&'
НЕТ                     = '0'B1
ДА                      = '1'B1
ИЛИ                     = '\'
НЕ                      = '^'

АДРЕС                   = ADDR
ДАТЬ_ПАМЯТЬ             = ALLOCATE
ЕСТЬ_В_СТЕКЕ            = ALLOCATION
ВРЕМЕННОЕ               = AUTOMATIC
ОСНОВА                  = BASED
БЛОК                    = BEGIN
ДВОИЧНОЕ                = BINARY
БИТ                     = BIT
БУЛЕВА_АЛГЕБРА          = BOOL
РОДНАЯ                  = BUILTIN
ЭТО                     = BY
С_ШАГОМ                 = BY
ВЫЗВАТЬ                 = CALL
ЦЕЛОЕ_НЕ_МЕНЬШЕ         = CEIL
ТЕКСТ                   = CHARACTER
ЗАКРЫТЬ                 = CLOSE
СТОЛБЕЦ                 = COLUMN
КОМПЛЕКСНОЕ             = COMPLEX
ОПЯТЬ                   = CONTINUE
С_ИМЕНАМИ               = DATA
ДАТА                    = DATE
ДАТА_4Г                 = DATE4Y
ОПС                     = DCL
ДЕСЯТИЧНОЕ              = DECIMAL
ОПИСАНИЕ                = DECLARE
НА_МЕСТЕ                = DEFINED
ЗАСНУТЬ                 = DELAY
РАЗМЕРНОСТЬ             = DIMENSION
ПРЯМОЙ                  = DIRECT
ДЕЛИТЬ                  = DIVIDE
ЦИКЛ                    = DO
В_ФОРМЕ                 = EDIT
ИНАЧЕ                   = ELSE
КОНЕЦ                   = END
КОНЕЦ_ФАЙЛА             = ENDFILE
КОНЕЦ_СТРАНИЦЫ          = ENDPAGE
ДЛЯ_ВЫЗОВА              = ENTRY
ДЛЯ_ОС                  = ENVIRONMENT
ОШИБКА                  = ERROR
ОШИБКУ                  = ERROR
ОБЩЕЕ                   = EXTERNAL
ОБЩАЯ                   = EXTERNAL
ФАЙЛ                    = FILE
ИЗ_ФАЙЛА                = FILE
В_ФАЙЛ                  = FILE
УЖЕ_ОТКРЫТ              = FILEOPEN
ТОЧНОЕ                  = FIXED
ЧИСЛО_НЕ_ПРЕДСТАВИМО    = FIXEDOVERFLOW
ВЕЩ                     = FLOAT
ВЕЩЕСТВЕННОЕ            = FLOAT
ЦЕЛОЕ_НЕ_БОЛЬШЕ         = FLOOR
ВВЕСТИ_ФОРМАТ           = FORMAT
ВЕРНУТЬ_ПАМЯТЬ          = FREE
ИЗ                      = FROM
ЧИТАТЬ                  = GET
ИДТИ                    = GOTO
ВЕРХ_ГРАНИЦА            = HBOUND
ЕСЛИ                    = IF
ЧУЖАЯ                   = IMPORT
ИСКАТЬ                  = INDEX
ЗАДАТЬ                  = INITIAL
ДЛЯ_ВВОДА               = INPUT
МЕСТНОЕ                 = INTERNAL
В_ПЕРЕМ                 = INTO
ИНДЕКС                  = KEY
ИНДЕКСНЫЙ               = KEYED
ИЗ_ИНДЕКСА              = KEYFROM
ДЛЯ_ИНДЕКСА             = KEYTO
МЕТКА                   = LABEL
НИЖ_ГРАНИЦА             = LBOUND
ХВАТИТ                  = LEAVE
ДЛИНА                   = LENGTH
КАК                     = LIKE
ПЕРЕВОД_СТРОКИ          = LINE
ПЕЧАТАЕТСЯ_СТРОКА       = LINENO
С_ДЛИНОЙ_СТРОКИ         = LINESIZE
В_ВИДЕ                  = LIST
ГЛАВНАЯ                 = MAIN
ОСТАТОК                 = MOD
УМНОЖИТЬ                = MULTIPLY
ПУСТО                   = NULL
ПУСТОЙ_УКАЗ             = NULL_PTR
КОГДА                   = ON
КОГДА_КОД               = ONCODE
КОГДА_ФАЙЛ              = ONFILE
КОГДА_ИНДЕКС            = ONKEY
ОТКРЫТЬ                 = OPEN
ДЛЯ_ВЫВОДА              = OUTPUT
ЧИСЛО_ВЕЛИКО            = OVERFLOW
ПЕРЕВОД_СТРАНИЦЫ        = PAGE
ПЕЧАТАЕТСЯ_СТРАНИЦА      = PAGENO
С_ДЛИНОЙ_СТРАНИЦЫ       = PAGESIZE
ПАРАМЕТР                = PARAMETER
УКАЗАТЕЛЬ               = POINTER
ПЕЧАТНЫЙ                = PRINT
ДЛЯ_ПЕЧАТИ              = PRINT
ПРОЦ                    = PROC
ПРОЦЕДУРА               = PROCEDURE
УКАЗ                    = PTR
ПИСАТЬ                  = PUT
ПЕЧАТАТЬ                = PUT
ВВОД                    = READ
НЕ_ТЕКСТОВЫЙ            = RECORD
РЕКУРСИВНАЯ             = RECURSIVE
ПОВТОРЯЯ                = REPEAT
ЗАМЕНИТЬ                = REPLACE
РЕСИГНАЛ                = RESIGNAL
ВОЗВРАТ                 = RETURN
ВОЗВРАЩАЕТ              = RETURNS
ОТМЕНИТЬ                = REVERT
ПЕРЕЗАПИСАТЬ            = REWRITE
```

```
ОКРУГЛИТЬ              = ROUND
ПООЧЕРЕДНЫЙ            = SEQUENTIAL
В_УКАЗ                 = SET
СИГНАЛ                 = SIGNAL
С_НОВОЙ                = SKIP
СТЕК                   = STACK
ПОСТОЯННОЕ             = STATIC
СТОП                   = STOP
ТЕКСТОВЫЙ              = STREAM
В_СТРОКУ               = STRING
ИЗ_СТРОКИ              = STRING
ПОДСТРОКА              = SUBSTR
СТД_ВВОД               = SYSIN
СТД_ВЫВОД              = SYSPRINT
ТАБУЛЯЦИЯ              = TAB
ПОДСЧЕТ                = TALLY
ПРОЦЕСС                = TASK
ТОГДА                  = THEN
ВРЕМЯ                  = TIME
ПО_ИМЕНИ               = TITLE
ДО                     = TO
ПЕРЕВЕСТИ              = TRANSLATE
ОЧИСТИТЬ               = TRIM
ЦЕЛАЯ_ЧАСТЬ            = TRUNC
НЕТ_ФАЙЛА              = UNDEFINEDFILE
ЧИСЛО_МАЛО             = UNDERFLOW
МАШ_КОД                = UNSPEC
ДЛЯ_ИЗМЕНЕНИЙ          = UPDATE
ЗНАЧЕНИЕ               = VALUE
РД                     = VAR
КОСВЕННОЕ              = VARIABLE
РАЗНОЙ_ДЛИНЫ           = VARYING
ИСКАТЬ_НЕСОВПАДЕНИЕ    = VERIFY
ЖДАТЬ                  = WAIT
ПОКА                   = WHILE
ВЫВОД                  = WRITE
ДЕЛЕНИЕ_НА_0           = ZERODIVIDE
```

Данная таблица отсортирована по английским названиям, для удобства поиска ниже приведена та же таблица, но отсортированная по русским названиям.

```
АДРЕС                = ADDR
БИТ                  = BIT
БЛОК                 = BEGIN
БУЛЕВА_АЛГЕБРА       = BOOL
В_ВИДЕ               = LIST
В_ПЕРЕМ              = INTO
В_СТРОКУ             = STRING
В_УКАЗ               = SET
В_ФАЙЛ               = FILE
В_ФОРМЕ              = EDIT
ВВЕСТИ_ФОРМАТ        = FORMAT
ВВОД                 = READ
ВЕРНУТЬ_ПАМЯТЬ       = FREE
ВЕРХ_ГРАНИЦА         = HBOUND
ВЕЩ                  = FLOAT
ВЕЩЕСТВЕННОЕ         = FLOAT
ВОЗВРАТ              = RETURN
ВОЗВРАЩАЕТ           = RETURNS
ВРЕМЕННОЕ            = AUTOMATIC
ВРЕМЯ                = TIME
ВЫВОД                = WRITE
ВЫЗВАТЬ              = CALL
ГЛАВНАЯ              = MAIN
ДА                   = '1'B1
ДАТА                 = DATE
ДАТА_4Г              = DATE4Y
ДАТЬ_ПАМЯТЬ          = ALLOCATE
ДВОИЧНОЕ             = BINARY
ДЕЛЕНИЕ_НА_0         = ZERODIVIDE
ДЕЛИТЬ               = DIVIDE
ДЕСЯТИЧНОЕ           = DECIMAL
ДЛИНА                = LENGTH
ДЛЯ_ВВОДА            = INPUT
ДЛЯ_ВЫВОДА           = OUTPUT
ДЛЯ_ВЫЗОВА           = ENTRY
ДЛЯ_ИЗМЕНЕНИЙ        = UPDATE
ДЛЯ_ИНДЕКСА          = KEYTO
ДЛЯ_ОС               = ENVIRONMENT
ДЛЯ_ПЕЧАТИ           = PRINT
ДО                   = TO
ЕСЛИ                 = IF
ЕСТЬ_В_СТЕКЕ         = ALLOCATION
ЖДАТЬ                = WAIT
ЗАДАТЬ               = INITIAL
ЗАКРЫТЬ              = CLOSE
ЗАМЕНИТЬ             = REPLACE
ЗАСНУТЬ              = DELAY
ЗНАЧЕНИЕ             = VALUE
И                    = '&'
ИДТИ                 = GOTO
ИЗ                   = FROM
ИЗ_ИНДЕКСА           = KEYFROM
ИЗ_СТРОКИ            = STRING
ИЗ_ФАЙЛА             = FILE
ИЛИ                  = '\'
ИНАЧЕ                = ELSE
ИНДЕКС               = KEY
ИСКАТЬ               = INDEX
ИСКАТЬ_НЕСОВПАДЕНИЕ  = VERIFY
КАК                  = LIKE
КОГДА                = ON
КОГДА_ИНДЕКС         = ONKEY
КОГДА_КОД            = ONCODE
КОГДА_ФАЙЛ           = ONFILE
КОМПЛЕКСНОЕ          = COMPLEX
КОНЕЦ                = END
КОНЕЦ_СТРАНИЦЫ       = ENDPAGE
КОНЕЦ_ФАЙЛА          = ENDFILE
КОСВЕННОЕ            = VARIABLE
МАШ_КОД              = UNSPEC
МЕСТНОЕ              = INTERNAL
МЕТКА                = LABEL
НА_МЕСТЕ             = DEFINED
НЕ                   = '^'
НЕ_ТЕКСТОВЫЙ         = RECORD
НЕТ                  = '0'B1
НЕТ_ФАЙЛА            = UNDEFINEDFILE
НИЖ_ГРАНИЦА          = LBOUND
ОБЩАЯ                = EXTERNAL
ОБЩЕЕ                = EXTERNAL
ОКРУГЛИТЬ            = ROUND
ОПИСАНИЕ             = DECLARE
ОПС                  = DCL
ОПЯТЬ                = CONTINUE
ОСНОВА               = BASED
ОСТАТОК              = MOD
ОТКРЫТЬ              = OPEN
ОТМЕНИТЬ             = REVERT
ОЧИСТИТЬ             = TRIM
ОШИБКА               = ERROR
ОШИБКУ               = ERROR
ПАРАМЕТР             = PARAMETER
ПЕРЕВЕСТИ            = TRANSLATE
ПЕРЕВОД_СТРАНИЦЫ     = PAGE
ПЕРЕВОД_СТРОКИ       = LINE
ПЕРЕЗАПИСАТЬ         = REWRITE
ПЕЧАТАЕТСЯ_СТРАНИЦА  = PAGENO
ПЕЧАТАЕТСЯ_СТРОКА    = LINENO
ПЕЧАТАТЬ             = PUT
ПЕЧАТНЫЙ             = PRINT
ПИСАТЬ               = PUT
ПО_ИМЕНИ             = TITLE
ПОВТОРЯЯ             = REPEAT
ПОДСТРОКА            = SUBSTR
ПОДСЧЕТ              = TALLY
ПОКА                 = WHILE
ПООЧЕРЕДНЫЙ          = SEQUENTIAL
ПОСТОЯННОЕ           = STATIC
ПРОЦ                 = PROC
ПРОЦЕДУРА            = PROCEDURE
ПРОЦЕСС              = TASK
ПРЯМОЙ               = DIRECT
ПУСТО                = NULL
ПУСТОЙ_УКАЗ          = NULL_PTR
```

ИНДЕКСНЫЙ	= KEYED	РАЗМЕРНОСТЬ	= DIMENSION
		РАЗНОЙ_ДЛИНЫ	= VARYING

РД	= VAR
РЕКУРСИВНАЯ	= RECURSIVE
РЕСИГНАЛ	= RESIGNAL
РОДНАЯ	= BUILTIN
С_ДЛИНОЙ_СТРАНИЦЫ	= PAGESIZE
С_ДЛИНОЙ_СТРОКИ	= LINESIZE
С_ИМЕНАМИ	= DATA
С_НОВОЙ	= SKIP
С_ШАГОМ	= BY
СИГНАЛ	= SIGNAL
СТД_ВВОД	= SYSIN
СТД_ВЫВОД	= SYSPRINT
СТЕК	= STACK
СТОЛБЕЦ	= COLUMN
СТОП	= STOP
ТАБУЛЯЦИЯ	= TAB
ТЕКСТ	= CHARACTER
ТЕКСТОВЫЙ	= STREAM
ТОГДА	= THEN
ТОЧНОЕ	= FIXED
УЖЕ_ОТКРЫТ	= FILEOPEN
УКАЗ	= PTR
УКАЗАТЕЛЬ	= POINTER
УМНОЖИТЬ	= MULTIPLY
ФАЙЛ	= FILE
ХВАТИТ	= LEAVE
ЦЕЛАЯ_ЧАСТЬ	= TRUNC
ЦЕЛОЕ_НЕ_БОЛЬШЕ	= FLOOR
ЦЕЛОЕ_НЕ_МЕНЬШЕ	= CEIL
ЦИКЛ	= DO
ЧИСЛО_ВЕЛИКО	= OVERFLOW
ЧИСЛО_МАЛО	= UNDERFLOW
ЧИСЛО_НЕ_ПРЕДСТАВИМО	= FIXEDOVERFLOW
ЧИТАТЬ	= GET
ЧУЖАЯ	= IMPORT
ЭТО	= BY

3. ТИПЫ И АТРИБУТЫ ДАННЫХ

Данные в PL/1 могут быть или константами, или переменными. Константы – это данные, которые не могут меняться при выполнении программы, в то время как переменные - могут. Имеются также переменные с атрибутом VALUE. Они не могут изменить свое значение после начальной инициации.

Каждый элемент данных ассоциируется с набором свойств, называемых атрибутами. Они определяют такие свойства, как объем требуемой памяти, операции, которые можно проводить с данными и количество размерностей, если это массив. Оператор DECLARE явно устанавливает атрибуты переменных, в других случаях, например для констант, атрибуты устана-вливаются по умолчанию.

Переменные могут быть единственным элементом данных. Такой единственный элемент данных (или переменная или константа) называется скаляром. Переменные могут быть также набором элементов данных, такие наборы называются агрегатами данных (это массивы или структуры). Раздел 5 описывает агрегаты данных.

PL/1 поддерживает шесть типов данных:

- арифметические
- строковые
- метки
- процедуры и функции (entry)
- указатели
- файлы

Следующие разделы посвящены детальному описанию каждого типа данных.

3.1. Арифметические данные

PL/1 поддерживает три типа арифметических данных:

- FIXED BINARY для представления целых чисел.

- FLOAT BINARY или FLOAT DECIMAL для представления чисел, которые могут изменяться от очень малых значений до очень больших. Это числа с так называемой "плавающей" точкой.

- FIXED DECIMAL для представления десятичных чисел, которые имеют фиксированное общее число цифр и фиксированное число цифр справа от десятичной точки.

Каждый арифметический тип данных имеет свою величину точности представления данных, которая обозначается как целая константа «p», заключенная в круглые скобки. Эта величина «p» определяет общее число десятичных или двоичных цифр, которое может содержать данный арифметический тип. Вместо конкретного значения «p» можно писать знак «*», что означает максимально возможное значение.

Для FIXED DECIMAL после числа «p», через запятую, может следовать необязательная целая константа «q», называемая масштабным коэффициентом (scale factor). Масштабный коэффициент «q» определяет число десятичных цифр справа от десятичной точки.

Вы можете явно не задавать величины «p» и «q» в операторе DECLARE, компилятор присвоит эти значения по правилам умолчания. Если масштабный коэффициент ноль – это значит, что у числа не может быть цифр после десятичной точки.

3.1.1. Тип FIXED BINARY

FIXED BINARY представляет целые данные. Переменные, описанные как FIXED BINARY(p), могут иметь p двоичных цифр. Диапазон изменения «p» - от 1 до 63, включая границы.

Компилятор переводит отрицательные данные в двоичный дополнительный код. Вследствие этого, например, диапазон изменения чисел FIXED BINARY(31) - от -2147483648 до 2147483647.

Объем памяти, выделяемой для FIXED BINARY, зависит от заданного «p». Если «p» меньше или равно 7 - выделяется один байт памяти, если «p» больше 7, но меньше или равно 15 - два байта, если больше 15, но меньше 32 - 4 байта, а иначе 8 байт.

По умолчанию «p» устанавливается равным 15 (но можно изменить через реестр Windows). Описания переменной как FIXED, BINARY или FIXED BINARY, эквивалентны описанию ее как FIXED BINARY(15).

Компилятор обрабатывает десятичные целые в исходном тексте программы как FIXED BINARY только в специфических местах, где требуется использование данных именно как типа FIXED BINARY. Во всех других случаях константы по умолчанию принимаются типа FIXED DECIMAL. В PL/1 преобразования типов данных обычно идут с усечением (см. раздел 4), например следующий оператор присваивания установит 1 в переменную A:

```
declare A fixed binary; A=1.99;
```

3.1.2. Тип FLOAT BINARY и FLOAT DECIMAL

Данные типа FLOAT BINARY используются в научных расчетах для представления очень больших или очень малых чисел. Переменные, описанные как FLOAT BINARY(p), состоят из трех частей: знака, p двоичных цифр, которые составляют мантиссу, и целого порядка, показывающего, сколько раз нужно умножить мантиссу на 10, чтобы получить требуемое число. Можно описать число как DECIMAL FLOAT(p), причем DECIMAL и FLOAT должны следовать именно в таком порядке. Число «p» в этом случае обозначает число десятичных цифр. Компилятор сразу переводит данные DECIMAL FLOAT в соответствующий BINARY FLOAT.

Например, число 3.56E3 имеет знак "+", мантиссу 3.56 и порядок 3.

Диапазон изменения «p» для чисел обычной точности - от 1 до 24, включая границы. Соответственно диапазон изменения чисел типа FLOAT BINARY – от 1.18*10**-38 до 3.40*10**38. Число занимает 4 байта.

Диапазон изменения «p» для чисел двойной точности - от 25 до 53, включая границы. Соответственно диапазон изменения таких чисел типа FLOAT BINARY – от 9.46*10**-308 до 1.8*10**308. Число занимает 8 байт.

По умолчанию величина «p» для FLOAT BINARY принимается равной 24 (но можно изменить через реестр Windows), описание одним словом FLOAT эквивалентно описанию FLOAT BINARY(24).

Константы FLOAT представляют собой запись в научной нотации, как последовательность десятичных цифр с необязательной точкой и буквой "E", за которой следует целая константа, возможно со знаком. Примеры:

```
A=2.3E2;        число 230
B=-4.67E+5;     число -467000
C=1.98E-2;      число 0.0198
```

Вы можете использовать разные типы данных в одном выражении. PL/1 автоматически преобразует их к общему типу, прежде чем вычислить выражение. Например, если p описана как FLOAT BINARY, то в операторе присваивания: p=p+3.14159; PL/1 преобразует FIXED DECIMAL константу 3.14159 в тип FLOAT BINARY прежде, чем выполнить сложение (см. раздел 4).

3.1.3. Тип FIXED DECIMAL

Данные типа FIXED DECIMAL используются там, где нужны точные вычисления, например в коммерческих расчетах, где требуется учитывать доллары и центы. FIXED DECIMAL с нулевым масштабным коэффициентом используется для представления целых чисел.

Переменная, описанная как FIXED DECIMAL [(p [, q])], является десятичным числом со знаком и общим числом десятичных цифр - p, при этом q цифр находится справа от десятичной точки. Максимально возможное значение p - 15, а q должно быть положительным и не больше p. Диапазон изменения чисел типа FIXED DECIMAL от минус-10**(p-q) до плюс 10**(p-q), где 1 <= p <= 15 и 0 <= q <= p.

Все десятичные константы в программе, имеющие или не имеющие десятичной точки, по умолчанию принимаются за FIXED DECIMAL. Исключение составляют только константы в специфических местах программы, там, где требуется FIXED BINARY.

Значения p и q по умолчанию соответственно 15 (можно изменить через реестр) и 0. Для констант значения p и q определяются неявно, по значению самой константы, например,

3.25 по умолчанию принимается как FIXED DECIMAL(3,2)

302 по умолчанию принимается как FIXED DECIMAL(3,0)

Внутреннее представление FIXED DECIMAL - дополнительный упакованный BCD-код. Число байт памяти, занимаемое FIXED DECIMAL, зависит от величины p и равно (p+2)/2 байт, т.е. минимум один и максимум восемь байт.

PL/1 отбрасывает все цифры, находящиеся за величиной масштабного коэффициента, при присваивании переменной типа FIXED DECIMAL. Также, PL/1 сигнализирует прерыванием FIXEDOVERFLOW, если число значащих цифр слева от десятичной точки больше, чем заданное у данной переменной.

3.1.4. Комплексные числа

Данная реализация комплексных чисел имеет два ограничения:

- во-первых, комплексные числа (точнее комплексные переменные) обязательно должны иметь атрибут COMPLEX FLOAT (но не FIXED);

- во-вторых, никаких преобразований (в том числе автоматических) между комплексными и обычными переменными не производится.

Данные ограничения сильно упростили реализацию, однако потребовали особой формы записи констант. В «полном» стандарте языка подразумевались автоматические преобразования при записи констант. Например, в операторе X1=1+2I; действительная константа 1 преобразовывалась в комплексную добавлением нулевой мнимой части, а

«чисто мнимая» константа 2I преобразовывалась в комплексную добавлением нулевой действительной части. Полученные комплексные числа складывались, т.е. комплексные константы заменялись комплексными выражениями. В данной реализации комплексные константы пишутся в кавычках, например:

```
DCL X1 COMLEX FLOAT STATIC INIT ('1+2I'); или
X1='1.5E0-2.13I';
```

При этом никаких преобразований и выражений не генерируется, однако такие константы в кавычках нельзя применять в сложных выражениях и как аргументы встроенных функций, а можно использовать только в простых операторах присваивания.

Замечание: *чисто мнимые константы можно записывать и без кавычек, например, X=-5I;*

Для комплексных переменных реализованы четыре арифметических действия, а также функции ABS, SQRT, SIN, COS, TAN, SINH, COSH, TANH, EXP, LOG, ASIN, ACOS, ATAN, изменение знака, сравнение на равенство, преобразование в текстовую строку и из текстовой строки и преобразования между FLOAT(53) и FLOAT(24). Функция ABS возвращает действительное число (модуль комплексного числа).

Замечание: *как и для обычных переменных, сравнение на равенство может не выполняться из-за округлений представления FLOAT, поэтому X1+X2*X3 может быть строго не равно X3*X2+X1.*

Доступ к действительной и мнимой частям комплексной переменной удобно осуществлять с помощью «наложенной» на эту переменную структуры (или массива размерностью 2), например,

```
DCL (X1,X2)       FLOAT CPLX,

      1 XX1       DEF(X1),
        2 (R1,I1) FLOAT,

      1 XX2       DEF(X2),
        2 (R2,I2) FLOAT;

X2=X1; I2=-I1; // ПОЛУЧЕНИЕ СОПРЯЖЕННОГО ЧИСЛА
PUT SKIP LIST (R1,R2,I1,I2); // ДОСТУП К  ЧАСТЯМ ЧИСЛА
```

Поэтому в данной реализации встроенные функции COMPLEX, REAL, IMAG, CONJG отсутствуют.

3.2. Строковые данные

PL/1 поддерживает два типа строковых данных:

- символьные строки
- битовые строки

Символьные строки представляют последовательность символов, включая и пустую последовательность, т.е. нулевую строку. Битовые строки - это последовательность бит. Длина строки определяется количеством символов или бит.

Следующие разделы посвящены описанию каждого типа строковых данных.

3.2.1. Данные типа символьная строка

Переменная, описанная как CHARACTER(n), является строкой символов длиной n, где n - целое от 1 до 254. Например, оператор:

```
declare A character(10);
```

определяет переменную A как символьную строку длиной 10 символов. Если символьная строка, посылаемая в A, короче, чем 10, PL/1 дополнит ее справа пробелами. Если строка, посылаемая в A, длиннее 10 символов, PL/1 обрежет ее справа. Если такое обрезание обнаружено компилятором, он выдает предупреждение "СТРАННО", например:

```
declare s char(5);
s='123456';      // «6» отбрасывается с предупреждением
s=s!!'1';        // «1» отбрасывается с предупреждением
```

Символьная строка-константа представляет собой последовательность символов, заключенных в апострофы. Если апостроф сам является частью строки – ставится два апострофа подряд. Таким образом, строку What's Happening? надо писать, как 'What''s Happening?'. Пустая или нулевая строка имеет длину ноль и записывается как два апострофа подряд.

При записи символьной константы можно указать коэффициент повторения в круглых скобках, но не перед константой как в стандарте PL/1, а после: put skip list('='(80)); // печать 80 символов "="

В конце символьной строки также может стоять квалификатор W, обозначающий «кириллицу Windows», например:

```
DCL S1 CHAR(*) VAR STATIC INIT('Привет!');
DCL S2 CHAR(*) VAR STATIC INIT('Привет!'W);
```

Первый символ строки S1 имеет код 08FH.
Первый символ строки S2 имеет код 0CFH.

Символьные строки могут также иметь атрибут VARYING, показывающий, что переменная может иметь переменную длину строки от 0 до n. Например, оператор: declare A character(10) varying; определяет переменную A, как символьную строку длиной от 0 до 10 символов.

Для символьных строк - формальных параметров подпрограмм вместо длины строки можно указывать символ "*" - т.е. длина любая от 0 до 254, например:

```
test:procedure(s);
declare s char(*) var;
...
```

PL/1 позволяет писать управляющие символы в символьных строках. Символ "^" показывает, что нужно погасить три старших разряда в следующем символе, превращая его в управляющий символ. Так, строка ^M или ^m будет переведена в символ возврата каретки, а строка ^I - в символ табуляции. Если сам символ "^" является частью строки, он записывается, как два подряд ("^^").

Замечание: *управляющие символы не являются частью стандарта языка и могут отсутствовать в других реализациях.*

3.2.2. Данные типа битовая строка

Битовые строки представляют логические данные. Битовая строка, содержащая только нулевые биты, принимается за «ложь», строка, содержащая хотя бы один единичный бит - принимается за «истину».

Переменная, описанная как BIT(n), является битовой строкой, содержащей n бит, где n - любое целое от 1 до 64 или «*». Для переменной выделяется память от 1 до 8 байт. Например, оператор: declare A bit(3); описывает переменную A, как битовую строку длиной 3. Если битовая строка, посылаемая в данную переменную, короче 3 бит - PL/1 добавит нулевые биты справа, если строка, посылаемая в A, длиннее 3 бит, PL/1 отбросит биты справа.

Вы можете писать битовые строки-константы в четырех различных форматах. Каждый формат определяется базой - числом двоичных разрядов, требуемых для записи одной цифры константы. Битовые строки-константы представляют последовательности цифр и букв, заключенные в апострофы, и стоящую за последним апострофом букву «B», после которой следует необязательный указатель базы. По умолчанию база равна 2 и обозначается как B или B1. Ниже приведены все форматы:

Формат	База	Цифры и буквы в записи строк
B	2	0, 1
B1	2	0, 1
B2	4	0, 1, 2, 3
B3	8	0, 1, 2, 3, 4, 5 ,6 ,7
B4	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Замечание: *цифры и/или буквы в битовой строке должны соответствовать своему указателю базы*.

Примеры представления битовых строк:

'101'B1	эквивалентно '101'B
'101'B2	эквивалентно '10001'B
'101'B3	эквивалентно '1000001'B
'101'B4	эквивалентно '100000001'B
'9A'B4	эквивалентно '10011010'B
'77'B3	эквивалентно '111111'B

3.3. Управляющие типы данных

Управляющие типы данных определяют место передачи управления во время выполнения программы. PL/1 поддерживает два типа управляющих данных:

- метки (LABEL)
- процедуры (ENTRY)

3.3.1. Данные типа LABEL

Данные типа LABEL включают метки-переменные и метки-константы.

- Метка-константа - это идентификатор метки, который стоит перед выполняемым оператором.
- Метка-переменная - это переменная, описанная в операторе DECLARE с атрибутом LABEL.

Если есть описание: declare name label; PL/1 присвоит переменной name атрибут LABEL и атрибут VARIABLE по умолчанию.

Когда программа выполняется, метка-переменная может принимать значения различных меток-констант.

Вы не можете явно задать метку-константу в описании DECLARE. Однако если какое-либо имя используется в операторах как метка-константа, это

является признаком неявного описания метки-константы. PL/1 не позволяет писать метки перед следующими операторами:

- оператором DECLARE
- ON-оператором
- операторов, начинающихся после ключевых слов THEN и ELSE (см. раздел 8.3.)

В данной реализации разрешается несколько меток перед каждым оператором, в этом случае «значение» всех меток одинаково, например:

```
m1:  m2: m3: x=1; /* допустимо */
```

Присвоение метке-переменной значения метки-константы или другой метки-переменной производится по обычным правилам оператора присваивания.

К данным типа LABEL могут применяться только еще два оператора сравнения: «равно» (=) или «не равно» (^=).

Область действия меток-констант и меток-переменных такая же, как и у данных других типов. Метка известна в блоке, если она описана в DECLARE в этом блоке или (если это метка-константа) стоит в этом блоке. Новое описание метки во внутреннем блоке, делает ее локальной в этом блоке.

Вы можете писать метки-константы с индексом, используя для индекса целую константу, возможно со знаком. Появление метки с индексом в блоке означает неявное описание массива меток-констант в этом блоке, причем такое неявное описание размера массива получится только для тех крайних значений индексов, которые встретились в этом блоке. Вы можете также явно описать массив меток-переменных в операторе DECLARE.

Следующий пример иллюстрирует применение массива меток-констант:

```
case_num(1):
...
case_num(2):
...
case_num(3):
```

PL/1 обрабатывает case_num, как если бы был описан массив меток и поэтому можно ссылаться на любой элемент массива в операторе GOTO (см. раздел 8.5.):

```
goto case_num(i);
```

где i - переменная с целым значением, по которому и будет выбран адрес передачи управления.

Замечание: *PL/1 не обрабатывает метки оператора FORMAT (см. раздел 11.3.4.) или оператора PROCEDURE как обычные метки, а скорее, как данные типа FORMAT или ENTRY соответ*ственно. *На эти метки нельзя передать управление оператором GOTO.*

Следующий пример иллюстрирует использование меток-переменных:

```
declare do_it label;
...
if answer='yes' then do_it=geometric_mean;
                else do_it=arithmetic_mean;
...
goto do_it;
```

В этом примере оператор GOTO передает управление или на метку geometric_mean или на метку arithmetic_mean, в зависимости от текущего значения метки-переменной do_it.

3.3.2. Данные типа ENTRY

В PL/1 все данные типа ENTRY могут быть или ENTRY-переменными или ENTRY-константами.

ENTRY-константы определяют или точку входа во внутреннюю процедуру или точку входа во внешнюю, отдельно компилируемую процедуру.

ENTRY-переменные - это данные, которые могут принимать значения ENTRY-констант во время выполнения программы или при инициализации.

Программа должна использовать описание ENTRY для определения характеристик всех параметров и возвращаемых значений внешних, отдельно компилируемых, процедур.

Замечание: *необходимо, чтобы описание ENTRY соответствовало входным и выходным параметрам внешней процедуры, иначе редактор связей не сможет правильно объединить модули в программу. Проверку межмодульных связей можно провести с помощью утилиты LINT-KT.*

ENTRY-переменные, которым будет присваиваться значения констант, также должны быть описаны в DECLARE. При необходимости, они могут иметь индексы, в то время как ENTRY-константы индексов иметь не могут. Как и в случае данных типа LABEL, к переменным ENTRY могут применяться только операторы присваивания и операторы сравнения: "равно" и "не равно".

Описание ENTRY-константы, как точки входа внешней процедуры имеет вид:

```
DECLARE имя [EXTERNAL] [IMPORT]
        [ENTRY [(список параметров)]]
        [RETURNS (атрибуты возвращаемого значения)]
```

Описание ENTRY-переменной имеет вид:

```
DECLARE имя [(пара индексов-1, ... пара индексов-n)]
        [ENTRY [(список параметров)]] VARIABLE [IMPORT]
        [RETURNS (атрибуты возвращаемого значения)]
```

Параметры описания может идти в любом порядке, но обязательно должно быть или ENTRY или RETURNS.

В описании имя - определяет имя ENTRY-константы или ENTRY-переменной, пара индексов 1...n - определяет необязательную размерность массива ENTRY-переменной, список параметров задает список описаний формальных параметров процедуры, а атрибуты возвращаемого значения задают атрибуты значения процедуры-функции.

Атрибут **EXTERNAL** определяет процедуру как внешнюю и позволяет использовать ее в других модулях.

Атрибут **VARIABLE** определяет данное имя как ENTRY-переменную, которой будет присваиваться значение ENTRY-констант при выполнении программы.

Атрибут **RETURNS** определяет процедуру как процедуру-функцию.

Атрибут **IMPORT** обозначает внешнюю процедуру Windows, которая будет динамически загружена из библиотек при выполнении программы. Или обозначает данную процедуру как «обратный вызов из Windows».Этот атрибут имеется только в данной реализации.

Если у процедуры нет параметров, список параметров опускается, в этом случае, можно также опустить слово ENTRY, если присутствует слово RETURNS. Вместо слова ENTRY можно использовать также слово PROC.

Если заданы индексы, обязательно должен быть атрибут VARIABLE. Если атрибут EXTERNAL не указан, он ставится по умолчанию.

Если очередной параметр - это массив, он должен начинаться с описания пар индексов. Если параметр – структура, он должен начаться с номера уровня (см. раздел 5.6).

Примеры описания ENTRY:

```
declare X entry;
declare Y entry variable;
declare P (0:10) entry(fixed,float) variable;
declare Q entry(1, 2 fixed, 2 float, (5:10) decimal);
declare R returns(character(10));
```

Следующий пример иллюстрирует использование данных ENTRY:

```
declare
(X,Y)       float binary,
A           entry variable,
F (3)       entry(float) returns(float) variable,
ZZ          entry(float) returns(float);

P1:procedure;
 X=5;
end P1;
P2:procedure;
X=25;
end P2;

Y=9;
if Y=5 then A=P1; else A=P2;
call A;
F (2)=ZZ;
Y=F (2)(X);
```

3.4. Данные типа POINTER

Данные типа указатель (POINTER) определяют адрес участка памяти. Значением данных типа POINTER является адрес переменной в программе. Описание данных имеет вид: DECLARE <имя> POINTER;

PL/1 не преобразует данные этого типа в другой тип и наоборот. Таким образом, переменной типа указатель можно присвоить только значение другой переменной типа указатель или значения встроенных функций NULL и ADDR. Данные типа указатель не могут быть выведены в файл с атрибутом PRINT. Как и в случае данных типа LABEL и ENTRY, единственные операторы, которые можно применять к указателям - это операторы сравнения "равно" и "не равно".

Вы можете использовать данные типа POINTER для динамического управления памятью.

3.5. Данные типа FILE

Данные типа FILE могут быть FILE-константами или FILE-переменными, они обозначают внешние наборы данных.

Описание файлов-констант имеет вид:

```
DECLARE file_id FILE;
```

Описание файлов-переменных имеет вид:

```
DECLARE file_id [(размерность массива)] FILE VARIABLE;
```

где file_id - идентификатор внешнего файла в программе. Если в описании больше нет атрибутов, автоматически устанавливается атрибут EXTERNAL, и эта переменная становится доступной во всех модулях.

Если Вы не открыли файл в программе явно, с помощью оператора OPEN, включающим опцию TITLE, PL/1 считает, что файл имеет имя file_id.dat и находится на текущем диске в текущей папке.

Раздел 10. детально описывает работу с файлами.

3.6. Оператор DECLARE

В PL/1 Вы должны с помощью оператора DECLARE описать все имена в программе, не являющиеся именами встроенных функций и псевдопеременных (см. раздел 6.9.). FILE-константы и FILE-переменные также должны быть явно описаны в DECLARE. Имена внутренних процедур и меток-констант описываются неявно, своим появлением в программе.

Оператор DECLARE присваивает каждой переменной список атрибутов, допустимых для этого типа данных. Простейший вид DECLARE для скалярной переменной:

```
DECLARE имя [список атрибутов];
```

где имя - идентификатор переменной и список атрибутов - одна или несколько характеристик типа переменной. Атрибуты могут следовать в любом порядке и должны разделяться хотя бы одним пробелом или табуляцией.

Примеры оператора DECLARE:

```
declare x                fixed binary;
declare pi               float binary(24);
declare overtime_pay fixed decimal(5,2) static initial(000.00);
declare eof              bit(1) static initial ('1'b);
declare list_head        pointer static initial(x);
```

3.7. Составной оператор DECLARE

Для простоты и уменьшения длины текста программы, PL/1 позволяет задавать несколько описаний в одном операторе DECLARE. Обычно можно ряд описаний:

```
DECLARE описание-1;
DECLARE описание-2;
...
DECLARE описание-n;
```

записать в виде:

```
DECLARE описание-1, описание-2, ... описание_n;
```

т.е. каждое следующее описание отделяется от предыдущего запятой и, возможно, пробелами и табуляцией. Весь оператор DECLARE оканчивается точкой с запятой.

Если некоторые переменные в описании имеют одинаковые атрибуты, Вы можете вынести эти атрибуты за скобки вправо. Так, последовательность описаний:

имя-1 атрибут-A, имя-2 атрибут-A, ... имя-n атрибут-A,

можно записать как: (имя-1, имя-2, ... имя-n) атрибут-A,

например: declare (first_name, last_name) character(20) varying;

Такое сокращение может быть многоуровневым, например оператор:

```
declare ((A,B) fixed binary, C float binary) static external;
```

эквивалентен описаниям:

```
declare A fixed binary static external;
declare B fixed binary static external;
declare C float binary static external;
```

3.8. Атрибуты DECLARE, принимаемые по умолчанию

Список атрибутов не должен содержать конфликтующие атрибуты, например, такие как два типа данных или два класса памяти. Если Вы не указываете все атрибуты в операторе DECLARE, компилятор допишет отсутствующие в соответствии со следующими правилами:

- Если у идентификатора вообще нет атрибутов, ему приписывается FIXED BINARY(15) и выдается предупреждение.
- Если атрибут DECIMAL или BINARY указан без спецификации FIXED или FLOAT, приписывается FIXED.
- Если для FIXED BINARY не указано число p, принимается FIXED BINARY(15).
- Если для FLOAT BINARY не указано число p, принимается FLOAT BINARY(24).
- Если для FIXED DECIMAL не указано число p и q, принимается FIXED DECIMAL(7,0).
- Если для BIT или CHARACTER не указана длина строки, принимается BIT(1) или CHARACTER(1)

Замечание: *при описании переменных как DECIMAL FLOAT FIXED значения длин по умолчанию 7, 24, 15 можно изменить через реестр Windows.*

4. ПРЕОБРАЗОВАНИЯ ДАННЫХ

Преобразованием данных называется процесс изменения представления заданных значений из одного типа в другой. В PL/1 в преобразованиях участвуют: источник, приемник и результат. Источник - это данные, которые преобразуются, приемник - это тип, к которому их надо привести и результат - это действительный тип преобразованных данных.

PL/1 проводит следующие преобразования типов:

- арифметический в арифметический (тип и точность представления)
- арифметический в строковый
- строковый в арифметический
- в тип, определяемый форматом EDIT для ввода/вывода (см. раздел 11.3)

PL/1 не преобразует типы ENTRY, LABEL, FILE и POINTER, а также комплексные числа в действительные.

Часть преимуществ и мощь языка PL/1 как раз и заключается в том, что Вы можете свободно манипулировать данными разных типов. Вместе с тем, такая свобода требует аккуратности и понимания, как язык преобразует данные из одного типа представления данных в другой явно или неявно.

Ниже приведен список мест в программах, где по умолчанию PL/1 преобразует типы данных:

- В операторах присваивания PL/1 преобразует тип выражения к типу переменной, которой это выражение присваивается.
- В операторе RETURN возвращаемое значение преобразуется к типу, определенному атрибутом RETURNS в заголовке или описании процедуры.
- В любом арифметическом выражении, если операнды имеют разные типы, PL/1 преобразует их к общему типу прежде, чем выполнить операцию.

Например, если A FLOAT BINARY и B FIXED BINARY, то в любом из операторов:

```
A+B
A-B
A*B
A/B
A**B
```

общий тип будет FLOAT BINARY и PL/1 преобразует B к этому типу в каждом случае.

В процессе ввода/вывода PL/1 преобразует в символьную строку или из символьной строки для операторов PUT и GET соответственно. Например, если I FIXED BINARY, то в операторе PUT LIST(I); он будет преобразован в CHARACTER, а в операторе GET LIST(I); внешнее представлении I в виде символьной строки будет преобразовано в FIXED BINARY.

PL/1 преобразует специфические операторы в целые значения. Например, в операторе цикла

```
DO имя=start TO finish BY inc;
...
END;
```

выражения start, finish и inc будут приведены к целому виду (FIXED BINARY или FIXED DECIMAL) перед началом выполнения цикла.

В PL/1 есть встроенные функции, которые проводят специфические преобразования.

4.1. Арифметические преобразования

Довольно объемные правила арифметических преобразований, приведенные ниже, имеют общую цель: как можно точнее удержать значения чисел, представленных в разной форме.

PL/1 проводит арифметические преобразования в нескольких контекстах. Первый контекст - это оператор присваивания, использующий арифметические выражения или арифметические переменные. PL/1 преобразует точность представления и масштабный коэффициент выражения к типу переменной-приемника.

Другой контекст - это когда арифметическая функция возвращает арифметическое выражение в операторе RETURN. PL/1 преобразует это выражение к типу данных приемника с учетом точности представления и масштабного коэффициента, которые указаны в атрибуте RETURNS описания функции.

Когда любой арифметический инфиксный (имеющий два операнда) оператор, но не оператор возведения в степень, имеет операнды различных типов, PL/1 производит преобразования в три этапа:

Этап 1: PL/1 определяет общий тип по двум операндам.

Случай A. Если один операнд FIXED BINARY, а другой FLOAT BINARY, общий тип - FLOAT BINARY.
Случай B. Если один операнд FIXED BINARY, а другой FIXED DECIMAL, общий тип - FIXED BINARY.
Случай C. Если один операнд FLOAT BINARY, а другой FIXED DECIMAL, общий тип - FLOAT BINARY.

Эти правила можно представить в виде таблицы:

Первый операнд	Второй операнд FIXED BINARY	Второй операнд FLOAT BINARY	Второй операнд FIXED DECIMAL
FIXED INARY	FIXED BINARY	FLOAT BINARY	FIXED BINARY
FLOAT BINARY	FLOAT BINARY	FLOAT BINARY	FLOAT BINARY
FIXED DECIMAL	FIXED BINARY	FLOAT BINARY	FIXED DECIMAL

Этап 2: PL/1 преобразует каждый операнд к общему типу.

Случай А. Если общий тип FLOAT BINARY, то:

- PL/1 преобразует FIXED BINARY(p) во FLOAT BINARY(p).
- PL/1 преобразует FIXED DECIMAL(p,q) во FLOAT BINARY(p'),

где p'=MIN(CEIL(p/3.322),24), а MIN и CEIL - это преобразования в смысле встроенных функций PL/1 (см. раздел 13).

Случай В. Если общий тип FIXED BINARY, то:

PL/1 преобразует FIXED DECIMAL(p,0) в FIXED BINARY(p'),
где p'=MIN(CEIL(p/3.322),31).

Замечание: *PL/1 не может преобразовать FIXED DECIMAL(p,q) в FIXED BINARY, если q не равно 0.*

Этап 3

После преобразования к общему типу, PL/1 корректирует точность представления (и масштабный коэффициент) результата. Тип результата определяется общим типом.

Случай А. Если общий тип FIXED BINARY, то результат - FIXED BINARY. Если p1 - точность 1 операнда и p2 - точность 2 операнда, PL/1 корректирует точность результата в зависимости от операции.

Для сложения и вычитания: p'= MIN(15,MAX(p1,p2)+1);

Для умножения: p'= MIN(15,p1+p2+1);

Для деления нужно использовать встроенную функцию DIVIDE, с масштабным

коэффициентом 0 для того, чтобы получить результат FIXED BINARY потому, что PL/1 подмножества «G» не поддерживает данные FIXED BINARY с ненулевым масштабным коэффициентом.

Случай B. Если общий тип FLOAT BINARY, то результат - FLOAT BINARY. Точность результата берется как MAX(p1,p2), где p1 и p2 - точности первого и второго операнда.

Случай C. Если общий тип FIXED DECIMAL, то результат - FIXED DECIMAL. Если точность и масштабный коэффициент первого операнда p1, q1, а второго - p2, q2, то PL/1 корректирует точность результата в зависимости от операции.

Для сложения и вычитания: p'=MIN(15, MAX(p1-q1,p2-q2)+MAX(q1,q2)+1)
и q'=MAX(q1,q2)

Для умножения: p'=MIN(15,p1+p2+1)
и q'=q1+q2;

Для деления: p'=15; q'=15-(p1+q1-q2)

Замечание: *будьте осторожны с делением данных типа FIXED DECIMAL. Точность представления и масштабный коэффициент операндов должны быть такими, чтобы у результата не получился отрицательный масштабный коэффициент. Можно использовать встроенную функцию DIVIDE, чтобы контролировать точность частного*.

Если используется оператор возведения в степень типа X**Y, то возможны два случая.

Случай A. Y десятичная целая константа. Если X FIXED BINARY с точностью p и ((p+1)*Y-1) <= 15, то результат FIXED BINARY с точностью p'=(p+1)*Y-1;
Если X FIXED DECIMAL с точностью и масштабным коэффициентом p, q и ((p+1)*Y-1) <= 15, то результат FIXED DECIMAL с точностью:
p'=(p+1)*Y-1; q'=q*Y

Случай B. Если любой из операндов FLOAT BINARY, то PL/1 преобразует другой операнд во FLOAT BINARY, результат также FLOAT BINARY с точностью p'=MAX(p1,p2), где p1 и p2 - точности операндов.

В любом арифметическом операторе, требующем преобразования, PL/1 усекает результат, если описанная точность приемника меньше получившейся точности результата. Усечение производится справа для FLOAT BINARY и для

цифр после точки для FIXED DECIMAL. При вычислениях FIXED BINARY, если значение превысило по абсолютной величине 2**63, результат не определен.

Замечание: *Если при вычислениях с обычной точностью получается величина меньше 1.18*10**-38 или больше, чем 3.40*10**38 по абсолютной величине или при вычислениях с двойной точностью получается величина меньше 9.46*10**-308 или больше, чем 1.80*10**308 по абсолютной величине, программа сигнализирует об этом прерываниями UNDERFLOW(2) и OVERFLOW(2) соответственно.*

4.2. Функции арифметических преобразований

PL/1 предоставляет ряд встроенных функций для задания явного преобразования из одного арифметического типа в другой. Это следующие функции:

- BINARY
- DECIMAL
- DIVIDE
- MULTIPLY
- FIXED
- FLOAT

Следующие разделы посвящены описанию этих функций.

4.2.1. Встроенная функция BINARY

Функция имеет вид: BINARY(X [,p]) или BIN(X [,p]), где X – арифметическое или выражение или переменная, которая должна быть преобразована к типу BINARY с точностью представления p.

Если преобразуется арифметическая переменная, то если X FIXED BINARY или FIXED DECIMAL - результат FIXED BINARY. Если X FLOAT BINARY, то и результат FLOAT BINARY.

Если p не задана, то результат будет следующим:

Имеется	Переводится в
X FLOAT BINARY (p)	FLOAT BINARY (p)
X FIXED BINARY (p)	FIXED BINARY (p)
X FIXED DECIMAL(p,q)	FIXED BINARY (MIN (CEIL ((p-q)*3.322)+1,31)

4.2.2. Встроенная функция DECIMAL

Функция имеет вид: DECIMAL(X [,p [,q]]) или DEC(X [,p [,q]]), где X - арифметическое выражение или переменная, которая должна быть преобразована к типу DECIMAL с точностью представления p и масштабным коэффициентом q. Ненулевой q допустим только, если X FIXED DECIMAL.

Если p и q не заданы, результат следующий:

Имеется	Переводится в
X FIXED BINARY (p)	FIXED DECIMAL (CEIL (p/3.322)+1,0)
X FLOAT BINARY (p)	FIXED DECIMAL (MIN (CEIL ((p/3.322), 15),0)
X FIXED DECIMAL (p,q)	FIXED DECIMAL (p,q)

4.2.3. Встроенные функции DIVIDE и MULTIPLY

Встроенная функция DIVIDE задает точность и масштабный коэффициент кроме собственно деления операндов. Функция имеет вид: DIVIDE(X,Y,p [,q]), где X и Y – арифметические выражения и X нужно разделить на Y. p - выражение типа FIXED BINARY, определяющее точность результата, q - выражение типа FIXED BINARY, определяющее масштабный коэффициент результата. Если q не задано, оно принимается равной нулю. Ненулевой масштабный коэффициент допустим только, если X и Y FIXED DECIMAL.

PL/1 требует использования этой функции для деления чисел типа FIXED BINARY. В «полном» стандарте PL/1 может быть деление и с ненулевым масштабным коэффициентом, однако PL/1 подмножества «G» не поддерживает такие FIXED BINARY.

Имеется также функция MULTIPLY, которая аналогична DIVIDE, но предназначена для управления точностью при умножении. Как и DIVIDE никаких дополнительных команд (кроме самого умножения) в выполняемом коде эта функция не создает.

Функция имеет вид: MULTIPLY(X,Y,p [,q])

4.2.4. Встроенная функция FIXED

Функция имеет вид: FIXED(X [,p [,q]]), где X - арифметическое выражение или переменная, которая должна быть преобразована к типу FIXED с точностью представления p и масштабным коэффициентом q. Ненулевой q допустим, только если X FIXED DECIMAL. Если же X FIXED DECIMAL, то результат FIXED DECIMAL, в противном случае - результат FIXED BINARY и q в этом случае должно быть равно нулю. Если p или q не заданы, результат следующий:

Имеется	Переводится в
X FIXED BINARY (p)	FIXED BINARY (p)
X FLOAT BINARY (p)	FIXED BINARY (MIN (31,p))
X FIXED DECIMAL (p,q)	FIXED DECIMAL (p,q)

4.2.5. Встроенная функция FLOAT

Функция имеет вид: FLOAT(X [,p]), где X - арифметическое выражение или переменная, которая должна быть преобразована к типу FLOAT с точностью представления p. Если p не задано, результат следующий:

Имеется	Переводится в
X FIXED BINARY (p)	FLOAT BINARY (p)
X FLOAT BINARY (p)	FLOAT BINARY (p)
X FIXED DECIMAL (p,q)	FLOAT BINARY (MIN (CEIL ((p-q)*3.322),24)

4.3. Строковые преобразования

PL/1 проводит преобразование между арифметическими и не арифметическими типами данных, если они встречаются в общих выражениях. Ниже приведен список использования встроенных функций для таких преобразований.

Преобразование	Встроенная функция
Арифметический тип в битовую строку	BIT [S [,L])
Арифметический тип в символьную строку	CHARACTER (S [,L])
Битовая строка в арифметический тип	BINARY (X [,p])
Битовая строка в символьную	CHARACTER (S [,L])
Символьная строка в арифметический тип	BINARY (X [,p]) FLOAT (X [,p]) DECIMAL X [,p [,q]])
Символьная строка в битовую	BIT [S [,L])

Следующие разделы описывают правила строковых преобразований.

4.3.1. Преобразование арифметического типа в битовую строку

Функция имеет вид: BIT(S [,L]), где S - арифметическое или строковое выражение или переменная и L константа типа FIXED BINARY. Сначала PL/1 преобразует ABS(S) к типу FIXED BINARY в соответствии с правилами арифметических преобразований, затем преобразует S в битовую строку длиной L. Если получившаяся строка короче L бит, она будет справа дописана нулевыми битами, если строка получилась длиннее L - лишние биты справа будут отброшены.

4.3.2. Преобразование арифметического типа в символьную строку

Функция имеет вид: CHARACTER(S [,L]) или CHAR(S [,L]), где S - арифметическое или строковое выражение или переменная и L константа типа FIXED BINARY.

Сначала PL/1 преобразует различные арифметические типы в строку следующим образом:

- Для FIXED BINARY(p)

PL/1 преобразует источник в FIXED DECIMAL(p'), где p'=CEIL(p/3.322)+1, затем преобразует FIXED DECIMAL(p') в символьную строку длиной p'+3 в формате, описанном ниже. Например, при преобразовании значения -32 типа FIXED BINARY(15) получится строка: ' -32'.

- Для FLOAT BINARY(p)

PL/1 преобразует мантиссу в FIXED DECIMAL(p'), где p'=CEIL(p/3.322). Результирующая строка будет длиной p'+7 символов в формате научной нотации (с показателем "E"), первый знак строки - минус, если число отрицательное или пробел - если число положительное. Следующая позиция содержит первую значащую цифру мантиссы с десятичной точкой, за которой следуют остальные p-1 цифр мантиссы. Затем идет признак порядка "E", знак порядка и две цифры порядка. Например, число типа FLOAT BINARY(24) со значением 250.1E1 преобразуется в строку: ' 2.501000E+03'. Для чисел двойной точности полная длина равна p'+8.

- Для FIXED DECIMAL(p,0)

Результирующая строка будет длиной p+3 и включает все цифры источника без незначащих нулей впереди со знаком минус, если число отрицательное или с первым пробелом, если число положительное. До длины p+3 строка будет дополнена пробелами. Например, число 330 типа FIXED DECIMAL(3) преобразуется в строку ' 330'. Ноль переведется в пять пробелов и одну цифру 0.

- Для FIXED DECIMAL(p,q) q>0

Результирующая строка будет также длиной p+3 и в том же формате, что и выше, но кроме этого будет включена десятичная точка и цифры после нее. Например, число -13.25 типа FIXED DECIMAL (5,2) будет преобразовано в строку ' -13.25'. PL/1 опускает все незначащие нули, за исключением нуля перед десятичной точкой.

После преобразования в строку, PL/1 дополняет строку пробелами справа, если она получилась короче заданной L. Если же строка получилась длиннее заданной L, она будет обрезана справа.

В данной реализации функция CHARACTER несколько модифицирована. Так как при преобразовании в текст число получается "прижатым" к правому краю строки, не всегда удобно трактовать параметр L как длину строки, начиная слева – там почти всегда будут пробелы. Поэтому, если L задана как отрицательная константа, считается, что это длина строки, но начиная, справа. Например:

```
dcl x fixed static init(55);
s char(15) varying;

s=char(x,4);    /* значение s '    ' */
s=char(x,-4);   /* значение s '  55' */
```

4.3.3. Преобразование битовой строки в арифметический тип

Функция BINARY описана выше. Когда задано преобразование битовой строки длиной n (1 <= n <= 64) в арифметический тип, PL/1 сначала преобразует строку в FIXED BINARY(63), затем это значение преобразуется к заданному арифметическому типу по обычным правилам, описанным выше. Например, битовая строка '1101'B будет преобразовано в число 13 типа FIXED BINARY.

4.3.4. Преобразование битовой строки в символьную

Функция CHARACTER описана выше. Когда задано преобразование битовой строки длиной n (1 <= n <= 64) в символьную длиной n, PL/1 преобразует каждый бит в символ "0" или символ "1". Если полученная строка короче заданной длины L, она будет дописана пробелами справа, если символьная строка получилась длиннее L, она будет усечена справа.

4.3.5. Преобразование символьной строки в арифметический тип

Используются следующие функции:

- FIXED(X [,p [,q]]) или DECIMAL (X [,p [,q]]) - возвращает значение типа FIXED DECIMAL. Если p не задано - принимается 15.

- BINARY(X [,p]) возвращает значение типа FIXED BINARY. По умолчанию p равно 15. Результат - только целая часть X.

- FLOAT(X [,p]) возвращает значение типа FLOAT BINARY. По умолчанию p равно 24. Если X ноль или содержит все пробелы - результат ноль. Если значение меньше минимально возможного или больше максимально возможного происходит прерывание UNDERFLOW(2) или OVERFLOW(2) соответственно.

При этих преобразованиях строка должна содержать правильные символы, иначе PL/1 сигнализирует прерыванием ERROR(1) о нарушении процесса преобразования.

Примеры преобразования строк:

Строка	Тип приемника	Результат
'000987'	FIXED BINARY (15)	987
'9.87'	FIXED DECIMAL (6,2)	0009.87
'-9.87E2'	FLOAT BINARY (24)	-9.87E2
'-9.87E2'	FIXED DECIMAL (9,2)	-0000987.00
'-9.87E2'	FIXED DECIMAL (5,0)	-00987
'-987.372'	FIXED DECIMAL (4,2)	ошибка ERROR (1)
'2X3'	FIXED BINARY (15)	ошибка ERROR (1)
''	FIXED BINARY (15)	0
' '	FIXED BINARY (15)	0

4.3.6. Преобразование символьной строки в битовую

Функция BIT описана выше. При этом преобразовании каждый символ "0" переводится в нулевой бит, а каждый символ "1" - в единичный бит. Следовательно, строка символов должна состоять только из нулей и единиц и, возможно, начинаться или оканчиваться пробелами, но не иметь пробелов между цифрами. Если это правило не соблюдено - происходит прерывание ERROR(1). Если полученная битовая строка оказалась короче заданной длины L, она будет дописана справа нулевыми битами, если оказалась длиннее L - усечена справа.

5. АГРЕГАТЫ ДАННЫХ

Агрегаты - это группы из нескольких элементов данных. В PL/1 есть два типа агрегатов данных: массивы и структуры.

- Массив — это упорядоченный набор данных, которые называются элементами массива. Все элементы массива имеют одинаковые атрибуты. Каждый элемент массива, в свою очередь, может быть или скалярным элементом данных или структурой. PL/1 позволяет или ссылаться на весь массив по имени или ссылаться на отдельный элемент массива, используя целые индексы, которые определяют положение элемента относительно начала массива.

- Структура — это набор данных разных типов. Элемент структуры может быть массивом. PL/1 позволяет ссылаться или на всю структуру по имени или на отдельный элемент структуры, используя составные имена, в которые входят имена элементов структуры.

Переменные, которые представляют агрегаты данных, называются переменными-массивами или переменными-структурами.

5.1. Описания массивов

Вы определяете переменную-массив, указывая как атрибут число элементов массива и порядок их организации в массиве. Эти атрибуты называются размерностями массива. Общий вид описания массивов:

DECLARE имя (пара индексов, ...) [список атрибутов] или
DECLARE имя [список атрибутов] DIMENSION(пара индексов, ...)

где имя - идентификатор переменной. Каждая пара индексов, определяющая число элементов, имеет вид: [L:] U, где L - нижняя граница массива и U – верхняя граница массива.
Если L не задана, она принимается за 1. L и U должны быть целыми константами и L должна быть не больше U. Список атрибутов устанавливает атрибуты данных, одинаковые для всех элементов массива. Порядок атрибутов неважен, но без указания слова DIMENSION пары индексов должны стоять впереди.

Число элементов в массиве для каждой размерности определяется как верхняя граница - нижняя граница + 1.

Общее число элементов в массиве равно произведению всех размерностей. Например, следующие описания эквивалентны:

```
declare A (3,4)          character(2);
declare A (1:3,1:4)      character(2);
```

Эти операторы определяют массив с двумя размерностями и каждый элемент которого - строка из двух символов. Такой массив можно представить как таблицу из 3 строк и 4 столбцов, в каждом клетке которой стоит 2 символа.

Оператор declare B (-2:5,-5:5,5:10) fixed; описывает трехмерный массив с диапазонами индексов от -2 до 5, от -5 до 5 и от 5 до 10 соответственно. Число элементов в каждой размерности восемь, одиннадцать и шесть соответственно. Таким образом, массив B содержит 528 элементов типа FIXED BINARY.

При описании массивов нужно соблюдать следующие правила:

- В PL/1 нет формальных ограничений на число элементов или размерностей в массиве. Однако реальный предел накладывает объем доступной свободной памяти и степень сложности выражений для индексов. В данной реализации индексов может быть не более 15.
- Все индексы при описании массива в DECLARE должны быть целыми константами (или символом «*» в массивах класса CONTROLED).
- Нижняя граница должна быть меньше или равна верхней.
- Выход индекса за границу массива во время работы программы приведет к непредсказуемым результатам (однако, компилятор контролирует индексы-константы). Динамический контроль индексов включается с помощью ключа компиляции "T".

5.2. Ссылки на массивы

В PL/1 ссылка на отдельный элемент массива должна быть индексированной, т.е. содержать список индексов, заключенный в круглые скобки. Для многомерных массивов, число таких индексов должно соответствовать числу размерностей массива.

Индекс в ссылке может быть любой переменной или выражением, которое PL/1 преобразует к целому типу. Например:

```
declare scores(20)       fixed binary;
declare (counter,total)  fixed binary;
total=0;
do counter=1 to 20;
  total=total+scores(counter);
end;
```

5.3. Инициация элементов массива

Используя атрибут INITIAL в описании массива, можно установить значения его элементов до начала выполнения программы. Например:

```
declare цвета(4) character(10) varying
static initial('красный','голубой','зеленый','желтый');
```

Заметим, что в данном примере вместо длины 10 можно было бы указать символ "*", и тогда бы компилятор выбрал наибольшую требуемую длину сам, по максимальному значению initial, (в этом случае 7).

Если требуется установить одинаковые значения в элементы массива, можно использовать коэффициент повторения в виде:

```
INITIAL (значение [, значение]);
```

где значение представлено в виде: [(коэффициент повторения)] константа.

Коэффициент повторения должен быть целой беззнаковой константой и обозначает число раз, которое надо использовать идущую за ним константу. Сама константа может быть любой арифметической или строковой константой, а для данных типа POINTER - именем переменной или встроенной функцией NULL. Для комплексных чисел константы обязательно заключаются в кавычки.

Например, оператор declare X (100) fixed binary static initial((100) 0); присвоит всем 100 элементам массива X значение 0.

Оператор declare grid(15) pointer static initial((15) null); присвоит всем 15 элементам массива grid нулевое значение указателя.

Оператор declare numbers(10) characters(10) static initial ((10) '1234567890'); присваивает десяти элементам массива значение '1234567890'.

Оператор declare numbers(10) characters(10) static initial ((5) '1234567890', (5) '0'); присваивает пяти элементам массива значение '1234567890' и пяти – значение константы '0'.

Указателям можно присваивать начальные значения именами переменных, например:

```
declare numbers(10) characters(10);
declare p (3) pointer static init(numbers(3),null,numbers);
```

PL/1 заполняет массив в порядке возрастания индексов. Правые индексы меняются быстрее. Например, описание:

```
declare test (2,2,2) fixed static initial(1,2,3,4,5,6,7,8);
```

эквивалентно заполнению элементов в следующем порядке:

```
test(1,1,1)=1;
test(1,1,2)=2;
test(1,2,1)=3;
test(1,2,2)=4;
test(2,1,1)=5;
test(2,1,2)=6;
test(2,2,1)=7;
test(2,2,2)=8;
```

5.4. Массивы в операторах присваивания

Только в некоторых случаях PL/1 допускает использование всего массива, как переменной в операторе присваивания. Оператор типа:

```
переменная-массив A = переменная-массив B;
```

допустим только в том случае, если это массивы одинакового типа данных и одинаковой размерности. В этом случае, каждый элемент массива B пересылается в соответствующий элемент массива A. Например:

```
declare A (20) fixed binary;
declare B (20) fixed binary;
A = B;
```

Отдельный элемент массива может быть обычным операндом выражения или присваивания, например:

```
declare A (10) fixed binary;
declare B (10) fixed binary static initial(0,1,2,3,4,5,6,7,8,9);
declare i fixed binary;
do i=1 to 10;
  A (i)=sqrt(B (i));
end;
```

Переменная-массив не может быть операндом арифметической операции, например сложения, т.е. операция типа C=A+B; недопустима, если A, B или C массивы, однако к одинаковым массивам могут быть применены операции "равно" и "не равно":

```
if A^=B then put list('массивы не равны !');
```

Данная реализация PL/1 не разрешает также операторы типа:

- переменная-массив = константа; (исключение массив=0;)
- переменная-массив = выражение;

Таким образом, данный пример недопустим:

```
declare A (10) fixed binary;
declare n fixed binary static initial (2);
A=n;                    // но допустимо A=0;
if A>1 then stop;  // но допустимо if A^=0 then stop;
```

Однако заданное действие можно получить с помощью команд:

```
do i=1 to 10;
  A (i)=n;
end;
```

В данной реализации можно также сравнивать массивы с нулем одним оператором.

5.5. Q

5.6. Массивы с изменяемыми границами

Стандарт «G» не предусматривает создание массивов с изменяемыми при работе программы границами. Однако в данной реализации такие массивы класса памяти CONTROLLED (CTL) все-таки возможны, хотя для простоты компилятора, они требуют некоторой дополнительной подготовки, что позволяет генерировать для них всегда один и тот же код. Этот код затем, в процессе выполнения программы, можно корректировать в зависимости от требуемых границ с помощью специальной встроенной функции «ретрансляции» ?RET, которая уже описана в самом верхнем блоке как:

```
dcl ?ret entry(ptr) external;
```

Имеется также встроенный массив ?index, уже описанный в верхнем блоке как:

```
dcl ?index(15,2) fixed(31) external;
```

Первая размерность этого массива 1:15, поскольку это максимально допустимое число размерностей массива в компиляторе. Служебная подпрограмма «ретрансляции», т.е. корректировки констант в командах,

использует текущие значения массива ?index. При запуске программы все значения границ в массиве ?index по умолчанию заданы единичными.

Передача новых значений границ через внешний массив ?index, а не через входные параметры подпрограммы ?ret сделана для большей гибкости их использования. Например, в этом случае не требуется повторять задания границ для массивов одинаковых размеров, не требуется перечислять нижние границы каждой размерности, если они всегда остаются единичными и т.п.

В случае какой-либо ошибки корректировки констант, подпрограмма ?ret не возвращает код ошибки, но инициирует перехватываемое исключение, поскольку далее обращение к массиву, указанному как входной параметр, даст непредсказуемые результаты.

«Динамические» границы, обозначаемые «*», могут быть только «старшими» размерностями и следовать подряд. При этом такие границы могут быть не только у массивов однородных элементов, но и у «массивов структур», т.е. агрегатов данных разных типов. «Младшие» размерности при этом могут оставаться константами, например:

```
dcl // структура-массив с изменяемыми границами
1 s(*,*)         ctl,
 2 x1            char(100) var,
 2 y1 (-1:25)  float,
 2 z1 (100)    fixed(31);
```

Для передачи в подпрограммы массивов с изменяемыми границами как параметров, учитывается тот факт, что такие массивы всегда неявно используют указатели. Но поскольку это служебные указатели, создаваемые компилятором, напрямую использовать их имена нельзя. Обращение к указателю без явного использования его имени возможно в языке PL/1 в случае применения встроенной функции ADDR, например:

```
dcl x(100) float based(p1),
    (p1,p2) ptr;
p2=addr(x);     // это эквивалентно p2=p1;
```

Таким образом, если нужно передавать как параметры массивы с «динамическими» границами, достаточно передать указатели на них с помощью ADDR, без явного использования имен служебных указателей:

```
call умножение_матриц(addr(a),addr(b),addr(c),m,n,q);
```

и тогда описание параметров таких подпрограмм становится единообразным, не зависящим от самих массивов:

```
dcl умножение_матриц entry(ptr,ptr,ptr,fixed(31),fixed(31),fixed(31));
```

Этот прием позволяет передавать «динамические» массивы в подпрограммы, но не позволяет «принимать» их внутри подпрограмм, поскольку тогда нужно присваивать указатели-параметры объектам класса «управляемой» памяти. Поэтому в компиляторе дополнительно разрешено использовать и в левой части присваивания встроенную функцию ADDR:

```
dcl x(*) float ctl,
    p1   ptr;
addr(x)=p1; // эквивалентно оператору <служебный указатель на x>=p1;
```

Ниже приведен пример решения типовой задачи умножения матриц, как массивов с заранее неизвестными границами. В данном примере применена описанная выше технология корректировки констант для выполнения универсального умножения матриц заданного размера.

«Динамические» массивы-матрицы создаются оператором ALLOCATE и передаются (неявно указателями) в универсальную процедуру умножения матриц. Корректировка констант в исполняемом коде производится как для фактических параметров A1, B1, C1, так и для формальных A, B, C.

Кроме этого, формальным параметрам присваиваются фактические значения с помощью разрешения использовать функцию ADDR в левой части присваивания.

Процедура «УМНОЖЕНИЕ_МАТРИЦ» может находиться в отдельном модуле и транслироваться автономно.

```
TEST:PROC MAIN;

DCL (A1,B1,C1)(*,*) FLOAT CTL;   // ДИНАМИЧЕСКИЕ МАТРИЦЫ
DCL (M1,N1,Q1)      FIXED (31);  // ЗАДАВАЕМЫЕ ГРАНИЦЫ
DCL (I,J)           FIXED (31);  // РАБОЧИЕ ПЕРЕМЕННЫЕ

//---- ДЛЯ ТЕСТА ЗАДАЕМ НЕКОТОРЫЕ ЗНАЧЕНИЯ ГРАНИЦ ----

M1=5; N1=4; Q1=3;

//---- КОРРЕКТИРУЕМ КОНСТАНТЫ A1(M1,N1), B1(N1,Q1), C1(M1,Q1) ----

?INDEX (1,2)=M1; ?INDEX(2,2)=N1;
?RET(ADDR (A1));              // ИЗМЕНЯЕМ КОМАНДЫ ДЛЯ A1
?INDEX (1,2)=N1; ?INDEX(2,2)=Q1;
?RET(ADDR (B1));              // ИЗМЕНЯЕМ КОМАНДЫ ДЛЯ B1
?INDEX (1,2)=M1;
```

```
?RET(ADDR (C1));                // ИЗМЕНЯЕМ КОМАНДЫ ДЛЯ C1

//---- СОЗДАЕМ МАТРИЦЫ A1(M1,N1), B1 (N1,Q1) И C1(M1,Q1) ----

ALLOCATE A1,B1,C1;

//---- ДЛЯ ТЕСТА ЗАПОЛНЯЕМ ИХ НЕКОТОРЫМИ ЗНАЧЕНИЯМИ ----

DO I=1 TO M1; DO J=1 TO N1; A1(I,J)=I+J; END J; END I;
DO I=1 TO N1; DO J=1 TO Q1; B1 (I,J)=I-J; END J; END I;

//---- УМНОЖЕНИЕ МАТРИЦ A1 И B1, РЕЗУЛЬТАТ - МАТРИЦА C1 ----

УМНОЖЕНИЕ_МАТРИЦ(ADDR (A1),ADDR(B1),ADDR(C1),M1,N1,Q1);

//---- ВЫДАЕМ ПОЛУЧЕННЫЙ РЕЗУЛЬТАТ ----

PUT SKIP DATA (C1);

FREE A1,B1,C1;

//======== УМНОЖЕНИЕ МАТРИЦ ЗАДАННОГО РАЗМЕРА ========

УМНОЖЕНИЕ_МАТРИЦ:PROC (P1,P2,P3,M,N,Q);

//---- ВХОД A (M,N) И B (N,Q), ОТВЕТ - МАТРИЦА C(M,Q) ----

DCL (P1,P2,P3)   PTR;             // УКАЗАТЕЛИ НА МАТРИЦЫ
DCL (M,N,Q)      FIXED (31);      // ЗАДАННЫЕ ГРАНИЦЫ
DCL (A,B,C)(*,*) FLOAT CTL;       // ДИНАМИЧЕСКИЕ МАТРИЦЫ
DCL (I,J,K)      FIXED (31);      // РАБОЧИЕ ПЕРЕМЕННЫЕ

//---- НОВЫЕ ОПЕРАТОРЫ ПРИСВАИВАНИЯ УКАЗАТЕЛЕЙ ----

ADDR (A)=P1;     // АДРЕС ДЛЯ МАССИВА A
ADDR (B)=P2;     // АДРЕС ДЛЯ МАССИВА B
ADDR (C)=P3;     // АДРЕС ДЛЯ МАССИВА C

//---- КОРРЕКТИРУЕМ КОНСТАНТЫ МАТРИЦ A (M,N), B (N,Q), C(M,Q) ----

?INDEX (1,2)=M; ?INDEX(2,2)=N;
?RET(ADDR (A));                 // ИЗМЕНЯЕМ КОМАНДЫ ДЛЯ A
?INDEX (1,2)=N; ?INDEX(2,2)=Q;
?RET(ADDR (B));                 // ИЗМЕНЯЕМ КОМАНДЫ ДЛЯ B
?INDEX (1,2)=M;
?RET(ADDR (C));                 // ИЗМЕНЯЕМ КОМАНДЫ ДЛЯ C

//---- СОБСТВЕННО УМНОЖЕНИЕ МАТРИЦ ----
```

```
DO I=1 TO M;
  DO J=1 TO Q;
       C(I,J)=0;
       DO K=1 TO N;
            C(I,J)+=A (I,K)*B (K,J);
       END K;
  END J;
END I;
END УМНОЖЕНИЕ_МАТРИЦ;
END TEST;
```

Результат запуска программы:

```
C1(1,1)=  2.600000E+01 C1(1,2)=  1.200000E+01 C1(1,3)= -2.000000E+00
C1(2,1)=  3.200000E+01 C1(2,2)=  1.400000E+01 C1(2,3)= -4.000000E+00
C1(3,1)=  3.800000E+01 C1(3,2)=  1.600000E+01 C1(3,3)= -6.000000E+00
C1(4,1)=  4.400000E+01 C1(4,2)=  1.800000E+01 C1(4,3)= -8.000000E+00
C1(5,1)=  5.000000E+01 C1(5,2)=  2.000000E+01 C1(5,3)= -1.000000E+01
```

5.7. Структуры

Структуры - это агрегаты данных, которые содержат данные различных типов. Вы можете использовать структуры для представления сложных объектов, отражающих внешние понятия решаемой задачи.

Структура состоит из элементов (members), которые могут быть или скалярными элементами данных или массивами скалярных элементов или другими структурами, называемыми в этом случае подструктурами или уровнями. Каждое описание структуры должно содержать следующее:

- имя главной структуры
- имена и атрибуты всех элементов структуры
- номера уровней структуры в иерархическом порядке

Общий вид оператора описания структуры:

```
DECLARE 1 имя [список атрибутов],
          [, [уровень] имя [список атрибутов]],
          …;
```

Уровень должен идти перед именем и отделяться от него хотя бы одним пробелом или табуляцией. Номер уровня главной структуры всегда единица. Описания каждого элемента структуры (вместе со своим номером уровня) должны отделяться от следующего запятой. Номер уровня подчиненного элемента структуры должен быть больше уровня предыдущего элемента. Если несколько элементов структуры имеют одинаковый номер уровня - номер уровня

можно вынести за скобки, но слева. Например, описание:
уровень-K элемент-1, уровень-K элемент-2, ... уровень-K элемент-n
можно записать как: уровень-K(элемент-1, элемент-2, ... элемент-n).
Оператор описания:

```
declare
1 A    based,
  2 (B fixed binary,
     C fixed character(2));
```

эквивалентен описанию:

```
declare
1 A   based,
  2 B fixed binary,
  2 C fixed character(2);
```

Замечание: *главный уровень структуры не может иметь атрибутов типов данных, но может иметь размерность массива, а также атрибут EXTERNAL и атрибуты класса памяти BASED, AUTOMATIC, STATIC, CTL или DEFINED.*

Пример описания структуры:

```
declare 1 bill,
2 name,
 3 last_name      character(20),
 3 first_name     character(20),
 3 middle_initial character(1),
2 address,
 3 street         character(20),
 3 city           character(10),
 3 state          character(3),
 3 zip            character(5),
2 charges,
 3 shop           fixed decimal(10,2),
 3 snack_bar      fixed decimal(10,2),
 3 misc fixed     decimal(10,2),
 3 dues fixed     decimal(10,2);
```

В программе имена промежуточных уровней (например: bill и name), могут быть опущены, если не возникает неоднозначности имен из-за того, что другая структура имеет такие же имена уровней и элементов и эта другая структура попадает в область действия этого имени, или же элементы на других уровнях структуры имеют такие же имена.

Для устранения неоднозначности применяются составные имена,

перечисляющие имена уровней структуры в порядке возрастания уровней. В таком составном имени, каждое имя уровня отделяется от соседнего точкой, и, возможно, пробелами и табуляцией. Только главные имена структур в одной области действия не должны совпадать.

Например, для структуры:

```
declare
1 A,
 2 B,
  3 C fixed,
  3 D fixed,
 2 BB,
  3 C fixed,
  3 D fixed;
```

ссылки на элементы C или D или A.C или A.D неоднозначны. Однозначность есть для ссылок B.C или B.D или BB.C или BB.D. Полные составные имена:

```
A.B.C
A.B.D
A.BB.C
A.BB.D
```

5.8. Смешанные агрегаты данных

Смешанные агрегаты данных - это или массивы, элементы которых являются структурами, или структуры, элементы которых являются массивами. Вы можете описать структуру, каждый элемент которой или несколько элементов имеют размерность массивов. Вы можете также описать массив, каждый элемент которого - структура, при этом после имени главной структуры указывается размерность массива. Таким образом, можно вкладывать массивы и структуры друг в друга. Например, оператор:

```
declare 1 student_list(100),
2 student_name,
 3 last_name                 character(20),
 3 first_name                character(20),
 3 middle_initial            character(1),
2 social_security_numbers    character(9),
2 country                    character(10),
2 grade (5)                  character(2);
```

определяет массив из 100 структур с именем student_list, в каждом элементе массива есть «подмассив» grade из 5 элементов. Ссылки на элементы массива

должны быть индексированы. Не обязательно указывать индекс у «своего» имени, однако индексы должны быть заключены в круглые скобки, отделяться друг от друга запятыми и следовать в правильном порядке. Например, ссылку на элемент grade можно представить в трех эквивалентных формах:

```
student_list(61).grade(3)
student_list.grade(61,3)
student_list(61,3).grade
```

5.9. Ссылки на смешанные агрегаты данных

На смешанные агрегаты данных можно ссылаться целиком по имени. Ссылка на элемент, входящий в смешанный агрегат, может быть индексирована и, возможно, уточнена составным именем. Такая ссылка определяет непрерывный участок памяти, в котором элементы расположены последовательно. Например, при описании:

```
declare 1 color(100),
          2 hue          character(10) varying,
          2 intensity    fixed binary;
```

элементы в памяти расположатся так:

```
hue(1)          \
intensity(1)    / color(1)
hue(2)          \
intensity(2)    / color(2)
...
hue(100)        \
intensity(100)  / color(100)
```

однако, при описании:

```
declare 1 color,
2 hue (100)             character(10) varying,
2 intensity(100)        fixed binary;
```

элементы в памяти расположатся так:

```
hue(1)          \
hue(2)           \ color.hue
...              /
hue(100)        /

intensity(1)    \
intensity(2)     \ color.intensity
...              /
intensity(100)  /
```

В первом случае, память для color(1) или color(100) непрерывна и это правильные ссылки, а память для color.hue не непрерывна и такая ссылка недопустима. Во втором случае наоборот, память для color.hue непрерывна и эта ссылка допустима, а ссылка color(1) - недопустима.

Выбор конкретного, наиболее удобного и понятного описания агрегата данных в программе, определяется целями и задачами при обработке этих данных.

5.10. Атрибут LIKE

Если описываемая структура или ее уровень копирует строение другой, уже описанной структуры (уровня), то для сокращения записи можно использовать атрибут LIKE («like» - подобный), например:

```
declare 1 x (1:100) based(p) like y;
```

Атрибут LIKE предписывает компилятору скопировать уровни, имена и атрибуты элементов указанного имени. Уровень и атрибуты самого имени (в нашем случае y) не копируются. Если уровни не соответствуют текущей структуре - они корректируются. Если нужно скопировать уровень структуры, его имя указывается обязательно полностью:

```
declare 1 x,
          2 y     like z.y,
          2 k     fixed;
```

Рекомендуется с помощью ключа компиляции "S" убедиться, что полученная таким образом структура соответствует ожиданиям.

6. ПРИСВАИВАНИЯ И ВЫРАЖЕНИЯ

6.1. Оператор присваивания

Оператор присваивания посылает в переменную значение выражения или константы. Общий вид оператора присваивания:

переменная = выражение;

где переменная - это скалярный элемент данных, переменная-массив, переменная-структура или псевдопеременная, или список этих объектов через запятую. Присваивание - единственный оператор, который распознается не по ключевому слову, а по символу "=".

Данная реализация PL/1 допускает несколько присваиваний в одном операторе, например оператор типа: A,B,C=1; правилен, причем для каждой переменной из списка возможно собственное преобразование к нужному типу. Заметьте, что в операторе присваивания вида: A=(B=C); второй знак равенства воспринимается как оператор "равно" и переменной A будет присвоено значение '1'B если B равно C или значение '0'B, если B не равно C.

Можно присваивать ноль всему массиву одним присваиванием.

Для четырех арифметических действий возможна еще одна форма присваивания в виде <имя> <операция>=<выражение>, например:

X+=1; эквивалентно X=X+1;

X (1)*=2+Z; эквивалентно X(1)=X(1)*2+Z;

т.е. левая часть повторяется после равенства с добавление заданной операции.

6.2. Оператор обмена

В данной реализации имеется также оператор обмена или оператор двунаправленного присваивания, который обозначается тремя символами «<=>». Например, оператор x <=> y; «обменяет» значения x и y. Переменные слева и справа в этом операторе должны обязательно иметь полностью совпадающие атрибуты и размерности.

Данный оператор переводится компилятором в одну или несколько аппаратных команд обмена процессоров x86-64. Он удобен для сортировок или работы с семафорами при параллельном программировании. При этом перед аппаратной командой обмена компилятор еще добавляет префикс блокировки шины, чтобы обмен обязательно выполнялся одной неделимой операцией.

6.3. Выражения

Выражение - это правильная комбинация операндов и операторов, которую PL/1 вычисляет при исполнении программы для получения значения. Синтаксические правила для выражений задаются порядком ссылок, операторов и круглых скобок в выражении. Ссылки или операнды могут быть константами, переменными или функциями. Операторы определяют те действия, которые нужно произвести, используя операнды, к которым они применяются. Круглые скобки выделяют части выражения. Последующие разделы описывают использование операндов, операторов и круглых скобок.

6.3.1. Префиксные выражения

Префиксное выражение состоит из одного префиксного оператора и следующего за ним операнда. PL/1 сначала обрабатывает эти операнды, которые сами могут быть выражениями, и затем возвращает результат префиксного оператора. Примеры префиксных выражений

```
^A              /*логическое отрицание A */
-SQRT (B)  /* минус квадратный корень из B */
```

6.3.2. Инфиксные выражения

Инфиксное выражение состоит из двух операндов, между которыми стоит инфиксный оператор. PL/1 сначала вычисляет операнды, которые сами могут быть выражениями, а затем применяет к ним указанный оператор и возвращает результат. Примеры инфиксных выражений:

```
A+B  /* сложение A и B */
C**2  /* возведение C в квадрат */
```

6.4. Приоритет выполнения операторов

В любом, не заключенным в круглые скобки выражении или подвыражении, PL/1 выполняет операторы в соответствии с их приоритетом. Ниже приводится этот порядок выполнения:

Оператор	Обозначение	Приоритет
Возведение в степень	**	1
Логическое НЕ	^	1
Префиксные операторы	- +	1
Умножение, деление	* /	2
Сложение, вычитание	+ -	3
Конкатенация (склейка)	|| или !! или \\	4
Отношения	= ^= < ^< <= > ^> >=	5
Логическое И	&	6
Логическое ИЛИ	| или ! или \	7

При разборе выражения в круглых скобках, PL/1 сначала ищет закрывающую скобку, а затем выполняет внутри скобок первый по старшинству оператор. Затем выполняется следующий по старшинству оператор и т.д., пока выражение в скобках не будет вычислено целиком. Если в выражении стоят операторы одинакового старшинства, PL/1 выполняет префиксные операторы и оператор возведения в степень справа налево, а остальные - слева направо. Например, обработка выражения без скобок:

```
2 + Z * X ** Y ** 2 / 5 - Q
```

пройдет следующим образом:

```
( 2 + ((Z * (X ** (Y ** 2 ))) / 5 )) – Q
```

6.5. Конкатенация (склеивание)

Инфиксный оператор «!!» склеивает битовые или символьные строки. Оба операнда должны быть одного типа и результат возвращается такого же типа. Длина строки результата равна сумме длин операндов. Для символьных строк, если любой операнд имеет атрибут VARYING, то и у результата будет установлен атрибут VARYING, например:

```
declare
A character (3),
B character (6) varying,
C character (20);
A='ABC';
B='ABCDEF';
C=A!!B;
```

переменной C будет присвоена строка из 9 символов 'ABCABCDEF'.

6.6. Операторы сравнения

В PL/1 операторы сравнения — это инфиксные операторы, которые определяют отношения между двумя операндами в алгебраическом смысле. Для вычисляемых операндов сравнения проводятся по общим алгебраическим правилам, для не вычисляемых (для FILE, POINTER, ENTRY, LABEL и агрегатов данных), а также комплексных чисел - только на "равно" и "не равно".

Для символьных и битовых строк возможно любое сравнение.

Если операнды имеют разные типы, PL/1 сначала преобразует их к общему типу по правилам преобразований и затем проводит сравнения, возвращая строку '1'B, если сравнение "истина" и строку '0'B, если сравнение "ложь".

При сравнении символьных строк, PL/1 сначала дополняет более кроткий операнд пробелами справа, а затем сравнивает слева направо попарно символы, как коды ASCII. В этом виде строчные буквы больше прописных, русские - больше латинских, цифры - меньше букв. Например, при сравнении строк 'JACK' и 'JACKSON', происходит сравнение кодов:

```
 J  A  C  K
4A 41 43 4B 20  20 20

 J  A  C  K  S  O  N
4A 41 43 4B 53  4F 4E
```

пятый символ в строке 'JACK' пробел с кодом 20, меньше чем S (код 53) и поэтому 'JACK' меньше, чем 'JACKSON'.

PL/1 сравнивает битовые строки, дополняя более короткий операнд нулевыми битами справа. Сравнение идет слева направо, нулевой бит считается меньше единичного. Например, '00010000' меньше, чем '00010001'.

6.7. Операторы для работы с битовыми строками

Оператор	Обозначение
Логическое НЕ (отрицание)	^ или ~
Логическое ИЛИ	| или ! или \
Логическое И	&

PL/1 применяет эти операторы к каждому биту операндов. Логическое НЕ меняет значение бит своего единственного операнда на противоположное, например, если задана строка A='01110010'B, то ^A='10001101'B.

Операторы ИЛИ и И применяются к двум операндам одинаковой длины. Если их длина разная - PL/1 дополнит более короткий операнд нулевыми битами справа. Результирующая строка имеет длину наибольшего операнда.

Операторы ИЛИ и И вычисляются по правилам Булевой алгебры:

X	Y	X ! Y	X & Y
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Все остальные функции Булевой алгебры легко реализуются с помощью встроенной функции BOOL (см. раздел 14.2).

6.8. Возведение в степень

PL/1 вычисляет возведение в степень как многократное умножение, если показатель степени - целая неотрицательная константа. В остальных случаях вычисление этого оператора проводится с использованием встроенных трансцендентных функций LOG и EXP. При обработке этого оператора PL/1 выделяет следующие специальные случаи:

- Если X = 0 и Y > 0, то X**Y = 0.
- Если X = 0 и Y < 0, то при выполнении оператора возникает прерывание ERROR(3).
- Если X < 0 и Y не целое, то при выполнении оператора возникает прерывание ERROR(3).

6.9. Псевдопеременные

В PL/1 существуют две встроенные функции SUBSTR и UNSPEC, которые могут использоваться как операнды в выражениях. Однако их также можно использовать и как операнд-приемник в левой части оператора присваивания. В этом случае SUBSTR и UNSPEC называются псевдопеременными потому, что они подобны обычным переменным в левой части присваивания.

6.9.1. Символьная SUBSTR

SUBSTR - это встроенная функция, позволяющая выделять подстроку из строки. Она может применяться в двух формах:

```
SUBSTR(строковая-переменная, i)
SUBSTR(строковая-переменная, i, j)
```

где строковая-переменная - это (возможно индексированная) ссылка на переменную типа CHARACTER или CHARACTER VARYING, а i и j - выражения типа FIXED BINARY.

SUBSTR с двумя аргументами выделяет подстроку, начиная с позиции i и до конца строки (позиция 1 - первый символ строки, позиция 2 - второй символ и т.д.). Результат не определен, если i находится за пределами длины строки.

SUBSTR с тремя аргументами выделяет подстроку, начиная с позиции i и длиной j символов. Результат не определен, если i или i+j находятся за пределами длины строки, где длина - это длина, описанная для данной переменной типа CHARACTER или текущая длина CHARACTER VARYING переменной.

Например, если строка word имеет текущее значение 'Josephine', то:

```
x = SUBSTR(word,1);    // x = 'Josephine'
x = SUBSTR(word,5);    // x = 'phine'
x = SUBSTR(word,1,4); // x = 'Jose'
```

SUBSTR-псевдопеременная подобна SUBSTR-функции за исключением того, что она находится в левой части оператора присваивания и используется несколько по-другому. Так, внутри такой SUBSTR уже не может быть строкового выражения, а только строковая переменная. Общий вид такой SUBSTR:

```
SUBSTR(имя-переменной, i)   = строковое выражение;
SUBSTR(имя-переменной, i, j)= строковое выражение;
```

SUBSTR с двумя аргументами пересылает внутрь указанной переменной, начиная с позиции i, подстроку, определяемую строковым выражением. Переменная заполняется с позиции i и до конца строки.

SUBSTR с тремя аргументами пересылает внутрь указанной переменной, начиная с позиции i, подстроку, определяемую строковым выражением. Пересылается j символов. Если i или i+j выходят за пределы переменной - результат не определен (скорее всего, при этом разрушится память, однако можно включить динамический контроль - ключ компиляции "T").

Одна и та же переменная может быть указана и справа и слева в операторе присваивания, в этом случае часть строки будет переписана. Например, если переменная word содержала строку 'Collegiate', то оператор:

```
substr(word,7)=substr(word,10,1);
```

изменит значение переменной word на 'College '.

6.9.2. Битовая SUBSTR

В PL/1 выделение битовой подстроки, аналогично выделению символьной, с некоторыми отличиями. Во-первых, длина битовой строки может меняться от 1 до 64 (что соответствует одному, двум, четырем или восьми байтам). Во-вторых, способ вычисления принят таким, что длина битовой подстроки должна быть константой. Таким образом, в обеих формах оператора:

```
SUBSTR(битовая-переменная, k)
SUBSTR(битовая-переменная, i, k)
```

где битовая-переменная - это переменная типа BIT, возможно с индексом, k – это целая константа от 1 до 63 и i выражение типа FIXED BINARY. Эффект от использования SUBSTR с битовыми строками такой же, как и с символьными: т.е. функция возвращает подстроку бит, если она стоит справа в выражении и устанавливает строку бит, если она стоит слева в операторе присваивания. В следующем разделе есть пример битовой SUBSTR.

6.9.3. Функция UNSPEC

UNSPEC - это встроенная функция, которая возвращает битовую строку внутреннего представления в машине своего аргумента. Общий вид UNSPEC:

переменная = UNSPEC(аргумент);

UNSPEC(аргумент)=<битовая строка>;

где аргумент - возможно индексированная ссылка на элемент данных, который занимает один, два или четыре байта памяти.

Замечание: *PL/1 не допускает выражение как аргумент UNSPEC, однако допускает один оператор без присваивания для записи произвольных кодов в командах программы. Например, контрольную точку командой INT 3 можно записать оператором UNSPEC('CC'B4);*

Когда UNSPEC используется как псевдопеременная, она стоит слева в операторе присваивания. PL/1 преобразует присваиваемое значение в битовую строку. Затем эта строка непосредственно пересылается в один два или четыре байта памяти, на которую указывает переменная внутри UNSPEC. Таким образом, UNSPEC позволяет прямо читать или устанавливать байты памяти как 8-ми, 16-ти или 32-х битовые строки.

Псевдопеременная UNSPEC часто используется для расширения языка и для взаимодействия с операционной системой. Пользуясь этой функцией, следует помнить, что она является машинно-зависимой и предполагает знание внутреннего представления данных. Однако в ряде случаев, это единственная возможность выполнить действия, не предоставляемые самим языком PL/1.

Следующий пример иллюстрирует прямое использование памяти. Две базированные переменные накладываются на фиксированные участки памяти с адресами FF80 и FFF0 и затем с помощью SUBSTR, содержимое 2 разрядов из одной области памяти переносится в другой.

```
declare
(P,Q)      pointer,
A bit(16) based(P),
B bit(8)   based(Q),
I          fixed;
I=4;
unspec(P)='FF80'b4;
unspec(Q)='FFF0'b4;
substr(B,4,2)=substr(A,I,2);
```

7. УПРАВЛЕНИЕ ПАМЯТЬЮ

7.1. Классы памяти

Каждая переменная в PL/1 ассоциируется с некоторым классом памяти. Класс памяти определяет, когда и как будет выделена память для этой переменной и будет ли переменная иметь свою собственную память или будет разделять ее с другими переменными.

В данной реализации PL/1 поддерживается шесть классов памяти:

- AUTOMATIC (назначается по умолчанию)
- BASED
- CONTROLLED
- DEFINED
- PARAMETER
- STATIC

Для классов AUTOMATIC и STATIC компилятор выделяет память переменным до начала работы программы и каждая переменная получает свою область памяти. Для классов BASED, CONTROLLED и PARAMETER компилятор устанавливает атрибуты переменных, но не выделяет им памяти. Память для таких переменных выделяется и освобождается в процессе работы программы. Класс DEFINED означает, что данная переменная будет иметь то же место в памяти, как и одна из ранее описанных переменных класса AUTOMATIC или STATIC, т.е. две или несколько переменных будут располагаться по одному адресу в памяти.

Классы памяти могут иметь скалярные переменные, массивы и главные уровни структуры. Данные типа ENTRY, FILE (без атрибута VARIABLE) и элементы структур не могут иметь класса памяти.

7.1.1. Класс памяти AUTOMATIC

Компилятор выделяет память переменной до начала выполнения программы. Память сохраняется за переменной до конца программы. В «полном» стандарте PL/1 этот класс памяти означает, что память переменным выделяется при входе в блок или процедуру и освобождается при выходе из блока. В данной реализации с целью упрощения и повышения быстродействия, принято статическое распределение памяти и для STATIC и для AUTOMATIC.

Только в случае рекурсивного вызова процедур для переменных класса AUTOMATIC действительно используется динамический механизм выделения и освобождения памяти.

7.1.2. Класс памяти BASED

Переменные этого класса памяти называются базированными и имеют ту память, на которую в данный момент указывает ассоциированный с ними указатель (типа POINTER). Указатель определяет адрес памяти, а сама переменная определяет, как PL/1 интерпретирует ту область памяти, с которой она начинается. Таким образом, базированная переменная с указателем эквивалентна обычной (не базированной) переменной.

Вы можете использовать базированную переменную как шаблон, накладываемый с помощью указателя на заданный участок памяти. Базированная переменная может иметь и свою собственную (выделенную только для нее) память.

Описание базированной переменной имеет вид:

```
DECLARE имя BASED [(указатель)];
```

где указатель - переменная типа POINTER без индексов или функция без аргументов, возвращающая значение типа POINTER.

Использование указателя базированной переменной может быть явным или неявным. Если в описании указатель не задан, то в программе каждая ссылка на эту переменную должна включать явное написание указателя в виде:

```
указатель-выражение -> переменная
```

где указатель-выражение - это выражение, возвращающее значение типа POINTER. Если в описании задан указатель, в ссылке на переменную его можно не писать. В этом случае, указатель определяется из описания. Следующий пример иллюстрирует использование явных и неявных указателей:

```
Main: procedure main;
declare
list_A (100) fixed binary based,
list_B (100) fixed binary based(list_B_ptr),
(list_A_ptr,list_B_ptr) pointer;
...
list_A_ptr -> list_A(47) = 0; /* явный указатель */
list_B (47) = 0;                    /* неявный указатель */
...
end Main;
```

Вы можете описать указатель в разных блоках и использовать его для ссылок на переменные в заданной области действия описания, например:

```
A:procedure main;
declare
(i,j) fixed binary,
p    pointer,
x    fixed binary based(p);
p=addr(i);
x=2;       /*  x совпадает с i */
B:procedure;
declare
p pointer;  локальная в B */
p=addr(j);
x=12;     /* x совпадает с i */
p->x=3;  /* x совпадает с j */
end B;
end A;
```

Примеры описаний базированных переменных:

```
declare A character(8) based;
declare B pointer      based(Q);
declare C fixed        based(P);
declare D bit(8)       based(F ());
```

7.1.3. Класс памяти CONTROLLED

Переменные этого класса подобны переменным класса BASED. Однако используются специальные служебные указатели (их максимальное число можно задать через реестр Windows), а не явно заданные. Кроме этого, для каждой переменной класса CONTROLLED (CTL) создается свой логический стек для хранения всех экземпляров этой переменной, для которых выделялась память, например:

```
declare x fixed ctl;      // переменная класса CONTROLLED
allocate x;               // создаем первый экземпляр переменной x
x=1;                      // присваиваем ей значение
allocate x;               // создаем следующий экземпляр переменной x
x=2;                      // присваиваем ему значение
free x;                   // x равно опять 1 (предыдущее значение)
free x;                   // x больше не существует
```

Для того, чтобы узнать, существуют ли еще экземпляры переменной класса CONTROLLED используется встроенная функция ALLOCATION(имя), возвращающая ’1’B, если еще есть экземпляры объекта с данным именем.

7.1.4. Класс памяти PARAMETER

Если переменная входит в список параметров при описании процедуры, ей автоматически присваивается класс памяти PARAMETER, но можно и явно указать этот класс. Память для параметров выделяется в момент входа в процедуру, при передаче ей списка параметров из вызывающей процедуры.

7.1.5. Класс памяти STATIC

Переменным этого класса памяти компилятор выделяет память до начала работы программы и сохраняет ее за ними до конца программы. Переменные класса STATIC могут быть инициализированы с помощью атрибута INITIAL. Этот атрибут задает компилятору константы, которые тот помещает в память, выделенную для данных переменных. Для данных типа POINTER такими константами могут быть имена переменных или имена уровней структур, тогда при начале работы программы, указатели будут показывать на память, выделенную переменным. Заметим, что все переменные с атрибутом EXTERNAL могут иметь класс памяти только STATIC.

Общий вид атрибута инициализации: INITIAL (значение [, значение] ...)
где значение имеет вид:

[(коэффициент повторения)] константа

Необязательный коэффициент повторения - это беззнаковая целая константа, показывающая, сколько раз нужно использовать следующую константу. Константа должна быть литеральной константой, совпадающей с типом инициируемых данных (или именем переменной для POINTER), т.е. арифметической константой, строковой константой или встроенными функциями NULL и DIM.

У массивов можно инициировать каждый элемент. Инициация начнется с первого элемента и будет идти в порядке возрастания правых индексов до исчерпания числа констант. Число констант не должно превышать число элементов в массиве. Каждый элемент структуры может быть индивидуально инициирован. Присвоения констант проводятся по правилам операторов присваивания. Например, если в INITIAL символьная строка короче длины инициируемой переменной, переменная справа будет дополнена пробелами.

Замечание: *В соответствии со стандартом PL/1 подмножества «G», атрибут INITIAL может быть только у переменных класса STATIC.*

Следующий пример иллюстрирует инициацию переменных STATIC:

```
declare A          fixed binary static initial(0);
declare B (8)      character(2) initial((8) 'AB') static;
declare
1 fcb              static,
 2 fcb_drive       fixed(7) initial(0),
 2 fcb_name        character(8) initial('emp'),
 2 fcb_type        character(3) initial('dat'),
 2 fcb_ext         bit(8) initial('00'b4),
 2 fcb_fill(19)    bit(8);
```

Существует разновидность конструкции STATIC INIT в виде атрибута VALUE. Отличие в том, что с таким атрибутом переменную после начальной инициации изменить уже нельзя, например:

```
dcl x1 fixed static init(1);
dcl x2 fixed value(1);
x1=2;                    // нормальная компиляция
x2=2;                    // предупреждение при компиляции
```

7.1.6. Класс памяти DEFINED

Переменные этого класса памяти переводятся в класс STATIC или AUTOMATIC и получают ту область памяти, которая уже ранее выделена переменной, указанной как база для DEFINED. Можно считать, что переменные DEFINED аналогичны переменным BASED, только у них указатель всегда неявен и всегда указывает на начало переменной-базы. Общий вид описания:

```
DECLARE имя DEFINED(имя_базы [+ смещение]);
```

где имя_базы, это имя переменной без индекса или имя главной структуры. Класс базы должен быть или STATIC или AUTOMATIC. База должна быть описана обязательно раньше и обязательно в этом же блоке. Она не может иметь атрибута EXTERNAL. Можно базировать на переменную, которая сама имеет класс DEFINED. Смещение - это целая беззнаковая константа, указывающая, на сколько байт от начала базы нужно отступить для начала памяти переменной DEFINED. По умолчанию смещение - ноль. Пример использования переменных

класса DEFINED:

```
A: procedure main;
declare
X (10) fixed(7) binary,
I       fixed(7) binary defined(X),
J       fixed(7) binary defined(I+9);
...
I=1;  /* присвоение первому элементу массива X */
J=1; /* присвоение последнему элементу массива X */
end A;
```

Замечание: *класс DEFINED реализован с отличиями от стандарта подмножества «G».*

7.2. Разрешение на использование памяти программой

В данной реализации необходимо указать, сколько памяти может использовать программа. По умолчанию считается, что программа использует столько памяти, сколько нужно для загрузки самого EXE-файла плюс небольшой буфер для открытия стандартного ввода/вывода. Для того, чтобы работали операторы ALLOCATE, рекурсия и т.п. нужно явно описать следующую встроенную переменную и задать ей значение объема памяти, например:

```
DCL ?MEMORY EXT FIXED (31) INIT (10000); // память в килобайтах
```

Обратите внимание, что присваивать значения можно только инициализацией, а не оператором присваивания, так как значение используется при загрузке программы, до начала выполнения операторов. Значение «-1» означает разрешить использование доступной всей памяти. По умолчанию значение ноль.

7.3. Оператор выделения памяти ALLOCATE

Оператор ALLOCATE явно выделяет память переменной класса BASED. Оператор имеет вид:

```
ALLOCATE переменная  [ SET (указатель)]
ALLOCATE (выражение) [ SET (указатель)]
```

Оператор ALLOCATE во время своего выполнения выделяет участок памяти из некоторой динамической области памяти, которая больше требуемой длины памяти для указанной переменной. Динамическая область уменьшается на эту величину. Если требуемую область памяти уже выделить нельзя - происходит прерывание ERROR(7).

Переменная в ALLOCATE должна быть без индексов, должна быть описана с атрибутом BASED и находится в области действия оператора ALLOCATE.

Адрес начала выделенной памяти возвращается как значение в указатель, записанный после ключевого слова SET. Если же конструкция SET не указана, адрес возвращается в указатель, на который базирована переменная в операторе DCL.

Во втором случае - выражение типа FIXED BINARY указывает, что такое число байт должно быть выделено из динамической области и адрес начала участка переслан в указатель.

Замечание: *оператор ALLOCATE (выражение) SET (указатель); отсутствует в стандарте подмножества «G».*

Выделенная память сохраняется за переменной до встречи соответствующего оператора FREE.

7.4. Многократное выделение памяти

При каждом выполнении оператора ALLOCATE происходит выделение памяти. Программа может выделить память переменной BASED несколько раз, записывая ее адрес в разные указатели или сохраняя предыдущее значение указателя, и затем ссылаться на любой из вариантов переменной, например:

```
declare names(5) character(10) based;
declare (P,Q)       pointer;
allocate names set(P);
P->names(1)='John';
...
allocate names set (Q);
Q->names(3)='Smith';
...
```

В этом примере, во время компиляции переменной-массиву не выделяется память. Компилятор выделяет память лишь для указателей P и Q. Во время выполнения программы, оператор ALLOCATE создает два варианта переменной-массива names и, в зависимости от указателя, можно обращаться к любому из вариантов.

Замечание: *если оператор ALLOCATE используется два раза для одного и того же указателя, старое значение указателя теряется, кроме случая переменных класса CONTROLLED.*

7.5. Оператор освобождения памяти FREE

Память, выделенная переменным BASED оператором ALLOCATE, освобождается и вновь возвращается в динамическую область с помощью оператора FREE, который имеет вид:

FREE [указатель ->] базированная переменная;

где базированная переменная - это переменная, предварительно получившая память из динамической области с помощью оператора ALLOCATE. Непредсказуемый результат получится при попытке освободить не выделенную оператором ALLOCATE память.

Если указатель не задан в операторе FREE, переменная должна быть описана с явным указателем или иметь класс CONTROLLED, в этом случае освобождается память, на которую указывает именно этот указатель в описании. Служебные подпрограммы управления динамической областью памяти автоматически склеивают смежные освобождающиеся участки памяти в непрерывный свободный участок.

Замечание: *после оператора FREE значения указателя и базированной переменной становятся неопределенными и попытка обратиться к ним без установки новых значений, даст непредсказуемые результаты. Для переменных класса CONTROLLED непредсказуемые результаты даст обращение к переменной, для которой встроенная функция ALLOCATION возвращает '0'b.*

Следующий пример иллюстрирует использование оператора FREE:

```
declare
(P,Q,R) pointer,
A       character(10) based,
B       fixed based(R);
...
allocate A set (P);
allocate B set (R);
allocate A set (Q);
...
free P->A;
free Q->A;
free B;
```

7.6. Встроенная функция NULL

Встроенная функция NULL возвращает значение, которое заведомо не является правильным адресом. Это значение используется как признак конца при организации связанных списков.

Связанный список - это такое построение данных, каждый элемент которых содержит адрес следующего элемента в списке. Связанные списки обычно используются там, где надо добавлять или исключать элементы, не перемещая в памяти весь список.

Функция NULL имеет вид: NULL [()].

Нет необходимости присваивать NULL всем указателям перед началом выполнения программы. Хотя это можно сделать при инициации с помощью атрибута INITIAL.

Указатель базированной переменной не может иметь своим текущим значением NULL, например, следующая программа недопустима:

```
declare A        pointer;
declare list(10) fixed binary based;
...
A=null();
A->list(10)=32767; /* это недопустимо */
```

В данной реализации имеется также встроенная константа NULL_PTR типа POINTER, которая всегда имеет значение NULL и предназначена для использования в качестве фактического параметра-указателя, когда его значение не используется или должно быть равно NULL. Если не используется параметр-число, то вместо него Вы можете написать ноль или любое другое число, однако, если не используется параметр-указатель вместо него нельзя написать NULL - это не указатель, а адрес! В таких-то случаях и используется встроенная константа NULL_PTR.

7.7. Встроенная функция ADDR

Встроенная функция ADDR возвращает адрес памяти переменной, заданной как аргумент ADDR. Общий вид ADDR:

```
ADDR (имя переменной)
```

В данной реализации возможно также использование функции ADDR в левой части оператора присваивания, для неявного присваивания значения

указателю, например:

```
dcl x fixed based(p);
dcl p ptr;
addr(x)=null; // эквивалентно p=null;
```

Замечание: *имя переменной должно иметь собственный адрес памяти или использовать указатель и не может быть промежуточным результатом, функцией (кроме SUBSTR) или оператором или константой или файлом-константой или меткой-константой или процедурой-константой (ENTRY).*

7.8. Общая память у переменных

Использование переменных BASED совместно с функцией ADDR или переменных класса DEFINED, позволяет в PL/1 использовать общую память для нескольких переменных. В этом случае, память у BASED-переменной не должна быть выделена с помощью ALLOCATE. Переменные BASED или DEFINED тогда можно рассматривать как шаблон, накладываемый на память, занятую некоторой другой переменной. Не обязательно вид и атрибуты переменных, использующих общую память, должны совпадать. Необходимо только, чтобы длина в битах у накладываемых переменных была не больше длины переменной, для которой была выделена память.

Следующий пример иллюстрирует использование общей памяти. Здесь на символьную строку накладывается массив битовых строк и затем содержимое символьной строки выводится как битовые строки (в шестнадцатеричном виде):

```
declare
i                fixed binary,
ptr              pointer,
word             character(8),
bit_vector (8) bit(8) based(ptr);
ptr=addr(word);
get list(word);
do i=1 to 8;
 put edit(bit_vector(i))(x(2),b4(2));
end;
```

Замечание: *типы данных важны при совместном использовании памяти. Данный пример подразумевал знание внутреннего представления данных в PL/1, а именно: восемь бит в одном символе. Таким образом, программа становится машинно-зависимой. Это несколько нарушает идею подмножества «G», ориентированного на написание машинно-независимых программ. PL/1 позволяет использовать общую память переменным, но тогда программы становятся зависимыми от реализации языка.*

7.9. Замечания по программированию с указателями

Базированные переменные и указатели являются мощным средством языка, поскольку они позволяют непосредственно управлять памятью. Однако, этим инструментом надо пользоваться аккуратно. Помните, что выделенная оператором ALLOCATE память сохраняется до оператора освобождения или до конца программы. На базированную переменную, получившую память от ALLOCATE, правильно ссылаться только внутри блока, где она описана.

Замечание: *PL/1 не может сообщить, что размер базированной переменной не соответствует размеру участка памяти, на которую она ссылается. Если базированной переменной присваивается значение, и эта переменная больше длины переменной, на которую она накладывается – память («куча») будет разрушена. В данной реализации имеются процедуры ?TEST_MEM и ?TEST_MEM_KVANT для контроля над целостностью памяти, используемой после операторов ALLOCATE и FREE.*

Наиболее типичные ошибки при работе с указателями:

- присвоение указателю значения NULL и использование его затем для ссылок на базированные переменные.
- использование указателя после освобождения памяти оператором FREE.
- использование указателя, значение которого потеряно, поскольку уже выходили из того блока, где он был описан.

8. УПРАВЛЯЮЩИЕ ОПЕРАТОРЫ

Обычно операторы в PL/1-программе выполняются последовательно, друг за другом. Однако можно изменить этот порядок, используя операторы условной и безусловной передачи управления, операторы обработки прерываний, а также операторные скобки (DO-группы).

8.1. Простая DO-группа

DO-группа - это последовательность операторов, заключенная между ключевыми словами DO; и END; Внутри DO-группы все операторы выполняются последовательно. Также внутри DO-группы не может быть локального описания данных (т.е. описание может быть, но оно не будет локальным).

Пример: if X=1 then do; Y=3.1415; Z=sin(A); end;

Заметим, что подмножество «G» не позволяет закрывать несколько DO-групп с помощью одного оператора END <метка>; (как реализация «полного» стандарта PL/1).

В данной реализации, для простоты, вместо ключевого слова "DO" можно писать символ "{", а вместо ключевого слова "END" - символ "}". Например:

```
IF X>0 THEN     {; Y=1; Z=2; };          полностью эквивалентно
IF X>0 THEN DO; Y=1; Z=2; END;
```

Кроме этого, в данной реализации перед DO-группой можно ставить префикс «1», показывающий, что данная группа должна выполниться в программе строго один раз. При исполнении такого оператора коды программы затем будут модифицированы и управление на данный DO-оператор больше ни разу не попадет:

```
1 do; x=1; y=1; end; /*вместо if flag then do; flag='0'b; x=1; y=1; end; */
```

8.2. Управляемая DO-группа (цикл)

Если DO-группа имеет заголовок, она превращается в циклическую конструкцию, тело которой (т.е. операторы внутри DO-группы) повторяется заданное число раз. Имеется 2 типа заголовков DO-групп:

```
DO WHILE(<условие>);
DO <переменная цикла>=<DO-спецификация>;
```

<Переменная цикла> - это скалярная переменная, <условие> - булевское выражение, а <DO-спецификация> - это одна из следующих конструкций:

<начальное значение> [TO <конечное>] [BY <изменение>]
[WHILE(<условие>)]
<начальное значение> [BY <изменение>] [TO <конечное>]
[WHILE(<условие>)]
<начальное значение> [REPEAT <повторение>] [WHILE(<условие>)]

<начальное значение> и <конечное значение> - это выражения, определяющие границы изменения <переменной цикла>;

<изменение> - выражение, задающее шаг изменения <переменной цикла> при каждом повторении DO-группы;

<повторение> - это выражение, которое необходимо вычислять при каждом повторении DO-группы;

В данной реализации можно объединять несколько заголовков в один «сложный», а также после END цикла можно для наглядности повторить начало заголовка DO-группы, т.е. или имя переменной (но не структуры) или REPEAT или WHILE. Использование циклов эквивалентно использованию операторов условия вместе с операторами GOTO.

8.2.1. Цикл DO-WHILE

Цикл имеет вид DO WHILE(S); ... END [WHILE]; и повторяется до тех пор, пока значение выражения S - истина. Если при входе в цикл S - не истина, цикл не выполнится ни разу.

8.2.2. Цикл DO-REPEAT

Цикл имеет вид DO I=X1 REPEAT(X2); ... END [I]; и повторяется до тех пор, пока внутри не встретится оператор выхода на метку вне цикла или оператор конца программы или оператор возврата. После каждого цикла выполняется вычисление выражения X2 и результат присваивается переменной цикла I.

В данной реализации имеется вырожденная форма DO REPEAT; ... END; являющаяся бесконечным циклом, эквивалентным конструкции

```
DO WHILE('1'B); ... END;
```

8.2.3. Цикл DO-REPEAT-WHILE

Цикл имеет вид DO I=X1 REPEAT(X2) WHILE(S); ... END [I]; и представляет собой объединение двух предыдущих конструкций.

8.2.4. Цикл DO-TO

Цикл имеет вид DO I=X1 TO X2; ... END [I]; и повторяется несколько раз, пока значение переменной I не пройдет все значения от X1 до X2 с шагом 1. Эквивалентен циклу DO I=X1 TO X2 BY 1; ... END; Если X1=X2, цикл выполнится один раз.

Имеется также форма этого цикла с «невидимой» переменной, например, когда нужно выполнить последовательность операторов заданное число раз, но переменная цикла, как таковая, внутри цикла не используется:

```
DO TO 10; … END;  // операторы внутри этого цикла повторятся 10 раз
```

8.2.5. Цикл DO-TO-BY

Цикл имеет вид
DO I=X1 TO X2 BY X3; ... END [I];
или
DO I=X1 BY X3 TO X2; ... END [I];
и повторяется несколько раз, пока значение переменной I не пройдет все значения от X1 до X2 с шагом X3. Шаг X3 может быть и отрицательным, это имеет смысл, если X1>X2.

8.2.6. Цикл DO-TO-BY-WHILE

Цикл имеет вид
DO I=X1 TO X2 BY X3 WHILE(S); ... END [I];
или
DO I=X1 BY X3 TO X2 WHILE(S); ... END [I];
и повторяется несколько раз, пока значение переменной I не пройдет все значения от X1 до X2 с шагом X3 и пока выражение S - истина. Если S при первом вхождении ложно - цикл не выполнится ни разу.

8.2.7. Цикл DO-BY

Если не указан верхний предел, цикл становится бесконечным:
DO I=1 BY 1; ... END [I];

8.2.8. Сложные (составные) циклы

В данной реализации заголовок цикла может состоять фактически из нескольких заголовков, разделенных запятыми. Признаком конца всех заголовков является точка с запятой. Каждый перечисленный через запятую заголовок может менять «формулу» переменной цикла и даже саму эту переменную. Рассмотрим несколько примеров.

Пример изменения переменной цикла по сложному закону:

```
DO I=1,3,5,15,25, I=100 TO 0 BY -4;
…
END I;
```

Например, два заголовка цикла можно использовать вместо конструкции типа UNTIL, когда первую итерацию нужно выполнять всегда:

```
DO F2=F (X), WHILE (ABS (F1-F2)>EPS);
    F1=F2;
    X+=DX;
    F2=F (X);
END WHILE;
```

Пример перемещения по «квадратной» траектории:

```
Y=0;
DO X=0 TO 99, Y=0 TO 99, X=100 TO 1 BY -1, Y=100 TO 0 BY -1;
    ПЕРЕМЕСТИТЬ(X,Y);
END;
```

По существу, здесь четыре цикла с двумя параметрами и когда один параметр меняется, второй – «замораживается» и остается таким, каким был при выходе из предыдущего цикла.

Пример вычисления полного приращения функции шести аргументов по приращениям аргументов.

```
X=X0; Y=Y0; Z=Z0; U=U0; V=V0; W=W0;
F0=F (X,Y,Z,U,V,W); DF=0;
DO X=X0+DX, Y=Y0+DY, Z=Z0+DZ, U=U0+DU, V=V0+DV, W=W0+DW;
    DF+=F (X,Y,Z,V,U,W)-F0;
   X=X0; Y=Y0; Z=Z0; U=U0; V=V0; W=W0;
END;
```

В этом примере громоздкое выражение из шести вызовов функции с разными параметрами заменено на одно обращение с одними и теми же аргументами. Хотя приходится шесть раз присваивать начальные значения аргументам, такой код компактнее и может оказаться даже эффективнее развернутого выражения.

8.2.9. Оператор LEAVE

Оператор LEAVE осуществляет безусловный выход из того цикла, внутри которого он находится. В подмножестве «G» этот оператор отсутствует. Нельзя использовать этот оператор вне цикла. Пример:

```
do while('1'b);        эквивалентен    do while('1'b);
...                                      ...
if FLAG then leave;                     if FLAG then goto M1;
...                                      ...
end;                                    end; M1:
```

Замечание: *в данной реализации имеется также конструкция LEAVE ON, которая служит для немедленного выхода из ON-оператора*.

8.2.10. Оператор CONTINUE

Оператор CONTINUE осуществляет безусловный переход на начало (на очередную итерацию) того цикла, внутри которого он находится. Данный оператор по действию противоположен оператору LEAVE. Нельзя использовать этот оператор вне цикла. В подмножестве «G» этот оператор отсутствует. Пример:

```
do while('1'b);        эквивалентен    do while('1'b);
...                                     ...
if FLAG then continue;                  if FLAG then goto M1;
...                                     ...
end;                                    M1: end;
```

8.3. Оператор условия

Общий вид оператора условия:

```
IF <выражение> THEN <действие 1> [ ELSE <действие 2> ]
```

Если <выражение> имеет значение истина - выполняется <действие 1>, иначе выполняется <действие 2>. Если <действие 2> опущено и

<выражение> ложно, никаких действий в программе не выполняется. <действие 1> и <действие 2> не должны содержать конструкций DECLARE, ENTRY, FORMAT и PROCEDURE.

Действие в операторе условия оканчивается точкой с запятой, которая входит в состав действия и поэтому в условном операторе не показана.

Операторы условий можно вкладывать друг в друга, при этом каждому слову THEN должен соответствовать свой ELSE, например:

```
if X=1 then Z=5; else
if X=2 then Z=8; else
if X=3 then do; Z=4; Y=2; end; else
            Z=0;
```

8.4. Оператор STOP

Оператор STOP может находиться в любом месте среди последовательности выполняемых операторов и даже внутри оператора PUT. Он безусловно оканчивает программу, закрывает все открытые файлы и возвращает управление в операционную систему. Можно использовать оператор в виде STOP (<значение>); где <значение> - переменная или выражение типа FIXED BINARY, возвращаемое как код ответа операционной системе, например:

```
ON ENDFILE(F) STOP(X+5); или
IF K>0 THEN STOP(K+1);
```

Замечание: *возвращающий значение оператор STOP(<значение>); отсутствует в подмножестве «G». Также отсутствует оператор EXIT, который может быть заменен оператором CALL <имя> STOP;*

8.5. Оператор GOTO

Оператор GOTO <метка> или GO TO <метка> безусловно передает управление в программе на указанную метку-переменную или метку-константу. Это может быть простая или индексированная ссылка. Нельзя передавать управление внутрь DO-групп, процедур и блоков. Отметим, что ошибку, связанную с такой передачей, компилятор может и не найти.

В данной реализации в операторе перехода можно опускать слово "TO".

8.6. Нелокальный переход GOTO

Возможна передача управления из блока или процедуры в ON-операторе. Обычно такие переходы не применяются, но иногда, для целей отладки или для обработки прерывания, необходим выход за пределы блока.
Пример:

```
P:procedure;
on endfile(SYSIN); goto EOF;  /* выход за пределы процедуры */
...
do while('1'B);
 get file(SYSIN) list(A (I)); I=I+1;
end;
end P;
...
call P;
EOF:
```

9. ОПЕРАТОРЫ ОБРАБОТКИ ПРЕРЫВАНИЙ

Во время выполнения программы могут возникнуть ситуации (например, деление на ноль), которые делают дальнейшее выполнение невозможным.

Программа должна как-то отреагировать на такую ситуацию, обычная реакция - это окончание программы. Вы можете задать свою реакцию на прерывание и даже смоделировать свое собственное прерывание, используя операторы ON, SIGNAL и REVERT.

Некоторые прерывания фатальные. Это означает, что невозможно вернуться в точку, где произошло прерывание, но можно передать управление в другое место с помощью нелокального перехода GOTO. Другие прерывания не фатальные и позволяют после обработки прерывания вернуться в ту точку программы, где произошло прерывание.

PL/1 различает три категории ситуаций:

- общие ошибки (ERROR-прерывания);
- ошибки арифметических операций (переполнение, антипереполнение (исчезновение порядка), деление на ноль);
- ошибки ввода/вывода (конец файла, не найден файл, неверный ключ, конец страницы);

Эти три категории разделяются на девять следующих типов:

- Тип 0 - общие ошибки (ERROR);
- Тип 1 - переполнение десятичного числа (FIXEDOVERFLOW)
- Тип 2 - переполнение двоичного числа (OVERFLOW)
- Тип 3 - антипереполнение двоичного числа (UNDERFLOW)
- Тип 4 - деление на ноль (ZERODIVIDE)
- Тип 5 - конец файла при чтении или исчерпание места на диске при записи (ENDFILE)
- Тип 6 - при попытке открытия не найден заданный файл или нет прав доступа (UNDEFINEDFILE)
- Тип 7 - неверное значение ключа при прямом доступе к файлу (KEY)
- Тип 8 - исчерпана длина страницы при выводе файла (ENDPAGE)

Каждый тип в пределе может иметь 255 подтипов, уточняющих причину прерывания. Подтипы 1-127 определяют, что прерывание фатальное. Подтипы 128-255 определяют не фатальное прерывание. Для ERROR-прерываний программисту доступны все подтипы для установки и перехвата (заметим, однако, что некоторые из подтипов зарезервированы для PL/1), для других типов

ситуаций доступны только некоторые подтипы и то только для перехвата, самому смоделировать конкретный подтип нельзя. Подтип равный нулю задает (или перехватывает) все прерывания данного типа.

9.1. ON-оператор

Общий вид оператора: ON <ситуация> <реакция>;

Оператор реакции на ситуацию становится активным, когда через него проходит управление в программе. Он действует до конца подпрограммы или пока не встретится соответствующий оператор отмены реакции REVERT. Если ON-оператор активен, в программе возникло указанная в нем <ситуация> и произошло прерывание, управление передается на операторы, записанные в нем как <реакция>. Если там записано несколько операторов, они должны быть заключены в операторные скобки BEGIN; и END; Например:

```
on endfile(F) begin; put list('КОНЕЦ ФАЙЛА'); goto start; end;
```

Нельзя выйти из ON-оператора с помощью RETURN, здесь может быть только оператор GOTO или оператор LEAVE ON. Если выход из ON-оператора не указан, после выполнения всех перечисленных в реакции операторов управление возвратится в программу в точку, расположенную за оператором, вызвавшим данную ситуацию.

Заметим, что число одновременно инициализированных ON-операторов в программе ограничено. Довольно распространенная ошибка - отсутствие соответствующих операторов отмены REVERT, тогда может произойти фатальная ошибка: "ПЕРЕПОЛНЕН СТЕК УСЛОВИЙ".

Прерывание обычно описывается конкретно, например ERROR(1), но можно задать и обобщенную реакцию: ON ERROR или ON ERROR(0) перехватывают все ERROR-прерывания.

Для выхода из ON-оператора без выполнения всех операторов его тела, можно использовать конструкцию LEAVE ON, которая эквивалентна переходу GOTO на последний END тела ON-оператора. Например:

```
on error(1); begin; if flag then leave on; ... end;
```

Если при входе в блок или процедуру устанавливается новый ON-оператор, при выходе из блока или процедуры будет автоматически восстановлен предыдущий ON-оператор, например:

```
A:
procedure options(main);
...
on endfile ... ;
on error(1) ... ;

B:
procedure;
...
on error(1) ... ;
...
end B:
...
end A;
```

здесь после выполнения второго ON ERROR(1), он будет действовать до конца процедуры B или до оператора REVERT ERROR(1), после чего опять будет действовать установленный в процедуре A ON ERROR(1).

Для одинаковых реакций можно при описании ON-оператора перечислить несколько условий через запятую, например:

```
on error(7), endfile(F1), error(1) go КОНЕЦ;
```

вместо:

```
on error(7)       go КОНЕЦ;
on endfile(F1)    go КОНЕЦ;
on error(1)       go КОНЕЦ;
```

9.2. Оператор SIGNAL

Оператор имеет вид: SIGNAL <прерывание>;

и моделирует возникновение указанного прерывания в программе, например SIGNAL ZERODIVIDE; моделирует все ситуации, связанные с делением на 0.

9.3. Оператор RESIGNAL

Данный оператор может находиться только внутри ON-оператора и по действию схож с оператором выхода LEAVE ON, с той разницей, что LEAVE ON просто оканчивает реакцию на данное событие, а RESIGNAL возобновляет поиск такого же ON-оператора, но установленного раньше («выше»). Обычно это

требуется, когда ON-оператор перехватил событие, но в результате анализа выяснилось, что это событие не для этого ON-оператора, а для какого-то предыдущего, и с той же причиной. RESIGNAL, по сути, отменяет текущий ON-оператор в пользу предыдущего, установленного ранее, если такой был. Если предыдущего ON-оператора нет – возбуждается стандартная реакция на данное событие.

9.4. Оператор REVERT

Оператор имеет вид: REVERT <прерывание>;
и отменяет последний инициализированный ON-оператор (или часть его, если было несколько ON-операторов через запятую). При этом восстанавливается действие предпоследнего ON-оператора или стандартная реакция, если ON-операторов больше нет. Заметим, что при выходе из процедуры все установленные в ней ON-операторы автоматически отменяются.

9.5. ERROR-прерывания

В PL/1-программе может быть до 255 ERROR-прерываний, они распределены следующим образом:

- номера 1 - 63 зарезервированы для PL/1 (фатальные)
- номера 64 - 127 оставлены для программиста (фатальные)
- номера 128 - 191 зарезервированы для PL/1 (не фатальные)
- номера 192 - 255 оставлены для программиста (не фатальные)

Разница между фатальными и не фатальными прерываниями в реакции PL/1-программы на то, что делать, если данная ситуация возникла, но ON-операторов для нее нет. Для фатальных прерываний произойдет окончание программы со стандартным сообщением об ошибке, для не фатальных прерываний никаких действий в программе не выполнится вообще, программа просто продолжит свое исполнение.

Операторы прерываний для ERROR-прерывания имеют вид:

```
ON      ERROR(целое выражение) тело оператора;
SIGNAL  ERROR(целое выражение);
REVERT ERROR(целое выражение);
```

<целое выражение> должно быть в диапазоне 0-255, если это выражение отсутствует или равно нулю, операторы прерывания действуют для всех ERROR-прерываний. В данной реализации ключевое слово ERROR можно опускать, оставляя лишь его номер в скобках, например, ON(35) BEGIN...

Пример ERROR-прерывания:

```
on error(1);
begin;
put skip list('Ошибочный ввод');
goto retry;
end;
retry:
get list(x);
```

оператор GET LIST читает данные из стандартного файла SYSIN в переменную X. Если данные были неверными, управление будет передано на ON-оператор, после чего будет выдано сообщение "Ошибочный ввод" и управление возвратится на начало ввода.

Вы можете инициировать фатальное или не фатальное прерывание. Например, оператор: SIGNAL ERROR(64); передаст управление на соответствующий ON-оператор или кончит программу со стандартным сообщением, если соответствующего ON-оператора нет.

В отличии от предыдущего, оператор SIGNAL ERROR(255); передаст управление на соответствующий ON-оператор или будет просто пропущен, если ON-оператора нет.

9.6. Прерывания при арифметических операциях

Имеется 4 прерывания:

FIXEDOVERFLOW (1)	ЦЕЛОЕ ПЕРЕПОЛНЕНИЕ(1)
OVERFLOW (n)	ВЕЩ. ПЕРЕПОЛНЕНИЕ(n)
UNDERFLOW (1)	АНТИПЕРЕПОЛНЕНИЕ (1) (исчезновение порядка)
ZERODIVIDE (n)	ДЕЛЕНИЕ НА 0 (n)

Необязательный номер n указывает, для какого подтипа данных возникла ситуация. Если этот номер не задан или задан 0, считается любой номер от 0 до 255 (как и для ERROR-прерываний).

Замечание: *все эти прерывания фатальные, то есть после обработки ON-оператора, управление может быть передано по GOTO или программа завершится.*

Условия возникновения прерываний:

ЦЕЛОЕ ПЕРЕПОЛНЕНИЕ(1)	Операции с данными DECIMAL
ВЕЩ. ПЕРЕПОЛНЕНИЕ(0)	Вычисление функции EXP
ВЕЩ. ПЕРЕПОЛНЕНИЕ(1)	Операции с данными FLOAT
ВЕЩ. ПЕРЕПОЛНЕНИЕ(128)	Переполнение при масштабировании FLOAT
АНТИПЕРЕПОЛНЕНИЕ(0)	Вычисление функции EXP (исчезновение порядка)
АНТИПЕРЕПОЛНЕНИЕ(1)	Операции с данными FLOAT (исчезновение порядка)
АНТИПЕРЕПОЛНЕНИЕ(128)	Исчезновение порядка при масштабировании FLOAT
ДЕЛЕНИЕ НА 0 (1)	Деление данных DECIMAL
ДЕЛЕНИЕ НА 0 (2)	Деление данных FLOAT
ДЕЛЕНИЕ НА 0 (3)	Деление целых данных

9.7. Встроенная функция ONCODE

Эта функция возвращает число типа FIXED BINARY (15), которое показывает код последнего возникшего в PL/1-программе прерывания. Если никаких прерываний не возникало, ONCODE возвращает 0. Пример использования:

```
on error begin; dcl X fixed; X=oncode();
if X=1 then do;
put list('НЕВЕРНЫЙ ВВОД'); goto RETRY;
end;
put list('ERROR',X);
end;
RETRY:
```

9.8. ON-операторы по умолчанию

По умолчанию заданы ON-операторы, выводящие стандартное сообщение об ошибке и оканчивающие программу. Для данных FIXED BINARY прерывание FIXEDOVERFLOW не перехватывается, хотя для FIXED DECIMAL перехватывается.

9.9. Прерывания при вводе/выводе

При работе с файлами может возникнуть 4 стандартные прерывания:

- ENDFILE(file) - КОНЕЦ ФАЙЛА

Подтип 1 - во время оператора GET
Подтип 2 - во время оператора PUT (уже нет места на диске)
Подтип 3 - во время оператора READ
Подтип 4 - во время оператора WRITE (уже нет места на диске)
Подтип 5 - во время пропуска строк (GET SKIP)
Подтип 7 - во время ввода CHAR VAR оператором GET EDIT

- UNDEFINEDFILE(file) - НЕ НАЙДЕН ФАЙЛ

Подтип всегда 1

- KEY(file) - НЕВЕРНЫЙ КЛЮЧ

Подтип 1 - во время установки ключа (KEY, KEYFROM)
Подтип 2 - во время получения ключа (KEYTO)

- ENDPAGE(file) - КОНЕЦ СТРАНИЦЫ

Подтип всегда 1

Замечание: *в данной реализации имеется также системная переменная ?ERROR (описываемая как dcl ?error fixed(15) ext;), в которую помещается код последней ошибки операционной системы при обмене. Т.е. при ошибке производится обращение к функции Win API GETLASTERROR. Например, если ?ERROR содержит 2 - не найден файл, 3 - не найдена папка и т.д. Если ошибок при обмене не было - переменная содержит ноль.*

10. ОПЕРАЦИИ ВВОДА-ВЫВОДА

PL/1 позволяет организовать обмен данными между PL/1-программой и устройствами внешней памяти, такими как принтер, экран, диск, сеть. Файл это, вообще говоря, набор данных. В PL/1 идентификатором файла будем также называть переменную, обозначающую внешнее устройство, передающее или принимающее набор данных программы.

Все идентификаторы файлов должны быть объявлены в программе, например: declare (F1, F2) file;

Можно описать идентификатор файла как FILE VARIABLE и затем динамически менять его значение в программе, присваивая ему идентификаторы файлов-констант, например:

```
declare X (1:3) file variable;
X (1)=F1; X(2)=F2;
```

В PL/1-программе все переменные FILE считаются переменными типа EXTERNAL. Поэтому описанные в разных процедурах разные файлы с одинаковыми идентификаторами превращаются в один и тот же файл. Однако FILE VARIABLE могут быть описаны одинаково в разных процедурах - это будут разные объекты.

10.1. Объявление файлов

Перед обменом требуется, чтобы файл был открыт, в PL/1 это можно сделать явно, задавая оператор OPEN, или неявно, обратившись к одному из операторов обмена: GET EDIT, PUT EDIT, GET LIST, PUT LIST, READ, WRITE.

10.1.1. Открытие файла оператором OPEN

Оператор имеет вид: OPEN [FILE] (<имя>) [атрибуты];

атрибуты могут быть следующими (* - назначаются по умолчанию):

- STREAM * или RECORD
- PRINT
- INPUT * или OUTPUT или UPDATE
- SEQUENTIAL * или DIRECT
- KEYED
- TITLE
- ENVIRONMENT
- PAGESIZE
- LINESIZE

Замечание: *в данной реализации можно опускать слово FILE в операторах OPEN, CLOSE, READ, WRITE, REWRITE, но тогда идентификатор файла в скобках должен идти первым:*
READ INTO(X) FILE(F);
READ FILE(F) INTO(X);
READ(F) INTO(X);

10.1.2. Атрибуты файлов

Атрибуты перечисляются в любом порядке и могут отсутствовать, но не должно быть конфликтующих между собой атрибутов.

STREAM - файл содержит данные в символьном виде

Данные могут быть представлены как текст с разбиением на строки и страницы, каждая строка оканчивается символами конца строки и возврата каретки. Каждая страница оканчивается символом конца страницы.

RECORD - файл содержит данные в двоичном коде

Они выбираются блоками указанного размера. В совокупности с атрибутом KEYED обеспечивает доступ к записям фиксированной длины по номеру, задаваемому ключом KEY типа FIXED BINARY (31).

PRINT - файл вывода на печать

Атрибут возможен только в совокупности с атрибутами STREAM и OUTPUT. В файле такого типа оператор PUT LIST не вставляет кавычки, букву "B" и т.д.

INPUT - работа с файлом только на чтение

Файл уже должен существовать, иначе возникнет прерывание UNDEFINEDFILE.

OUTPUT - работа с файлом только на запись

Файл создается вновь, а если он уже был, то обнуляется (если нет опции A в ENVIRONMENT). Если файл нельзя создать - возникнет прерывание UNDEFINEDFILE.

UPDATE - работа с файлом и на чтение и на запись

В файл можно записывать и из него считывать, если он не существовал, то создается. Не может быть с атрибутом STREAM.

DIRECT - файл прямого доступа

Для такого файла возможно чтение/запись по ключу, причем автоматически устанавливается атрибут RECORD.

SEQUENTIAL - файл последовательного доступа

Файл будет просматриваться последовательно от начала к концу. Опять к началу вернуться нельзя. (Для этого нужно закрыть файл и открыть его снова).

KEYED - файл с доступом по ключу

Файл содержит записи фиксированной длины. Они выбираются с помощью ключа KEY (типа FIXED BINARY(31)), соответствующему номеру записи. Фактически, файл превращается в массив записей. Автоматически устанавливается атрибут RECORD.

TITLE(c) - атрибут внешнего имени файла

Атрибут устанавливает связь между именем файла и внешним устройством. Если этот атрибут опущен, то связь будет с файлом <имя>.DAT, где <имя> берется из OPEN. Общий вид символьной строки: <путь> <имя>.<расширение>. Можно указывать имя и расширение в виде: $1.$1 и $2.$2, в этом случае имя и расширение берутся как первый и второй параметры из командной строки программы, например:

```
open file(F1) title('$1.$1');
open file(F2) title('$2.PL1');
```

ENVIRONMENT - атрибут задания среды для файла

Атрибут определяет размеры фиксированных записей при атрибуте RECORD и размеры внутренних буферов, например:

```
open file(F) environment(f(128),b(512));
```

Внутри этого атрибута могут стоять следующие опции:

A - для STREAM INPUT-файлов эта опция («A»-ANSII) имеет смысл только для операции чтения READ в строку CHARACTER VARYING. При задании этой опции, символ "^" в читаемом файле воспринимается, как и в строчных константах PL/1, т.е. гасит три старших разряда в следующим за ним символе, превращая его в управляющий.
Таким образом, ^I превращается в символ табуляции, ^M - в символ конца строки и т.д. В текстовом файле становится возможным записывать управляющие символы и другую информацию (например, атрибуты цветов символов), не используя непечатных символов. Сам символ "^" записывается как два подряд ("^^").
Если задана опция A для OUTPUT-файлов («A»-APPEND) и такой файл уже существует, строки дописываются в конец существующего файла. Опция A должна стоять первой.

F(i) - эта опция определяет файл с записями фиксированной длины, размером i байт, i должна быть типа FIXED BINARY(31). По умолчанию i равно 128 байтам. Размер записи может быть любой длины от 1 до 2**31-1.

B(i) - эта опция определяет размер буфера ввода/вывода размером i байт, i - должна быть типа FIXED BINARY(31). Обмен с файлом будет идти порциями по i байт. По умолчанию i равно 128 байтам. Если значение равно -1 – файл «отображается на память» средствами Windows.

Замечание: *для повышения скорости обмена и F и B выгодно делать кратными 512 байтам, поскольку физический обмен с дисками обычно идет секторами такой длины.*

Если есть F, должен быть атрибут KEYED, если есть B, но нет F, то считается, что записи переменной длины и KEYED не может быть. Если есть и атрибут F и атрибут B, F - должен стоять первым. Ключевое слово ENVIRONMENT можно заменять словом OPTIONS.

PAGESIZE(n) - размер страницы

Атрибут задает размер страницы при выводе, 0 - бесконечная (по умолчанию 0).

LINESIZE(n) - размер строки

Атрибут задает размер строки при выводе, 0 - не определен (по умолчанию 80).

10.1.3. Соответствие атрибутов

Поскольку некоторые атрибуты могут быть только вместе, но не все они могут быть указаны явно, компилятор автоматически добавляет нужные:

Заданные	Добавляемые
DIRECT	RECORD KEYED
KEYED	RECORD
PRINT	STREAM OUTPUT
SEQUENTIAL	RECORD
UPDATE	RECORD

10.1.4. Допустимые сочетания атрибутов

PL/1 проверяет все атрибуты в OPEN на совместимость. Совместимыми считаются:

```
STREAM INPUT      ENVIRONMENT TITLE
STREAM OUTPUT     PRINT ENVIRONMENT TITLE LINESIZE
STREAM OUTPUT     PRINT ENVIRONMENT TITLE LINESIZE PAGESIZE

RECORD INPUT      SEQUENTIAL ENVIRONMENT TITLE
RECORD OUTPUT     SEQUENTIAL ENVIRONMENT TITLE

RECORD INPUT      SEQUENTIAL KEYED ENVIRONMENT TITLE
RECORD OUTPUT     SEQUENTIAL KEYED ENVIRONMENT TITLE

RECORD INPUT      DIRECT KEYED ENVIRONMENT TITLE
RECORD OUTPUT     DIRECT KEYED ENVIRONMENT TITLE
RECORD UPDATE     DIRECT KEYED ENVIRONMENT TITLE
```

10.1.5. Атрибуты, по умолчанию приписываемые OPEN

Если нет оператора OPEN, но есть обращение к файлу, ему приписываются следующие атрибуты:

```
GET FILE (F) LIST            STREAM INPUT
PUT FILE (F) LIST            STREAM OUTPUT
GET FILE (F) EDIT            STREAM INPUT
PUT FILE (F) EDIT            STREAM OUTPUT
READ   FILE (F) INTO (v)     STREAM INPUT
WRITE FILE (F) FROM (v)      STREAM OUTPUT
READ   FILE (F) INTO(x)      RECORD INPUT SEQUENTIAL
WRITE FILE (F) FROM(x)       RECORD OUTPUT SEQUENTIAL

READ FILE (F) INTO(x) KEYTO (K)
      RECORD INPUT SEQUENTIAL KEYED ENVIRONMENT (F(128))

READ FILE (F) INTO(x) KEY (K)
      RECORD INPUT       DIRECT KEYED ENVIRONMENT (F(128))
      или
      RECORD UPDATE      DIRECT KEYED ENVIRONMENT (F(128))

WRITE FILE (F) FROM(x) KEYFROM (K)
      RECORD OUTPUT      DIRECT KEYED ENVIRONMENT (F(128))
      или
      RECORD UPDATE      DIRECT KEYED ENVIRONMENT (F(128))
```

x - переменная скалярного типа, массив или структура, но не CHARACTER VARYING, v - переменная типа CHARACTER VARYING.

10.1.6. Оператор CLOSE

Оператор CLOSE (<имя>); закрывает указанный файл, сбрасывает, если нужно, все буфера на диск. Теперь можно снова открыть этот файл или другой файл с этим идентификатором. Если файл не был открыт, оператор CLOSE игнорируется. Заметим также, что оператор OPEN для уже открытого файла вызывает ошибку ERROR(15) «УЖЕ ОТКРЫТ». Можно закрыть несколько файлов одним оператором:

```
CLOSE (F1),(F2),(F3);
```

10.1.7. Блок параметров файла (FPB)

Для каждого файла выделяется постоянная область памяти File Parameter Block (FPB), содержащая информацию о файле:

- имя внешнего устройства;
- номер позиции, с которой будет ввод/вывод данных для STREAM;
- текущее значение счетчика строк для STREAM OUTPUT;
- текущее значение счетчика страниц для PRINT;
- текущий номер записи;
- размер строки;
- размер страницы;
- размер записи фиксированной длины;
- размер внутреннего буфера;
- указатель на динамически выделяемую область File Control Block (FCB);

Каждый открытый файл добавляется в связанный список открытых файлов. При закрытии, файл исключается из списка и память, выделенная под FCB и буфер, освобождается. В конце программы все оставшиеся в списке файлы автоматически закрываются. Подробно структуры данных типа FILE описаны в разделе 18.8.

10.2. Прерывания при работе с файлами

10.2.1. Прерывание ENDFILE

Данная ситуация возникает, когда встретился физический конец файла DIRECT или символ ^Z в файле типа STREAM, а также, когда не хватает места на диске при записи.

10.2.2. Прерывание UNDEFINEDFILE

Данная ситуация возникает, когда невозможно открыть или создать файл оператором OPEN, например неверно задано имя в TITLE.

10.2.3. Прерывание KEY

Данная ситуация возникает при неверном значении ключа для файлов типа KEYED.

10.2.4. Прерывание ENDPAGE

Данная ситуация возникает в момент, когда номер очередной строки от начала страницы при выдаче файла типа PRINT превысил размер, указанный в PAGESIZE. Первоначальный номер строки до выдачи всегда 0 и он автоматически увеличивается на 1 при каждой выдаче строки. Если был задан PAGESIZE (0) - ситуация не наступит никогда. Номер текущей строки сбрасывается в 1, если выдается символ FF (код 0C или ^L) или происходит вывод оператором PUT PAGE, а также по достижении 32767.

10.2.5. Установленные перехваты прерываний по умолчанию

Если нет соответствующих ON-операторов, ситуация ENDPAGE вызывает исполнение оператора PUT PAGE, а затем работа программы продолжается. Остальные ситуации прекращают выполнение программы и выдают стандартные сообщения.

10.3. Встроенные функции при обмене

10.3.1. Функция LENGTH

Оператор LENGTH (<имя>); возвращает длину файла, открытого как INPUT или UPDATE. Для других типов файлов длина не определена. Удобно использовать этот оператор для файлов, отображенных на память. Этой функции для файлов нет в стандарте языка.

10.3.2. Функция ONFILE

Эта встроенная функция возвращает строку - внутреннее имя того файла, при работе с которым возникла последнее прерывание, если прерываний не возникало - возвращается пустая строка. Если это ошибка преобразования - возвращается имя файла, активного в этот момент. Пример:

```
on error(1) begin; put list('ОШИБКА В ФАЙЛЕ ',onfile()); goto m; end;
```

10.3.3. Функция ONLOC

Эта встроенная функция возвращает имя последней вызванной перед прерыванием процедуры, в случае, если компиляция производилась с ключом «R». Обратите внимание, что тогда все модули нужно компилировать с этим ключом, иначе будет выдано имя последней процедуры только из тех модулей, которые транслировались с этим ключом, а не реально последняя вызываемая процедура.

10.3.4. Функция ONTERM

Эта встроенная функция возвращает номер последнего считанного элемента в операторах GET DATA или GET LIST. В случае, если эти операторы содержат длинные списки элементов, бывает трудно понять, при чтении какого именно элемента из списка произошла ошибка.

10.3.5. Функция ONKEY

Эта встроенная функция возвращает номер ключа, при котором возникла последнее прерывание KEY, работает только внутри ON KEY, пример:

```
on key(F1) put skip list('НЕВЕРНЫЙ КЛЮЧ ',onkey());
```

10.3.6. Функция PAGENO

Эта встроенная функция возвращает текущий номер страницы для своего файла.

10.3.7. Функция LINENO

Эта встроенная функция возвращает текущий номер строки для своего файла. Если произошло прерывание ENDPAGE, то в этот момент номер строки на 1 больше, чем указано в PAGESIZE.

10.3.8. Функция FILEOPEN

Эта встроенная функция имеет параметром переменную типа файл и возвращает ’1’B, если в данный момент этот файл открыт и ’0’B в противном случае.

10.3.9. Стандартные файлы системного ввода/вывода

Если в операторах обмена не указывается идентификатор файла, то подразумевается, что ввод идет из файла SYSIN, а вывод в файл SYSPRINT. Эти переменные не нужно описывать в DECLARE и не нужно открывать. Они открываются по умолчанию операторами:

```
open file (SYSIN)      stream input      environment(B(128)) title('$con')
linesize(80);
open file (SYSPRINT) stream output print environment(B(128)) title('$con')
linesize(80) pagesize(0);
```

Однако, если надо что-либо изменить в их настройке, придется их описывать и открывать как обычные файлы.

10.4. Методы доступа к файлам

10.4.1. Метод STREAM

Это «потокоориентированный» ввод/вывод применяется в операторах GET LIST, PUT LIST, GET EDIT, PUT EDIT.

Файл представляет собой последовательность символов, возможно разделенных символами концов строк. При чтении и записи таких файлов, данные преобразуются в текстовый вид.

10.4.2. Метод RECORD

Это ввод/вывод данных, которые представляют собой отдельные записи. Никаких преобразований при обмене не производится, поэтому представление данных здесь зависит от их внутреннего представления в PL/1 и компьютере.

10.4.3. Метод отображения файлов на память

Windows предоставляет удобное средство работы с файлами: их отображение на память, т.е. работа с файлами как с обычными массивами программы. Для отображения файла на память в данной реализации нужно открыть его с указанием в атрибуте ENVIROMENT параметра B(-1). После этого один раз нужно обратиться к оператору READ с установкой указателя. Описав базированный на этот указатель массив или структуру, теперь можно работать с файлом как с массивом. При окончании программы или после оператора CLOSE, все исправления запишутся в файл, если он имел атрибут UPDATE.

Пример работы с файлом, отображенным на память:

```
DCL F FILE, P PTR, X (10000) FIXED BASED (P);
//---- открыли с отображением на память ----
OPEN FILE (F) UPDATE RECORD TITLE ('TEST.DAT') ENV (B (-1));
READ FILE (F) SET (P);      // установили начало в указатель
X (500)=1;                  // меняем содержимое файла
CLOSE FILE (F);             // записываем исправления в файл
```

10.4.4. Стандартные устройства

В качестве имен файлов в опции TITLE могут быть указаны следующие стандартные устройства. Признаком устройства является символ '$':

- '$CON' - стандартный ввод или вывод (обычно клавиатура и экран)
- '$RDR' - вспомогательный ввод (из COM1 или AUX)
- '$PUN' - вспомогательный вывод (в COM1 или AUX)
- '$LST' - вывод на печать (в LPT1)

11. ТЕКСТОВЫЙ (STREAM) ВВОД/ВЫВОД

PL/1 поддерживает три формы такого «потокового», а по сути, текстового обмена:

- LIST-управляемый - данные преобразуются в форматы согласно своим спецификациям;
- EDIT-управляемый - данные преобразуются в форматы, которые указывает программист;
- строчный обмен (для переменных типа CHARACTER VARYING) с использованием операторов READ и WRITE. Этой формы нет в «полном» стандарте PL/1.

Для всех этих форм действуют следующие правила:

- Номер колонки, строки и страницы первоначально установлены в 1;
- Каждый символ конца строки или страницы сбрасывает номер колонки в 1;
- Если символ при вводе/выводе специальный (код <32), номер колонки продвигается на 1;
- При выводе, если номер колонки превысил размер строки, программа выдает символ конца строки, увеличивает номер строки и сбрасывает номер колонки в 1;
- Если номер строки превысил размер страницы, возникает ситуация ENDPAGE. Если на эту ситуацию нет ON-оператора, программа выдает символ конца страницы, увеличивает номер страницы на 1 и сбрасывает номер строки и колонки в 1;
- Все значения, которые определяют размеры строк и страниц, количество пропускаемых символов, табуляцию и т.д., должны иметь тип FIXED BINARY;

Список переменных при вводе/выводе перечисляется через запятую и может включать скалярные переменные, структуры и массивы. Также можно использовать оператор цикла или написать DO-группу прямо внутри списка переменных, следующие примеры эквивалентны:

```
declare A (10) fixed;

do I=1 to 10; put list(A (I)); end;
put list((A (I) do I=1 to 10));
put list(a);
```

Список переменных при выводе может включать оператор STOP, что

иногда позволяет несколько упростить текст программы. Естественно, после оператора STOP все переменные списка вывода игнорируются. Следующие примеры эквивалентны:

```
on error (7) begin; put skip list('не хватает памяти'); stop; end;

on error (7) put skip list('не хватает памяти',stop);
```

Удобно применять данный прием при отладочных выдачах:

```
if x<0 then put skip data(x,y,z,stop);
```

вместо обычной выдачи, требующей «лишних» (при чтении программы) скобок:

```
if x<0 then {; put skip data(x,y,z); stop; };
```

11.1. LIST-управляемый ввод/вывод

При вводе должны соблюдаться следующие условия:

- Выводимые/вводимые данные могут быть арифметическими константами, символьными или битовыми строками.
- За каждым данным должен следовать разделитель в виде последовательности пробелов или запятой, или символа конца строки.
- Знаки табуляции воспринимаются как пробелы.
- Символьные строки, которые сами могут содержать пробелы или запятые должны быть заключены в апострофы, иначе программа воспримет эти символы как разделители.
- Если первый символ в файле, отличный от пробела, это запятая, или в файле встретились две запятые подряд, между которыми нет символов или только пробелы, считается, что это пустое поле без данных и в очередную переменную ничего не передается. Значение этой переменной останется таким же, как и до ввода.

11.1.1. Оператор GET LIST

Оператор читает данные LIST-управляемого потока и имеет вид:

```
GET [ FILE(<имя>) ] [ SKIP [(nl)] ] LIST(<список ввода>);
```

Атрибуты FILE и SKIP могут идти в любом порядке, LIST - обязателен и

последний. Если атрибута FILE нет, подразумевается FILE(SYSIN). Если указан атрибут SKIP, программа проигнорирует входные символы, пока не встретится nl символов конца строки. Если nl не задан - считается nl=1.

После выполнения оператора, переменным в списке ввода будут присвоены значения из входного потока, номер колонки установится на первый не введенный еще символ. Символьные строки во входном потоке могут быть заключены в апострофы, в переменные эти апострофы не попадут, аналогично в битовые переменные не попадут апострофы и признак системы счисления B. При вводе с клавиатуры нужен только один первый апостроф, концом символьной строки в этом случае может служить клавиша ВВОД (Enter).

Имеется разновидность оператора GET LIST - упрощенный оператор GET DATA. Этот оператор работает аналогично описанному выше, однако пропускает в данных все символы до очередного знака равенства, после которого и начинают считываться данные. Если оператор get list(x,y,z); требует записи данных 0, , 1E-15,... , то оператор get data(x,y,z); требует записи данных x=0, y=, z=1E-15,... , что иногда бывает удобнее и нагляднее. Однако, в отличие от «полного» стандарта PL/1, данные здесь должны идти в строгом порядке, и недопустима точка с запятой как разделитель.

11.1.2. Оператор PUT LIST

Оператор пишет данные в выходной LIST-управляемый поток и имеет вид:
PUT [FILE(<имя>)] [SKIP [(nl)]] [PAGE] LIST(<список вывода>);

Атрибуты FILE, SKIP, и PAGE могут идти в любом порядке, LIST - обязателен и последний. Если атрибута FILE нет, подразумевается FILE(SYSPRINT).

Если указан SKIP, то программа запишет nl символов конца строки, если nl=0, конец строки не запишется, но номер колонки сбросится в 1. Если nl не указан, считается, что nl=1. Атрибут PAGE может быть только для файлов типа PRINT, в этом случае пишется символ конца страницы (^L), номера строки и колонки сбрасываются в 1.

Примечание: *В данной реализации SKIP можно указывать и в списке вывода: PUT LIST(SKIP,X,Y,Z,SKIP); Поэтому нельзя вывести оператором PUT переменную с именем SKIP, зато можно перевести строку в конце оператора PUT, что невозможно, если следовать стандарту* .

Данные из списка вывода преобразуются в текст и выдаются в файл, в качестве разделителей используются пробелы. Если очередной элемент не

помещается на строке, он переносится в начало следующей. Если встретилась символьная строка, которая вообще больше заданного размера строки, на строке помещается ее часть, а остальное переносится на следующую строку. Если исчерпана страница, возникает ситуация ENDPAGE. Символьные строки заключаются в апострофы, однако для файлов типа PRINT этого не делается. Битовые строки выводятся в апострофах и затем следует буква B.

Для совместимости программ и удобства отладки, в данной реализации имеется оператор PUT DATA, который аналогичен оператору PUT LIST, но при этом еще выводит имена (идентификаторы) переменных, например, если:

```
PUT SKIP LIST(X,Y);
```

Выводит `1024 768`, то

```
PUT SKIP DATA(X,Y);
```

Выведет `X= 1024 Y=768`

Имеется также встроенная переменная ?F_E, типа FIXED которая определяет, сколько знаков мантиссы для переменных типа FLOAT нужно печатать при выводе операторами PUT LIST и PUT DATA. Если эту переменную присвоить нулю (значение по умолчанию), то выводится 14 знаков.

11.2. Строчный обмен

Этого обмена нет в других реализациях PL/1.

11.2.1. Строчный оператор READ

Оператор читает строку переменной длины из входного потока и имеет вид:

READ [FILE(<имя>)] INTO(v); v - обязательно CHARACTER VARYING.

Если атрибута FILE нет, подразумевается FILE(SYSIN). Символы читаются, пока не заполнится переменная v или пока не встретится символ конца строки. Сам символ конца строки в v не попадает. Пример:

```
dcl F file,
1 BUFFER
  2 BUFF CHAR (254) VAR;
read file(F) into(BUFFER); // не CHAR VAR, поэтому чтение не строчное
read file(F) into(BUFF);      // это CHAR VAR, строчный обмен
```

11.2.2. Строчный оператор WRITE

Оператор пишет строку переменной длины в выходной поток и имеет вид:

WRITE [FILE(<имя>)] FROM(v); v - обязательно CHARACTER VARYING.

Если атрибута FILE нет, подразумевается FILE(SYSPRINT). Никаких управляющих символов не добавляется, если это необходимо, их нужно явно задать в строке, например, используя символ "^" : 'ПРИМЕР СТРОКИ ^M^J'.

11.3. EDIT-управляемый ввод/вывод

Этот ввод/вывод полностью аналогичен LIST-управляемому, за исключением того, что после списка ввода/вывода указывается список форматов, по которым должны быть преобразованы данные. Списков ввода/вывода и форматов может быть несколько. Общий вид операторов:

```
PUT [FILE(<имя>)] [SKIP[(nl)]] [PAGE ]
EDIT(<список вывода>)(<формат>) [ (<список вывода>)(<формат>)…];
GET [ FILE(<имя>) ] [ SKIP [(nl)] ]
EDIT(<список ввода>)(<формат>)   [ (<список вывода>)(<формат>)…];
```

Для удобства можно не писать один длинный список переменных и один длинный список форматов к ним, а разбить на одиночные переменные и форматы к ним, например:

```
PUT SKIP FILE(F) EDIT
(X)(F(5))
(Y)(A(10))
(Z)(SKIP,B4(8));
```

11.3.1. Список форматов

Элементы списка разделяются запятыми, имеется три типа элементов:

- элементы, описывающие формат, по которому преобразуются данные;
- элементы, определяющие место, где будет располагаться объект обмена;
- ссылка на другой список форматов.

Перед каждым элементом списка формата может стоять коэффициент повторения - константа от 1 до 254, он показывает сколько раз нужно повторить очередной, следующий за ним элемент. Если его нет - считается равным 1. От элемента списка форматов коэффициент повторения отделяется пробелами, а не

запятой.

11.3.2. Форматы преобразования данных

A(w) - выборка w символов из очередного алфавитно-цифрового поля лишние символы справа отсекаются, недостающие заполняются пробелами. Если w нет, то за размер поля принимается длина символьной переменной.

Переменная	Формат	Результат	
Byte	A(6)	'byte'	если ввод
Космотехника	A(12)	'Космотехника'	если ввод
String	A	'String'	если ввод
abcdef	A(6)	abcdef	если вывод
abcdef	A(3)	abc	если вывод
пробел	A(4)	4 пробела	если вывод

B[n][(w)] - для битовых строк n - число бит для одной цифры, w - ширина поля. n может принимать значения 1,2,3,4, если не указано - считается 1. Если преобразование выполнить невозможно - возникает ситуация ERROR(1).

Переменная	Формат	Результат	
00101	B(5)	'00101'B	если ввод
22	B2(2)	'1010'B	если ввод
7C4	B4(3)	'011111000100'B	если ввод
'00'B	B	'00'	если вывод
'1'B	B(4)	'0001'	если вывод
'011101'B	B3(2)	'35'	если вывод

E(w[,d]) - выборка поля шириной w с максимальной точностью для данных типа FLOAT, d - число цифр справа от десятичной точки. При выводе w должна быть на 7 больше чем d, потому что минимальное поле +n.ddddE+eee

Переменная	Формат	Результат	
4 пробела	E(4)	0	если ввод
2.9E7	E(5,3)	.29E+8	если ввод
345678	E(6,2)	.345678E+4	если ввод
0	E(11,3)	0.00E+000	если вывод
4.7E-10	E(11,3)	4.700E-010	если вывод
-30	E(15)	-3.000000E+001	если вывод

F(w[,d]) - выбирает арифметические значения с фиксированной запятой в поле

шириной w цифр, d-число цифр справа от десятичной точки. Пробелы игнорируются, если все пробелы - выдается 0. Если число не помещается в заданном размере - вместо первого символа печатается знак "*".

Переменная	Формат	Результат	
4 пробела и 0	F(5)	0	если ввод
-6	F(4)	-6	если ввод
13.09	F(5)	14	если ввод
0	F(5,1)	0.0	если вывод
-27	F(5,1)	-27.0	если вывод
.39	F(6,2)	0.39	если вывод

11.3.3. Форматы управления вводом/выводом

LINE(n) - переход к строке с номером n (только для PRINT), n должно быть больше 0, если текущий номер строки меньше чем n, программа выдает символы конца строк, пока номер строки не станет равен n. Если при этом исчерпана страница - ситуация ENDPAGE.

COLUMN(n) - переход к колонке с номером n, если больше текущего номера, то на следующую строку в заданную позицию, если больше размера строки, то на первую позицию; Для PUT EDIT записывает пробелы при перемещении к нужной позиции. Если текущая позиция больше чем n, то выводится символ конца строки, затем вставляются пробелы до нужной позиции на новой строке. Если n превосходит размер строки, то выводится метка конца строки и текущая позиция устанавливается в 1.

PAGE - переход на новую страницу (только для PRINT) т.е. программа выдает символ конца страницы (^L) и сбрасывает номер строки и колонки в 1.

SKIP[(n)] - выдача n концов строк или пропуск всех символов пока не встретится n концов строк. Этот оператор может стоять и внутри списка PUT.

X(n) - переход через n позиций в потоке данных или вставка n пробелов в PUT EDIT, символ конца строки (^J) не воспринимается.

TAB(n) - переход к ближайшей колонке с номером, кратным n*8.

11.3.4. Спецификация удаленного формата

R(format) - спецификация формата, заданного оператором FORMAT с меткой "format". Сам формат задается отдельным оператором с ключевым словом FORMAT.
Это удобно при частом выводе по одному и тому же формату. Если в PUT и GET стоит удаленный формат R, никаких других форматов (например, шаблона) уже быть не может. Например:

```
put edit(A,B,C)(R (M));
...
M: format (A (5),F (6,2),SKIP (3),A(2));
```

11.3.5. Шаблоны

Шаблоны - это специальные форматы для вывода чисел в виде с фиксированной точкой. Обычно их применяют при составлении статистических или финансовых документов. Общий вид шаблона: P'<спецификация>', где <спецификация> - это символьная строка из следующих символов:

$ + - S	плавающие символы знаков
* Z	условные символы цифр
9	безусловный символ цифр
V	позиция десятичной точки
/ , . : B	символы вставок
CR DB	символы дебета и кредита

Примеры шаблонов:

```
'BB$***,***V.99BB'
'$----,999V.99BCR'
'****'
```

Символы + - и S определяют знак числа по следующим правилам:

	S	+	-
Положительное число	+	+	Пробел
Отрицательное число	-	Пробел	-

- Как и знак $, эти знаки плавающие, т.е. на распечатке они будут стоять в одном месте перед первой значащей цифрой числа.
- Символы * и Z определяют, что печатать, если данная цифра числа 0. Печатается соответственно звездочка или пробел.

- Символы B / , . : - просто печатаются в указанном месте. B означает пробел.
- Символ 9 означает безусловную печать цифры (в том числе и 0).
- Символ V означает позицию десятичной точки в шаблоне. Число "подвинется" так, чтобы дробная часть начиналась с этой позиции, но точка по этому символу не печатается. Заметим, что этот символ не учитывается в длине шаблона.
- Символы дебета DB и кредита CR подобны знаковым символам. Эти символы применяются в бухгалтерских расчетах. Если число отрицательно, то печатается "дебет" (DB) или "кредит" (CR). Они могут стоять только в конце шаблона.

Ниже приведены примеры использования шаблонов, в первой колонке дается выводимое число, во второй - используемый шаблон, в третьей - результат на распечатке.

Число	Шаблон	Результат на печати
0.00	BB$*******V.99BB	$*******.00
0.01	BB$***,***V.99BB	$*******.01
0.25	BB$***,***V.99BB	$*******.25
1.50	BB$***,***V.99BB	$******1.50
12.34	BB$***,***V.99BB	$*****12.34
123.45	BB$***,***V.99BB	$****123.45
1234.56	BB$***,***V.99BB	$**1,234.56
12345.67	BB$***,***V.99BB	$*12,345.67
123456.78	BB$***,***V.99BB	$123,456.78
0.00	$$$$$B$$$V.99	$.00
0.01	$$$$$B$$$V.99	$.01
0.25	$$$$$B$$$V.99	$.25
1.50	$$$$$B$$$V.99	$1.50
12.34	$$$$$B$$$V.99	$12.34
123.45	$$$$$B$$$V.99	$123.45
1234.56	$$$$$B$$$V.99	$1 234.56
12345.67	$$$$$B$$$V.99	$12 345.67
123456.78	$$$$$B$$$V.99	$123 456.78
0.00	99/99/99	00/00/00
0.01	99/99/99	00/00/00
0.25	99/99/99	00/00/00
1.50	99/99/99	00/00/02
12.34	99/99/99	00/00/12

123.45	99/99/99	00/01/23
1234.56	99/99/99	00/12/35
12345.67	99/99/99	01/23/46
123456.78	99/99/99	12/34/57
0.00	**:**:**	********
0.01	**:**:**	********
0.25	**:**:**	********
1.50	**:**:**	*******2
12.34	**:**:**	******12
123.45	**:**:**	****1:23
1234.56	**:**:**	***12:35
12345.67	**:**:**	*1:23:46
123456.78	**:**:**	12:34:57
0.00	/++++,+++.V++/	
0.01	/++++,+++.V++/	/ +01/
0.25	/++++,+++.V++/	/ +25/
1.50	/++++,+++.V++/	/ +1.50/
12.34	/++++,+++.V++/	/ +12.34/
123.45	/++++,+++.V++/	/ +123.45/
1234.56	/++++,+++.V++/	/ +1,234.56/
12345.67	/++++,+++.V++/	/ +12,345.67/
123456.78	/++++,+++.V++/	/+123,456.78/
0.00	S***B***.V**	**********
-0.01	S***B***.V**	-********01
0.25	S***B***.V**	+********25
-1.50	S***B***.V**	-******1.50
12.34	S***B***.V**	+*****12.34
-123.45	S***B***.V**	-****123.45
1234.56	S***B***.V**	+**1 234.56
-12345.67	S***B***.V**	-*12 345.67
123456.78	S***B***.V**	+123 456.78
0.00	$SSSSBSSSV.SS	
-0.01	$SSSSBSSSV.SS	$ -.01
0.25	$SSSSBSSSV.SS	$ +.25
-1.50	$SSSSBSSSV.SS	$ -1.50
12.34	$SSSSBSSSV.SS	$ +12.34
-123.45	$SSSSBSSSV.SS	$ -123.45

1234.56	$SSSSBSSSV.SS	$ +1 234.56
-12345.67	$SSSSBSSSV.SS	$ -12 345.67
123456.78	$SSSSBSSSV.SS	$+123 456.78
0.00	***.***S	********
-0.01	***.***S	*******-
0.25	***.***S	*******+
-1.50	***.***S	******2-
12.34	***.***S	*****12+
-123.45	***.***S	****123-
1234.56	***.***S	**1.235+
-12345.67	***.***S	*12.346-
123456.78	***.***S	123.457+
0.00	$***,***V**CR	**********
-0.01	$***,***V**CR	$*******01CR
0.25	$***,***V**CR	$*******25
-1.50	$***,***V**CR	$******150CR
12.34	$***,***V**CR	$*****1234
-123.45	$***,***V**CR	$****12345CR
1234.56	$***,***V**CR	$**1,23456
-12345.67	$***,***V**CR	$*12,34567CR
123456.78	$***,***V**CR	$123,45678
0.00	/++++,+++.V++/	
-0.01	/++++,+++.V++/	/ 01/
0.25	/++++,+++.V++/	/ +25/
-1.50	/++++,+++.V++/	/ 1.50/
12.34	/++++,+++.V++/	/ +12.34/
-123.45	/++++,+++.V++/	/ 123.45/
1234.56	/++++,+++.V++/	/ +1,234.56/
-12345.67	/++++,+++.V++/	/ 12,345.67/
123456.78	/++++,+++.V++/	/+123,456.78/

11.4. Встроенная функция STRING

В операторах PUT и GET вместо указания файла, с которым производится обмен, можно указать специальную функцию STRING, имеющую один аргумент - строку типа CHARACTER или CHARACTER VARYING. В этом случае данные, преобразованные в текст, будут попадать в эту строку или наоборот, браться из нее. Например:

```
declare
s           char(80) var,
k           float,
(x,y,z)     char(2);
k=1234;
put string(s) list(k, k+1); // значение s ' 1.234000E+03  1.235000E+03'
s=date;                     // значение s '20.05.91'
get string(s) edit(x,y,z)(a(2),x(1)); // значения x='20', y='05', z='91'
```

В начале работы PUT переменная внутри STRING каждый раз обнуляется.

11.5. Циклы внутри операторов PUT и GET

Внутри списков переменных, перечисленных в операторах PUT и GET, могут быть указаны простые циклы, признаком которых является открывающая скобка.

Допустимы только циклы в форме DO-BY-TO. Пример:

```
DCL (X,Y)(10) FLOAT, I FIXED(31);

PUT SKIP LIST((X(I),Y(I) DO I=1 TO 10));
```

Это аналогично записи:

```
PUT SKIP:
DO I=1 TO 10; PUT  LIST(X(I),Y(I)); END I;
```

12. ВВОД/ВЫВОД ЗАПИСЕЙ (RECORD)

При этом способе обмена, преобразование данных в текст не производится. Различаются файлы двух типов:

- последовательного доступа (SEQUENTIAL) - записи обрабатываются в том порядке, как они расположены в файле;
- прямого доступа (DIRECT) - записи обрабатываются в произвольном порядке с использованием номеров записей (ключей) подобно обработке массива.

Данные пишутся или читаются из скалярных переменных, которые, в отличие от STREAM-обмена, уже не могут иметь тип CHARACTER VARYING. Ключ всегда должен иметь тип FIXED BINARY(31).

12.1. Оператор READ

Оператор читает из файла данные в переменную фиксированной длины и имеет вид: READ FILE (<имя>) INTO(x [,y]);

Если не было оператора OPEN, файл открывается как RECORD, SEQUENTIAL и INPUT. Количество считанных байт определяется длиной x, если явно не была задана длина y. Если в ENVIRONMENT была указана запись фиксированной длины, прочитается именно это число байт. Если x (или y) не соответствует длине записи, лишние байты будут отброшены (справа), недостающие - дополнятся нулями.

Возможна еще одна форма записи оператора:

READ FILE (<имя>) SET (p);

В этом случае, чтение производится во внутренний буфер файла и адрес этого буфера передается в указатель p. Используя буфер, можно не пересылать данные в переменные (а базировать переменные на p) и кроме этого, становится возможным использование оператора REWRITE.

12.2. Оператор READ-KEY

Оператор читает из файла данные в переменную фиксированной длины и имеет вид: READ FILE (<имя>) INTO(x) KEY (k);

Если не было оператора OPEN, файл открывается как RECORD, DIRECT,

INPUT и KEYED. Ключ k принимает значения от 0 до того значения, которое максимально может занимать файл с записями данного размера на диске. Все записи в файле считаются одинаковой длины. Будет считана запись с указанным номером.

12.3. Оператор READ-KEYTO

Оператор определяет значение ключа, которое затем можно использовать при прямом обмене.

```
READ FILE (<имя>) INTO(x) KEYTO (k);
```

Если не было оператора OPEN, файл открывается как RECORD, SEQUENTIAL, INPUT и KEYED. В переменную k возвращается значение, которое можно было бы использовать в дальнейшем как ключ записи, если затем открыть этот же файл как DIRECT KEYED.

12.4. Оператор WRITE

Оператор пишет в файл данные из переменной фиксированной длины и имеет вид:

```
WRITE FILE (<имя>) FROM(x [,y]);
```

Если не было оператора OPEN, файл открывается как RECORD, SEQUENTIAL и OUTPUT. Количество записываемых байт определяется длиной x, если не была явно задана длина y. Если в ENVIRONMENT была указана запись фиксированной длины, записывается именно это число байт. Если x (или y) не соответствует длине записи, лишние байты будут отброшены (справа), недостающие - дополнятся нулями.

12.5. Оператор REWRITE

Оператор пишет в файл данные из текущего внутреннего буфера обратно на диск, таким образом можно обновить последнюю считанную запись:

```
REWRITE FILE(<имя>);
```

В этом случае файл должен иметь атрибут UPDATE.

12.6. Оператор WRITE-KEYFROM

Оператор делает действие обратное оператору READ-KEY и имеет вид:

```
WRITE FILE (<имя>) FROM(x) KEYFROM (k);
```

Если не было оператора OPEN, файл открывается как RECORD, DIRECT, OUTPUT и KEYED. Ключ k принимает значения от 0 до того значения, которое максимально может занимать файл с записями данного размера на диске. Все записи в файле считаются одинаковой длины. Будет записана на диск запись с указанным номером.

В данной реализации вместо слова KEYFROM можно также писать слово KEY.

13. ВСТРОЕННЫЕ ФУНКЦИИ

Встроенные функции PL/1 - это стандартные подпрограммы языка, которые не требуют описания и включены в системную библиотеку языка PL1LIB.L86.

Вы можете назвать свою функцию как встроенную и в этом случае вместо встроенной будет вызываться Ваша. (Предполагается, что если Вы забыли, что есть такая встроенная функция, Вы и не используете ее). Для того чтобы воспользоваться встроенной функцией во внутреннем блоке (если во внешнем она описана как своя, а не встроенная), нужно описать ее с атрибутом BUILTIN.

PL/1 имеет следующие встроенные функции (примеры использования всех функций приведены в разделе 14):

- арифметические
- математические
- для работы со строками
- для преобразования типов
- для обработки прерываний
- общего назначения

13.1. Арифметические функции

Это следующие функции:

ABS	FLOOR	MOD	SIGN
CEIL	MAX	MULTIPLY	TRUNC
DIVIDE	MIN	ROUND	

Функции возвращают результаты арифметических действий и обычно используются при арифметических вычислениях.

13.2. Математические функции

Это следующие функции:

ACOS	ATAND	COSH	LOG10	SQRT
ACOSD	ATAND2	ERF	LOG2	TAN
ASIN	COS	ERFC	RANDOM	TAND
ASIND	COSD	EXP	SIN	TANH
ATAN	COSDSIN	GAMMA	SIND	
ATAN2	COSSIN	LOG	SINH	

Эти функции используются для математических вычислений с плавающей точкой и включают:

- наиболее распространенные тригонометрические функции
- логарифмические функции по основанию 2, 10 и e (натуральные)
- возведение числа «e» в степень
- гиперболические синус, косинус и тангенс
- функцию извлечения квадратного корня
- интеграл вероятности и гамма-функция
- датчик псевдослучайных чисел

Каждая из этих функций имеет единственный аргумент типа FLOAT BINARY и возвращает результат типа FLOAT BINARY. Только RANDOM возвращает FIXED BINARY. Вы можете задать аргумент другого типа, но компилятор автоматически преобразует его во FLOAT BINARY. Некоторые функции могут иметь аргументом комплексные числа.

Функции COSSIN и COSDSIND вычисляют одновременно синус и косинус аргумента и возвращают ответ как комплексное число (cos+i*sin).

Если аргумент математических и арифметических функций - число с обычной точностью (FLOAT BINARY (24)), то и результат - число с обычной точностью. Если же аргумент - число с двойной точностью (FLOAT BINARY(53)), то и результат - число с двойной точностью.

Все вычисления проводятся с помощью команд FPU x86-64 и обеспечивают точность (для FLOAT(53)) в 16 значащих цифр.

Для интегрирования, нахождения экстремума и т.д. нужно использовать специальные библиотеки математических функций (например, MATH.L86).

13.3. Функции для работы со строками

Это следующие функции:

BOOL	REPLACE	TRANSLATE
COLLATE	STRING	TRIM
INDEX	SUBSTR	VERIFY
LENGTH	TALLY	

Функции SUBSTR и INDEX работают как с символьными, так и с битовыми строками. Функция BOOL работает только с битовыми строками.

13.4. Функции для преобразования

Это следующие функции:

ASCII	CHARACTER	FLOAT
BINARY	DECIMAL	RANK
BIT	FIXED	UNSPEC

Эти функции преобразуют один тип данных в другой, они же используются для внутренних автоматических преобразований.

13.5. Функции для обработки прерываний

Это следующие функции:

ONCODE	ONFILE	ONKEY	ONLOC	ONTERM

Эти функции предоставляют дополнительную информацию при обработке прерывания нормального выполнения программы. Они не имеют параметров и возвращают значение только внутри ON-оператора. При этом они имеют смысл только при некоторых типах прерываний.

13.6. Функции общего назначения

Это следующие функции:

ADDR	GETDAY	LINENO
DATE	HBOUND	NULL
DATE4Y	LBOUND	NULL_PTR
DELAY	LDATE	PAGENO
DIMENSION	LDATE4Y	TIME

Эти функции дают информацию о базированных переменных, текущей дате и времени, о текущей строке и странице файла, о размерностях и границах индексов массива.

Функция DATE возвращает две последние цифры текущего года в текущей дате, DATE4Y – четыре цифры года.

Функции LDATE и LDATE4Y полностью аналогичны функциям DATE и DATE4Y, но берут дату из последнего вызова функции TIME, не обращаясь, лишний раз к Win API. Функция GETDAY возвращает день недели (1-понедельник).

14. СПИСОК КЛЮЧЕВЫХ СЛОВ, КОНСТРУКЦИЙ И ВСТРОЕННЫХ ФУНКЦИЙ PL/1

В данном разделе приведен список ключевых слов, конструкций языка и встроенных функций в алфавитном порядке, в заголовках указаны полные названия и возможные сокращения.

14.1. Обозначения

Далее используются без пояснений следующие условные обозначения:

- b1 b2 b3 произвольные битовые выражения (битовые строки)
- f имя файла
- n десятичная константа
- p целая часть
- q дробная часть
- s строка переменной длины
- s1 s2 s3 произвольные строковые выражения
- x x1 x2 x3 произвольные выражения
- v переменная языка

14.2. Ключевые слова, конструкции и встроенные функции

A или **A(n)** спецификация символьной строки в операторах PUT EDIT GET EDIT, а также атрибут оператора OPEN, например:
а) длина строки берется из переменной - PUT EDIT (C)(A);
б) длина строки явно задана - GET EDIT (C) (A(20));
лишние символы справа усекаются, недостающие - дополняются пробелами

ABS(x) выдает абсолютное значение выражения, например:
Y=ABS (5-Z/2);
ABS (-100) возвращает 100
ABS (18.78) возвращает 18.78

ACOS(x) встроенная функция арккосинуса, результат в радианах, например:
Y=ACOS (5-Z/2);
аргумент от -1 до 1
ACOS (0.886) возвращает 5.236490E-01

ACOSD(x) встроенная функция арккосинуса, результат в градусах, эквивалентен 180/PI*ACOS (X), например:
Y=ACOSD (5-Z/2);
аргумент от -1 до 1
ACOSD (0.886) возвращает 2.762517E+01

ADDR(v) функция возвращает адрес переменной v в памяти, например:
P=ADDR (A (I));

ALIGNED атрибут выравнивания на границу слова, например:
DCL G(0:5) BIT (2) ALIGNED; в данной реализации не имеет смысла и оставлен для совместимости при переносе программ

ALLOCATE оператор выделения памяти для базированных переменных, например: ALLOCATE B SET (P);

ALLOCATION(ADDR(v)) встроенная функция проверки наличия экземпляров переменной **v** класса CONTROLLED. Если эта функция возвращает ’0’b, значит в стеке уже нет ни одного экземпляра данной переменной, иначе можно использовать оператор FREE (v), чтобы получить очередной (предыдущий) экземпляр переменной **v**, который был ранее создан с помощью оператора ALLOCATE v; Например: IF ALLOCATION(ADDR(X) THEN...

ASCII(x) встроенная функция, возвращает символ с порядковым номером x из стандартной последовательности символов ASCII, например:
ASCII (88) возвращает 'X'

ASIN(x) встроенная функция арксинуса, результат в радианах, например:
Y=ASIN (5-Z/2);
аргумент от -1 до 1
ASIN (0.886) возвращает 1.047146E+00

ASIND(x) встроенная функция арксинуса, результат в градусах, эквивалентен 180/PI*ASIN (X), например:
Y=ASIND (5-Z/2);
аргумент от -1 до 1
ASIND (0.886) возвращает 6.237483E+01

ATAN(x) встроенная функция арктангенса, результат в радианах, например:
Y=ATAN (5-Z/2);
ATAN (0.577) возвращает 5.233360E-01

ATAN2(x,y) встроенная функция арктангенса, результат в радианах, эквивалентен ATAN (x/y), например:
Y=ATAN2 (5-Z/2,Z);
ATAN (0.886, 0.555) возвращает 1.0111830E+00

ATAND(x) встроенная функция арктангенса, результат в градусах, ATAND(X) эквивалентен 180/PI*ATAN (X), примеры:
Y=ATAND (5-Z/2);
ATAND (0.577) возвращает 2.998494E+01

ATAND2(x,y) встроенная функция арктангенса, результат в градусах, ATAND2 (X,Y) эквивалентен 180/PI*ATAN2 (X,Y), примеры:
Y=ATAND2 (5-Z/2,Z);
ATAND2(0.866,.0555) возвращает 5.793652E+01

AUTOMATIC или **AUTO** атрибут автоматического распределения памяти для переменных (исполняется только для рекурсивных процедур), например:
DCL A (10) FIXED AUTO;

B или **B(n)** или **B1 ... B4**
а) указатель битовой константы, например:
IF H='1010'B THEN ...;
б) B (n)-спецификация битовой строки при вводе-выводе, лишние биты справа усекаются, например: GET EDIT (G(I),H)(B(4),B(16));
в) B1 или B1(n) эквивалентно B или B(n)
г) B2 или B2(n) указатели 4-ричной системы счисления, т.е. строк, состоящих из символов 0,1,2,3; применяются так же, как B или B1;
д) B3 или B3(n) указатели 8-ричной системы счисления, т.е. строк, состоящих из символов 0,1,2,3,4,5,6,7; применяются так же, как B или B1;
е) B4 или B4(n) указатели 16-ричной системы счисления, т.е. строк, состоящих из символов 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F; применяются так же, как B или B1.

BASED атрибут, объявляющий о базировании переменных, например:
DCL B FIXED BASED; DCL C FLOAT BASED (P);

BEGIN ключевое слово начала блока, должно иметь парный END, например:
ON ERROR BEGIN; ... END;

BINARY или **BINARY(x)** или **BINARY(x,p)** или **BIN**
а) атрибут, объявляющий о представлении данных в памяти в двоичном виде, например: DCL X (0:5) FLOAT BINARY;
б) встроенная функция для перевода выражения к двоичному представлению, например: Y=BINARY (X+Z,13); второй аргумент задает размер целой части в разрядах, дробной части нет. Если X=12.675 FIXED DECIMAL (6,3), то
BINARY (X,15) возвращает 12, а
BINARY (12.675,15) возвращает 1.2000000E+01

BIT(n) или **BIT(x)** или **BIT(x,p)**
а) атрибут, объявляющий о представлении данных в битах $1 < n < 64$, например: DCL X BIT (15);
б) встроенная функция для перевода выражения в битовую строку длиной p, например: Y=BIT (X,13);
BIT (3,8) возвращает 00000110
BIT (-4,16) возвращает 0000100000000000

BOOL(b1,b2,b3) булевские функции над аргументами - битовыми строками b1 и b2, результат работы функции определяется программистом в аргументе b3:
если очередные биты b1=0 и b2=0, то результат - первый бит b3
если очередные биты b1=0 и b2=1, то результат - второй бит b3
если очередные биты b1=1 и b2=0, то результат - третий бит b3
если очередные биты b1=1 и b2=1, то результат - четвертый бит b3
например:
BOOL ('0011'B,'0101'B,'1001'B) возвращает '1001'B
BOOL ('01011'B,'11'B,'1001'B) возвращает '01100'B
более короткая строка из b1 и b2 дополняется нулями справа

BUILTIN если в блоке описаны собственные процедуры с именами, совпадающими с именами встроенных функций, то для того, чтобы во внутренних блоках отличать эти процедуры, встроенные функции описываются с атрибутом BUILTIN, например: DCL BIT BUILTIN;

BY параметр, определяющий шаг выполнения DO-оператора, например:
DO I=1 TO 10 BY 2;

CALL оператор вызова процедуры, в данной реализации необязателен, например:
CALL INVERT(A,N,M);

CEIL(x) встроенная функция, возвращающая наименьшее целое, большее или равное x, т.е. CEIL(X) эквивалентна -FLOOR (-X), например:
Y=CEIL (5-Z/2);
CEIL (7.9) возвращает 8
CEIL (5/3) возвращает 2

CHARACTER(x) или **CHARACTER(x,y)** или **CHAR**
а) спецификация строковых переменных: DCL X CHAR (100);
б) функция, преобразующая выражение в строку символов длиной y, лишние символы отрезаются справа, недостающие - заменяются пробелами
если x=-13.25, то
CHAR (X,10) возвращает ' -13.25'
CHAR (2*(3+7)-6,10) возвращает '   14'

CLOSE закрывает указанный файл. Место, занимаемое блоком управления файла (FCB), освобождается, например: CLOSE FILE (SYSPRINT); В данной реализации можно не писать ключевое слово FILE.

COLLATE() выдает строку из 128 символов ASCII в порядке возрастания, начиная с 0-го: C=COLLATE();

COLUMN(n) или **COL(n)** формат, определяющий номер колонки следующего символа в операторах GET и PUT, например: PUT EDIT (H)(COLUMN(10),B);

COMPLEX или **CPLX** атрибут при описании комплексных переменных, в данной реализации может быть только для данных типа FLOAT, например:
DCL A (1:10) FLOAT (53) COMPLEX;

CONTINUE оператор, прекращающий текущую итерацию цикла и начинающий новую, например: DO I=1 TO 100; IF I=50 THEN CONTINUE; ... END;

CONVERSION или **CONV** название исключительной ситуации при преобразовании данных. В данной реализации эквивалентно ERROR (1), таким образом, операторы ON CONVERSION BEGIN; … и ON ERROR(1) BEGIN; эквивалентны

COS(x) встроенная функция косинуса, аргумент - в радианах, например:
Y=COS (5-Z/2);
COS (3.1415/3.0) возвращает 5.000267E-01

COSD(x) встроенная функция косинуса, аргумент - в градусах, т.е. COSD(X) эквивалентен COS (X*PI/180), например:
Y=COSD (5-Z/2);
COSD (0.50) возвращает 9.999618E-01

COSDSIND(x) встроенная функция, одновременно выдающая косинус и синус в виде комплексного числа и работающая быстрее двух одиночных функций, аргумент - в градусах, т.е. COSDSIND(X) эквивалентен COSSIN (X*PI/180), например: DCL X FLOAT (53) CPLX; X=COSDSIND(Y);
COSDSIND (0.50e0) возвращает 9.99961923E-001+8.7265354E-003I

COSH(x) встроенная функция гиперболического косинуса, т.е. COSH(X) эквивалентен (EXP(X)+EXP(-X))/2, например:
Y=COSH (5-Z/2);
COSH (2.75) возвращает 7.853279E+00

COSSIN(x) встроенная функция, одновременно выдающая косинус и синус в виде комплексного числа и работающая быстрее двух одиночных функций, аргумент - в радианах, например: DCL X FLOAT (53) CPLX; X=COSSIND(Y);
COSSIN (0.50e0) возвращает 8.77582561890372E-001+4.79425538604203E-001I

CTL атрибут памяти класса CONTROLLED для переменных, позволяющий создавать несколько экземпляров переменной в стеке, а также массивы с динамически меняющимися границами, например:
DCL X (*,*) FLOAT (53) CTL;

DATA атрибут операторов GET и PUT, оставлен для совместимости при переносе программ. В PUT выдает имена переменных, использование в операторе GET эквивалентно использованию LIST, но позволяет записывать текстовые данные вместе с комментариями, т.е. при вводе игнорируются все символы до очередного '=', например: GET FILE (F) DATA(X,Y,Z);
при этом содержимое файла F: X=1, Y=0.25E-1, Z='СТРОКА', ... разделять данные можно только запятой, ";" (как в «полном» стандарте PL/1) недопустима.

DATE встроенная функция выдачи текущей даты, возвращает строку CHAR (8), например: S=DATE; S станет равным '13.07.21'

DATE4Y встроенная функция выдачи текущей даты, возвращает строку CHAR (10), например: S=DATE4Y; S станет равным '13.07.2021'

DECIMAL или **DECIMAL(x,p)** или **DECIMAL(x,p,q)** или **DEC**
а) атрибут, объявляющий о представлении данных в памяти в десятичном виде, например: DCL D DECIMAL (13,5);
б) встроенная функция, переводящая выражение в десятичный вид длиной p знаков и дробной частью q знаков (в тип FIXED DECIMAL), например:
D=DECIMAL (X+Y,15,4); p от 1 до 15, q от 0 до p
DECIMAL (125,6,2) возвращает 125.00

DECLARE или **DCL** объявляет о переменных и их атрибутах в программе, все переменные должны быть явно определены этим оператором, например:
DCL X FLOAT BIN;

DEFINED или **DEF** объявляет о том, что данная переменная разделяет общую память с переменной, указанной как параметр DEF, например:
DCL X FLOAT BIN;
DCL Y (4) BIT (8) DEF (X);

DELAY(x) заставляет PL/1-программу «заснуть» на x миллисекунд. Реализована с помощью функции SLEEP Win API, например, остановиться на секунду:
DELAY (1000);

DIMENSION(v[,n]) или **DIM** возвращает размерность массива v по индексу n или длину CHARACTER (при n=0), или размер элемента в байтах (при n<0) например: IF DIM(A,1)=10 THEN ...; Если второй параметр (номер размерности массива) не указан, принимается 1. Может также использоваться в операторах описания массивов, тогда размерность не обязательно должна идти первой:
DCL A (1:10) CHAR (*) VAR;
DCL B CHAR (*) VAR DIMENSION (1:10);

DIRECT один из параметров оператора OPEN для файлов типа RECORD, в том числе обрабатываемых по ключу, например:
OPEN (F) DIRECT KEYED ENV (F(1)B (1000));

DIVIDE(x1,x2,p) или **DIVIDE(x1,x2,p,q)** делит x1 на x2 с целой частью p и дробной частью q, т.е. с управляемой точностью (для FIXED BINARY q=0)
I=DIVIDE (K,L,7);
DIVIDE (296,49,15) возвращает 6
DIVIDE (233.456E2,1.19E1,24) возвращает 1.971092E+02
DO начало оператора цикла или операторных скобок, должен иметь парный END, например: DO I=X TO Y/2; ... END; После конца цикла можно указать для

наглядности переменную цикла или ключевые слова WHILE/REPEAT;

E(p) или **E(p,q)** формат чисел в представлении с порядком для операторов GET EDIT, PUT EDIT, например: PUT EDIT(Z)(E(20,4));

EDIT ключевое слово спецификации в операторах PUT и GET, например:
PUT EDIT (X,Y)(2(X(5),F (10,2));

ELSE ключевое слово альтернативной ветви в конструкции IF ... THEN ..., например: IF X=1 THEN X=Y; ELSE Y=1;

END общий указатель конца конструкций DO ..., BEGIN; ..., PROCEDURE ... После этого слова и перед точкой с запятой можно указать имя процедуры или переменную цикла для наглядности.

ENDFILE(f) ситуация, указывающая, что операторы GET или READ нашли конец файла или произошло переполнение диска при работе операторов PUT или WRITE, например: ON ENDFILE(F) CALL (P);

ENDPAGE(f) ситуация, указывающая, что последняя выведенная в файл f строка, превысила размер страницы для этого файла, например:
ON ENDPAGE (SYSPRINT) PUT SKIP;

ENTRY спецификация типа процедур и процедур-функций, например:
DCL X ENTRY RETURNS (FIXED);

ENVIRONMENT или **ENV** или **OPTIONS** один из параметров оператора OPEN, задающий размеры записей для файлов типа RECORD и размер буфера для работы с файлом, например: OPEN FILE (F) ENV(B (1024));

ERF(x) встроенная функция интеграла вероятности, результат - FLOAT BINARY, например: Y=ERF(Z/2);
ERF (1E0) возвращает 8.42700795123128E-001

ERFC (x) встроенная функция дополнительного интеграла вероятности (1-ERF), результат - FLOAT BINARY, например: Y=ERFC(Z/2);
ERFC (1E0) возвращает 1.57299204876871E-001

ERROR(n) или **ERROR** ситуация, указывающая на возникновение различных арифметических и системных ошибок в программе, например:

ON ERROR GO TO RETRY;

EXP(x) встроенная функция экспоненты, результат - FLOAT BINARY, например:
Y=EXP(5-Z/2);
EXP (5.13) возвращает 1.690170E+02

EXTERNAL или **EXT** спецификация, определяющая данные как внешние для этой процедуры, например: DCL X FIXED BINARY (7) EXTERNAL;

F(p) или **F(p,q)** или **F(x)**
а) спецификация десятичного числа в операторах PUT EDIT и GET EDIT, p - число знаков до точки, q - после, если q нет - точка не представляется, например:
PUT EDIT (K)(F (10,2));
б) параметр, определяющий длину записи файла, например:
OPEN FILE (F) ENV (F(100)) ...;

FILE(f) ключевое слово, определяющее имя файла в операторах: OPEN, GET, PUT, READ, WRITE и CLOSE, например: PUT FILE(F1) LIST ('Report'); В данной реализации это слово в некоторых операторах можно опускать.

FILEOPEN(f) встроенная функция, возвращает ’1’B, если указанный файл в данный момент открыт или ’0’B в противном случае, например:
IF ^FILEOPEN(F1) THEN OPEN FILE(F1)…

FIXED, FIXED(x), FIXED(x,p) или **FIXED(x,p,q)**
а) спецификация переменных типа BINARY и DECIMAL как данных с фиксированной точкой, например: DCL X FIXED BINARY;
б) функция, преобразующая выражение к типу FIXED, например:
K=FIXED (X+Y,7); если S='01010010'B, то
FIXED (S,8) возвращает 82
FIXED (S,24) возвращает 8.200000E+01

FIXEDOVERFLOW или **FOFL** ситуация, указывающая, что произошло переполнение переменной типа FIXED DECIMAL, например:
ON FOFL GO TO RETRY;

FLOAT или **FLOAT(x)** или **FLOAT(x,p)**
а) спецификация переменных типа BINARY как данных с плавающей точкой,

например: DCL X FLOAT BINARY;
б) функция, преобразующая выражение к такому виду, p - число разрядов, например: K=FLOAT (X+Y,15); если Y=4589 FIXED BINARY (15), то
FLOAT (Y,24) возвращает 4.589000E+03

FLOOR(x) возвращает наибольшее целое, не превышающее выражение x, например: K=FLOOR(5-Z/2);
FLOOR (7.9) возвращает 7
FLOOR ((5/3)) возвращает 1

FORMAT можно отдельно описать формат для операторов PUT EDIT, GET EDIT, пометить его и обращаться по этой метке к формату, например:
R1:FORMAT (SKIP (3),3(A (2),F (1)));
PUT EDIT (X,Y,Z,I,J,K)(R (R1));

FREE оператор освобождает память базированной переменной, выделенную раньше с помощью ALLOCATE, например: FREE B;

FROM(v) ключевое слово для вывода данных оператором WRITE, например:
WRITE FILE (F1) FROM (A);

GAMMA(x) встроенная функция гамма-интеграла, результат - FLOAT BINARY, например: Y=GAMMA(Z/2);
GAMMA (10E0) возвращает 3.62880000016692E+005

GET оператор ввода данных из файла в программу (из файла типа STREAM), например: GET LIST (K,G(I),B);

GETDAY встроенная функция, для текущей даты возвращает день недели (1 – понедельник, 7 – воскресенье), например: D=GETDAY;

GO TO или **GOTO** или **GO** оператор передачи управления в программе, например: GO TO RETRY;
в данной версии компилятора можно опускать слово "TO"

HBOUND(v[,n]) функция возвращает верхнюю границу индекса n в массиве v, например: K=HBOUND(A,1); Если второй параметр (номер размерности массива) не указан, принимается 1.

IF начало условного оператора, например: IF A=5 THEN CALL P;
IMPORT атрибут процедуры, которую должна динамически загрузить Windows из библиотек при выполнении программы.

INCLUDE оператор, позволяющий во время компиляции вставить в текст программы текст из другого файла, например: %INCLUDE 'c:\test\struc.pl1';

INDEX(s1, s2 [,s3]) функция возвращает номер позиции вхождения второй строки в первую, начиная с позиции s3, или 0, если вхождений нет, например:
IF INDEX (C,'?')=0 THEN ...;
INDEX ('0123456789','7') возвращает 8
INDEX ('0123456789','3',5) возвращает 8
Для символьных строк возможен третий параметр (s3), показывающий номер позиции в строке, начиная с которого нужно вести поиск. По умолчанию с 1. Если третий параметр отрицателен – поиск ведется с конца строки.

INITIAL или **INIT** ключевое слово начального присвоения значений переменным, требует атрибута STATIC, например:
DCL X FLOAT STATIC INIT (0);

INPUT параметр оператора OPEN, настраивающий файл на чтение операторами GET и READ, например: OPEN FILE (F1) INPUT;

INTERNAL или **INT** атрибут, определяющий данные, как описанные только в этом блоке, например: DCL X FLOAT INT; Принимается по умолчанию.

INTO(v) ключевое слово для ввода данных оператором READ, например:
READ FILE (F1) INTO (A);

KEY(f) или **KEY(x)**
а) ситуация, указывающая на неправильный ключ файла f в операторе ввода/вывода по ключу, например: ON KEY (F1) CALL P;
б) ключ, определяющий номер записи файла при его чтении оператором READ, файл имеет тип DIRECT, например: READ FILE (F1) KEY (K) INTO (A);

KEYED параметр оператора OPEN, определяющий, что в файле записи будут иметь ключи (записи должны быть фиксированной длины), например:
OPEN FILE (F1) KEYED;

KEYFROM(x) параметр оператора WRITE, записывающий переменную по ключу, ключ – обязательно FIXED BINARY (в этой реализации можно писать KEY), например: WRITE FILE (F1) FROM (A) KEYFROM (K);

KEYTO(v) параметр оператора READ возвращающий значение ключа, например: READ FILE (F1) INTO (A) KEYTO (K);

LABEL атрибут, описывающий переменную-метку, например:
DCL LAB(0:5) LABEL;

LBOUND(v [,n]) функция возвращает нижнюю границу индекса n в массиве v, например: K=LBOUND(A,1); Если второй параметр (номер размерности массива) не указан, принимается 1.

LDATE встроенная функция выдачи текущей даты, возвращает строку CHAR (8), аналогична встроенной DATE, но дату берет от последнего обращения к TIME, исключая возможную ошибку полуночи (время запросили до полуночи, а дату уже после) например: S=LDATE; S станет равным '13.07.21'

LDATE4Y встроенная функция выдачи текущей даты, возвращает строку CHAR (10), аналогична встроенной DATE4Y, но дату берет от последнего обращения к TIME, исключая возможную ошибку полуночи (время запросили до полуночи, а дату уже после) например: S=LDATE4Y; S станет равным '13.07.2021'

LEAVE оператор безусловного окончания цикла, внутри которого он расположен, например:
DO I=1 BY 1;
S=S+K; IF S>100 THEN LEAVE;
END;
Оператор LEAVE ON используется для выхода из ON-оператора без выполнения всех операторов внутри ON-оператора.

LENGTH(s) или **LENGTH(f)** возвращает текущую длину строки s или длину файла f (открытого с атрибутом INPUT или UPDATE), например:
K=LENGTH (STR); LENGTH ('') возвращает 0 LENGTH ('ABAB') возвращает 4

LIKE атрибут описания структуры, копирующий описание другой структуры, например: DCL 1 X (1:100) STATIC LIKE Y;

LINE(n) формат оператора PUT EDIT, переводит текущую строку на позицию n с начала страницы, например: PUT EDIT(K)(LINE(5),F (3));

LINENO(f) оператор возвращает номер текущей строки в файле f, например:
K=LINENO (SYSPRINT);

LINESIZE(x) параметр оператора OPEN, задающий размер строки в странице, 0-вой размер не определен, например: OPEN FILE (F1) LINESIZE(50);

LIST ключевое слово спецификации операторов PUT, GET, определяет ввод/вывод данных списком, (может быть опущен) например:
PUT LIST ((A (I) DO I=1 TO 4));
PUT ('ПРИМЕР');

LOG(x) встроенная функция натурального логарифма, результат имеет тип FLOAT BINARY, если x<0, возникает ситуация ERROR (3), например:
Y=LOG (5-Z/2);
LOG (10.0) возвращает 2.302585E+00

LOG10(x) встроенная функция логарифма по основанию 10, результат имеет тип FLOAT BINARY, если x<0, возникает ситуация ERROR (3), LOG10(X) эквивалентен LOG (X)/LOG(10):
Y=LOG10 (5-Z/2);
LOG10 (125.0) возвращает 2.096910E+00

LOG2(x) встроенная функция логарифма по основанию 2, результат имеет тип FLOAT BINARY, если x<0, возникает ситуация ERROR (3),
LOG2 (X) эквивалентен LOG (X)/LOG(2)
Y=LOG2 (5-Z/2);
LOG2 (10.0) возвращает 3.321927E+00

MAIN параметр, определяющий процедуру как главную, вызывающую все остальные, например: X:PROCEDURE OPTIONS (MAIN); или F:PROC MAIN;

MAX(x1,x2) функция возвращает наибольший из двух аргументов, например:
Y=MAX (X,Z);
MAX (234,64) возвращает 234

MIN(x1,x2) функция возвращает наименьший из двух аргументов, например:
Y=MIN (X,Z);
MIN (234,64) возвращает 64

MOD(x1,x2) функция возвращает остаток от деления x1 по модулю x2, т.е. если x2=0, то возвращает x1, иначе возвращает x1-(x2)*FLOOR (x1/x2), например:
Y=MOD (5-Z/2,Y+1);

MOD (7,3) возвращает 1
MOD (-7,3) возвращает 2
MOD (7,-3) возвращает -2
MOD (-7,-3) возвращает -1

MULTIPLY(x1,x2,p) или **MULTIPLY(x1,x2,p,q)** умножает x1 на x2 с целой частью p и дробной частью q, т.е. с управляемой точностью (для FIXED BINARY q=0)
I=MULTIPLY (K,L,7);
MULTIPLY (296,49,15) возвращает 14504
MULTIPLY (233.456E2,1.19E1,24) возвращает 2.778126E+05

NULL функция возвращает указатель конца списка данных при списковой организации структуры данных, например: IF P=NULL THEN ...; имеется также встроенная переменная NULL_PTR всегда имеющая значение NULL.

NULL_PTR встроенная переменная типа указатель, всегда указывает на NULL. Применяется, если формальный параметр процедуры указатель, а фактический – не требуется. Писать в таком случае NULL нельзя, так как это не указатель.

ON начало описания реакции на прерывание в программе, например:
ON OVERFLOW GO TO RETRY;

ONCODE() встроенная функция, возвращает код ошибки, установленный последним сигналом ERROR, например: K=ONCODE();

ONFILE() встроенная функция, возвращает имя файла, для которого последней сработала ситуация ENDFILE или ENDPAGE, например: F2=ONFILE();

ONKEY() встроенная функция, возвращает значение ключа, для которого последней сработала ситуация, связанная с ключом, например: K=ONKEY();

ONLOC() встроенная функция, возвращает имя последней вызванной процедуры, для которой сработала ситуация, связанная с вводом/выводом, например: S=ONKLOC();

ONTERM() встроенная функция, возвращает номер последнего считанного/записанного элемента из списка GET LIST и PUT LIST при котором сработала ситуация, связанная с вводом/выводом, например: K=ONTERM();

OPEN list начало оператора открытия файла, настраивающего программу на работу с файлом, после OPEN следует список параметров, например:
OPEN FILE (F1) PRINT ENV (A);

OPTIONS(list) ключевое слово в заголовке процедуры, задающее среду процедуры, например: X:PROC OPTIONS(MAIN,STACK (50000));

OUTPUT параметр оператора OPEN, настраивающий файл на работу с операторами PUT и WRITE, т.е. на запись, например:
OPEN FILE (F1) OUTPUT;

OVERFLOW или **OFL** ситуация, указывающая, что произошло переполнение переменной с плавающей точкой, например: ON OFL BEGIN; ... END;

P 's1' спецификация шаблонов в форматах операторов PUT EDIT, GET EDIT, например: PUT EDIT(X)(P'9999V.999DB');

PAGE параметр оператора PUT, начинающий вывод с новой страницы, например:
PUT PAGE LIST ('Стр.');
PUT EDIT (I)(PAGE,F (10));

PAGENO(f) возвращает номер текущей страницы файла f, например:
K=PAGENO (ФАЙЛ);

PAGESIZE(x) параметр оператора OPEN, определяющий размер страницы в строках, размер 0 задает бесконечную страницу, например:
OPEN FILE (F1) PRINT PAGESIZE (40);

PARAMETER необязательное обозначение класса памяти при описании формальных параметров процедуры. Оставлено для улучшения читаемости текста программ.

POINTER или **PTR** атрибут определяет переменную как указатель на адрес, например: DCL P POINTER;

PRINT параметр оператора OPEN, определяющий файл как выводимый на печать, например: OPEN FILE (F1) PRINT;

PROCEDURE или **PROC** ключевое слово начала процедуры, должно иметь парный END, например: P1: PROCEDURE(A) RETURNS (FIXED);

PUT начало оператора вывода данных в файл типа STREAM, например:
PUT FILE (F1) LIST (A (I),X);

R спецификация в операторах PUT EDIT и GET EDIT, указывающая, что вывод описан удаленным оператором FORMAT, например: PUT EDIT(I,J)(R(R1));

RANDOM(n) возвращает целое псевдослучайное число в диапазоне от 0 до n. Например:
K=RANDOM(100); возвращает псевдослучайные числа от 0 до 100. Имеется служебная переменная ?RANDOM FIXED(31) EXT, в которой хранится текущее состояние функции. В начале программы эта переменная содержит ноль.

RANK(s1) возвращает номер символа s1 в стандартом ряде ASCII. s1 описан обязательно как CHAR (1). RANK(X) эквивалентен INDEX (COLLATE (),X)-1. Например:
K=RANK ('c');
RANK ('Y') возвращает 89

READ оператор ввода данных из файла в память, например:
READ FILE (F1) INTO (A);

RECORD параметр оператора OPEN, настраивающий файл на работу с данными в двоичном виде, например:
OPEN FILE (F1) RECORD;

RECURSIVE атрибут процедуры, позволяющий обращаться ей к самой себе, например: X: PROC (N) RECURSIVE;

REPEAT(x) параметр оператора цикла, повторяющий цикл после вычисления выражения x и присвоения ее значения параметру цикла, например:
DO I=1 REPEAT (I+1);
REPEAT без параметра используется для бесконечного цикла:
DO REPEAT; эквивалентно DO WHILE (‘1’B);

REPLACE
а) оператор, присваивающий имена глобальным константам, например:
%REPLACE PI BY 3.1415926;
б) оператор замены фрагментов в текстовой строке, например:
s2=REPLACE (s1, ’елки’, ’палки’); заменит все найденные в строке s1 слова

«елки» на слова «палки» и присвоит новую строку-результат в s2.

RESIGNAL оператор отмены выполнения ON-оператора, который вызван да данную реакцию и возобновление поиска предыдущих установленных ON-операторов на данную реакцию, например:
ON ERROR BEGIN; IF F^=ONFILE THEN RESIGNAL; … END;

RETURN или **RETURN(x)** оператор возврата из процедуры, например:
RETURN (5-Z/2);

RETURNS(attribute) определяет спецификацию для процедур-функций, например: DCL X ENTRY RETURNS (BIT (4)); Это слово может быть опущено в заголовке процедуры, например вместо
P1:PROC (A,B) RETURNS (FLOAT);
можно писать просто
P1:PROC (A,B) FLOAT;

REVERT x отменяет ранее установленный ON-оператор для ситуации x, например: REVERT ENDFILE (SYSIN);

REWRITE записывает текущий буфер обратно в файл, таким образом можно обновить последнюю запись: REWRITE FILE (f);

ROUND(x,n) возвращает значение выражения x сдвинутое вправо (+n) или влево (-n) на n знаков от двоичной или десятичной точки, например:
Y=ROUND (Z+3.2,20);
ROUND (12345.24689,3) возвращает 12345.24700
ROUND (34567.12345,-3) возвращает 35000.00000
если аргумент битовая строка - производится циклический сдвиг вправо или влево (если второй аргумент отрицательная константа) на заданное число разрядов, например, перестановка «тетрад» в байте:
DCL X BIT (8); X=ROUND (X,4);

SEQUENTIAL параметр оператора OPEN, определяющий файл как последовательный, например: OPEN FILE (F1) SEQUENTIAL;

SET(v) ключевое слово в операторе ALLOCATE, определяющее в какой указатель вернуть адрес выделенной памяти для базированной переменной, например: ALLOCATE B SET(P);

SIGN(x) сигнатура выражения x (+1 0 -1), например:

IF SIGN (5-Z/2)=1 THEN ...;
SIGN (-76.45E4) возвращает -1

SIGNAL x оператор, искусственно вызывающий ситуацию x, например:
SIGNAL ENDPAGE (SYSPRINT);

SIN(x) встроенная функция синуса, аргумент в радианах, например:
Y=SIN (5-Z/2);
SIN (3.1415/6.0) возвращает 0.4999866E-01

SIND(x) встроенная функция синуса, аргумент в градусах, SIND(X) эквивалентен SIN (X*PI/180), например:
Y=SIND (5-Z/2);
SIND (0.50) возвращает 8.726534E-03

SINH(x) встроенная функция гиперболического синуса, SINH(X) эквивалентен (EXP (X)-EXP(-X))/2, например:
Y=SINH (5-Z/2);
SINH (2.75) возвращает 7.789352E+00

SKIP или **SKIP(n)** параметр операторов PUT и GET, переводит или пропускает строки, например:
PUT SKIP LIST (I);
PUT EDIT (G(I))(SKIP (3),B);

SQRT(x) квадратный корень из x, результат - FLOAT BINARY, если x<0, то возникает ситуация ERROR (3):
Y=SQRT (Z+3);
SQRT (2) возвращает 1.414213E+00

STACK(n) параметр конструкции OPTIONS, задающий нестандартный размер стека в байтах в заголовке EXE-файла, который попадает в поле SIZE_OF_STACK_RESERVE, например:
X: PROC OPTIONS(MAIN,STACK(10000));

STATIC атрибут статического выделения памяти. При использовании функции INITIAL всегда необходим. Например: DCL X FLOAT BIN STATIC;
STOP оператор, прекращающий выполнение программы и возвращающий управление операционной системе. Все открытые файлы закрываются. Может иметь код возврата типа FIXED BINARY. Например: STOP; или STOP(0);
В операторе CALL означает прекращение запущенной параллельно (оператором CALL STOP) процедуры, например:

```
CALL ОПРОС;              // обычный запуск процедуры ОПРОС
CALL ОПРОС TASK;         // параллельный запуск процедуры ОПРОС
…
CALL ОПРОС STOP;         // прекращение работы процедуры ОПРОС
```

STREAM параметр оператора OPEN, настраивающий файл на работу с символами ASCII, например: OPEN FILE (F1) STREAM;

STRING функция в операторах PUT и GET, позволяющая писать в строчную переменную или читать из нее, например:
PUT STRING (S) EDIT (X,Y,Z)(A,2 F (10,5));

STRINGRANGE название ситуации выхода индексов за границы строки во встроенной функции SUBSTR. Разрешена только при компиляции с ключем «T». В данной реализации эквивалентна ситуации ERROR (19).

SUBSCRIPTRANGE или **SUBRG** название ситуации выхода индексов за границы массива. Разрешена только при компиляции с ключем «T». В данной реализации эквивалентна ситуации ERROR (16).

SUBSTR(s,x1) или SUBSTR(s,x1,x2) функция выделяет подстроку из строки s, начиная с символа номер x1 и длиной x2, если x2 не указан, длина берется до конца строки, если используется как псевдопеременная - меняет строку, например:
IF SUBSTR (C,K+2,J+1)='#1' THEN SUBSTR(C,K+2)='New string';
SUBSTR ('01110101'B,4,2) возвращает '10'B.

SYSIN стандартное имя файла, вставляемое по умолчанию в оператор GET, это файл ввода, например: GET PAGE LIST (I);

SYSPRINT стандартное имя файла, вставляемое по умолчанию в оператор PUT, это файл вывода, например: PUT EDIT ('ОТВЕТ:',V)(A,F (3));

TAB(n) формат спецификации EDIT, задающий номер следующего столбца, кратного n*8, например: PUT EDIT(I)(TAB(3),F (4));

TALLY(s1, s2) функция возвращает число вхождения образца s2 в строке s1, т.е. применяет функцию INDEX(s1,s2) ко всем символам строки, начиная с первого, например:
IF TALLY(C,'?')<10 THEN ...;
TALLY ('xxxx','xx') возвращает 3

TALLY ('0x123456789x','x') возвращает 2

TAN(x) встроенная функция тангенса, аргумент в радианах, TAN(X) эквивалентен SIN (X)/COS (X), если COS(X)=0, то возникает ситуация ERROR (3):
Y=TAN (5-Z/2);
TAN (3.1415/4.0) возвращает 0.9999536E-01

TAND(x) встроенная функция тангенса, аргумент в градусах, TAND(X) эквивалентен TAN (X*PI/180), если COS (X*PI/180)=0, то возникает ситуация ERROR (3):
Y=TAND (5-Z/2);
TAND (0.50000) возвращает 0.8726867E-03

TANH(x) встроенная функция гиперболического тангенса, TANH(X) эквивалентен (EXP (X)-EXP(-X))/(EXP(X)+EXP(-X))
Y=TANH (5-Z/2);
TANH (2.75) возвращает 9.918597E-01

TASK атрибут оператора CALL, означающий создание нового потока и параллельный запуск указанной процедуры. Запущенную параллельно процедуру можно прекратить оператором CALL STOP, например:

```
CALL ОПРОС;              // обычный запуск процедуры ОПРОС
CALL ОПРОС TASK;         // параллельный запуск процедуры ОПРОС
…
CALL ОПРОС STOP;         // прекращение работы процедуры ОПРОС
```

THEN ключевое слово условного оператора, например:
IF A=B THEN CALL P1;

TIME функция выдачи текущего времени, возвращает строку CHAR (8), например: S=TIME(); в S будет передана строка '12:06:45'

TITLE(s1) параметр оператора OPEN, связывающий имя файла в программе с внешним наименованием файла, в том числе с командной строкой, например:

```
OPEN FILE (F1) TITLE ('c:\test\type.lib');
OPEN FILE (F1) TITLE ('$1.$1');
```

TO ключевое слово верхней границы параметра цикла, например:
DO I=1 TO 10;

TRANSLATE(s1,s2 [,s3]) возвращает строку, в которой символы из строки s1,

встреченные в строке s3, заменены на соответствующие им символы из строки s2, если s3 не указан, принимается COLLATE (), например:
C=TRANSLATE (C, 'YES', 'yes');

TRIM(s) встроенная функция, отбрасывающая в заданной символьной строке пробелы и нулевые байты, подряд идущие в начале и в конце строки. Возвращает результат как CHARACTER VARYING. Пример:
X=TRIM (X);
TRIM (’ 1 ’) возвращает ’1’

TRUNC(x) возвращает целую часть выражения x, т.е. если x<0, то возвращает CEIL (x), если x>0, то возвращает FLOOR (x), например:
Y=TRUNC (5-Z/2);
TRUNC (52.146) возвращает 52
TRUNC (-52.146) возвращает -52

UNALIGNED или **UNAL** атрибут, указывающий что данную переменную типа битовая строка может быть не выровнена на начало слова, например:
DCL G(0:5) BIT (2) UNAL; в данной реализации не имеет смысла и оставлен для совместимости при переносе программ

UNDEFINEDFILE или **UNDF** ситуация, указывающая, что файл не может быть открыт, например: ON UNDF BEGIN; ... END;

UNDERFLOW или **UFL** ситуация, указывающая, что порядок переменной с плавающей точкой слишком мал, например: ON UFL GO RETRY;

UNSPEC(v) выдает содержимое байта, слова или полуслова памяти или записывает значение аргумента в память, например:
UNSPEC (K)=UNSPEC(K)&'1'B;
если X=2500, то UNSPEC (X) возвращает '0110000110101000'B
UNSPEC, стоящий не в операторе присваивания, позволяет писать в программе произвольные коды, например: UNSPEC(‘CC’B4); устанавливает контрольную точку в данном месте программы командой INT 3.

UPDATE параметр оператора OPEN, позволяющий в дальнейшем и читать из файла и писать в файл, например: OPEN FILE (F1) UPDATE;
VALUE атрибут описания переменный, аналогичный описанию STATIC INIT, однако при компиляции проверяется, что эта переменная не меняется, иначе выдается предупреждение. Таким образом, можно создавать переменные, которые нельзя менять после начальной инициации, например:
DCL PI FLOAT (53) VALUE (3.1415926E0);

VARIABLE атрибут, определяющий данные типа ENTRY или FILE как переменные, например: DCL F FILE VARIABLE;

VARYING или **VAR** атрибут, определяющий данные как строки с переменной длиной, например: DCL C1 CHAR (20) VAR;

VERIFY(s1,s2) возвращает номер позиции символа в строке s1, который не совпал с соответствующим символом строки s2, или 0 в противном случае, например:
IF VERIFY (C,' ')=0 THEN ...;
VERIFY ('ABCDE','ABDE') возвращает 3
VERIFY ('ABC123','1A2B3C4D') возвращает 0
VERIFY ('','A') возвращает 0
VERIFY ('A','') возвращает 1

WAIT(b1) оператор ожидания события от параллельно запущенных процедур. Вызывающая этот оператор процедура переводится в режим ожидания возникновения, указанного как параметр события, например:
CALL ОПРОС TASK;
IF WAIT (A=B) THEN CALL ОПРОС STOP;

WHILE(b1) параметр оператора цикла, проверяющий возможность очередного повторения цикла, например: DO I=1 TO 10 WHILE(J=K);

WRITE оператор вывода данных из памяти в файл, например:
WRITE FILE (F1) FROM (A);

X(n) формат в спецификации EDIT, пропускающий или вставляющий n пробелов перед следующим символом, например:
PUT EDIT (I)(X (4),F (5));

ZERODIVIDE или **ZDIV** ситуация, указывающая, что переменная FLOAT BINARY или FIXED BINARY делится на ноль, например:
ON ZDIV GO TO RETRY;

15. РАСШИРЕНИЕ ТИПИЗАЦИИ «ФИЗИЧЕСКИМИ» ТИПАМИ

15.1. Понятие «физического» типа

При решении «инженерной» задачи на языках с развитой системой типов легко ввести «абстрактные» типы, например, «длина» и «ширина». Но при использовании таких типов, единственное, что при этом получается по умолчанию — запрет использования их в одной операции без специального указания на «преобразование» типов.

Однако по физическому смыслу «длина» и «ширина» — однотипные величины, и, если умножить «длину» на «ширину» должен получиться тип «площадь». А при делении «длины» на «ширину» должен получиться безразмерный коэффициент, который, как и обычные константы может быть использован в выражениях с любыми другими типами.

Поэтому для класса «инженерных» задач естественно расширить систему типов не «абстрактными» типами, а Международной системой единиц СИ.

15.2. Физический смысл типов в инженерных задачах

Приписывая при решении «инженерной» задачи объектам программы не абстрактный, а физический смысл (и используя как тип размерность, т.е. выражение производной величины через первичные), мы сразу добавляем хорошо определенное множество зависимостей для этих объектов, а значит, облегчаем компилятору автоматическую проверку корректности действий и, тем самым, повышаем надежность программы.

При этом теоретическое обоснование системы независимых «физических» типов уже давно имеется в виде успешно применяемой теории размерности, основная теорема («пи-теорема») которой формулируется следующим образом: *соотношение, не зависящее от выбора единиц измерения между (n+1) размерными величинами a, a_1, a_2,…a_n, k из которых имеют независимые размерности, может быть представлено в виде соотношения между (n+1-k) величинами П, $П_1$, $П_2$,… $П_{n-k}$, представляющими собой безразмерные комбинации (n+1) размерных величин.*

Основной смысл пи-теоремы состоит в том, что всякое физическое соотношение между размерными величинами можно сформулировать как соотношение между безразмерными величинами.

Следовательно, поскольку в СИ независимыми являются лишь семь физических величин:

- длина **L** (единица — метр),
- масса **M** (единица — килограмм),
- время **T** (единица — секунда),
- термодинамическая температура **Θ** (единица — Кельвин),
- сила электрического тока **I** (единица — ампер),
- сила света **J** (единица — кандела),
- количество вещества **N** (единица — моль),

то в программе любой переменной, которой мы приписали физический смысл, может быть сопоставлен «физический» же тип в виде размерности из степенного одночлена, выраженного через основные единицы: $[x]=L^l M^m T^t \Theta^\theta I^i J^j N^n$.

Таким образом, в программе переменным может быть назначено произвольное число видов «физического» типа, каждый из которых представляется при трансляции одинаково — степенным одночленом, имеющим возможно разные показатели степеней, в том числе нулевые. Если все показатели степеней нулевые — объект не имеет размерности.

15.3. Масштабный коэффициент в «физических» типах

На практике величины, имеющие физический смысл, часто задаются в производных единицах измерения, отличающихся от базовых единиц масштабным коэффициентом. Например, длины — в км, площади — в Га и т.д. Поэтому целесообразно ввести в представление «физических» типов ещё и масштабный коэффициент в виде одного числа в формате IEEE-754.

Наличие такого коэффициента позволит компилятору автоматически учитывать масштабы при генерировании операций для выражений. Например, если переменным X1, X2, X3 приписаны типы соответственно [км], [см] и [мм], то при вычислении выражения X1=X2+X3, компилятор должен автоматически добавить операции умножения X2 на 0.01 и X3 на 0.001, а затем результат сложения перед присваиванием X1 ещё разделить на 1000.

Наличие приближенного масштабного коэффициента дает важное следствие — «физические» типы могут быть тоже только у приближенных переменных, т.е. только у переменных в формате FLOAT. Однако для класса «инженерных» задач такое ограничение на переменные практически несущественно.

При этом автоматическая подстановка масштабных коэффициентов при трансляции выражений означает, что все расчеты в программе будут производиться в базовых величинах СИ, как в приведенном выше примере — в метрах. Это также означает, что масштабные коэффициенты в «физическом» типе могут быть только у переменных, но не у выражений, где масштабы уже

учтены при генерации операций загрузки переменных этого выражения.

15.4. Дополнительные величины в «физических» типах

Хотя в СИ имеется лишь семь базовых величин, на практике используются ещё две вспомогательные величины — плоский угол (единица — радиан) и телесный угол (единица — стерадиан).

Здесь наблюдается некоторое противоречие с точки зрения формального определения типа. В самом деле, радиан и стерадиан — по определению безразмерные величины и могут быть использованы или не использованы в выражениях для других производных единиц СИ. Однако в некоторых случаях они ведут себя как обычные (размерные) величины. Например, имеется встроенная функция sin, входным параметром которой должно быть выражение в радианах и функция sind, входным параметром которой должно быть выражение в градусах. Переменная с «физическим» типом «градус» отличается от переменной с «физическим» типом «радиан» только масштабным коэффициентом π/180.

Таким образом, целесообразно и радианы и стерадианы также включать в степенной одночлен представления «физического» типа и, например, проверять, что функция sin берется от величины в радианах или в градусах, а не в метрах. Но тогда формально возникает конфликт размерностей при других вычислениях, например:

```
W=φ/t;    // угловая скорость (угол делится на время)
V=R*W; // линейная скорость (радиус поворота умножается
          // на угловую скорость)
```

Если угол φ имеет «физический» тип «радиан», то получается, что угловая скорость W имеет «физический» тип «радиан в секунду», а линейная скорость после умножения на радиус — «метр на радиан в секунду», что не соответствует требуемой размерности скорости «метр в секунду».

Поэтому хотя вспомогательные величины радиан и стерадиан действительно также представлены в степенном одночлене «физического» типа, но ведут себя не совсем так, как семь остальных базовых величин. Они считаются «размерными» величинами, только если остальные величины в одночлене имеют нулевые показатели степени (т.е. отсутствуют).

Как только при вычислениях появляется любая другая величина — радиан и стерадиан «сокращаются», т.е. их показатель степени транслятором обнуляется. Тогда в приведенном примере выражение φ/t (и угловая скорость W) получает «физический» тип «единица, деленная на секунду», а выражение R*W правильный «физический» тип скорости «метр в секунду». Но при этом выражение, например, sin(2*φ) по-прежнему можно проверить на корректный «физический» тип аргумента «радиан».

15.5. Внутреннее представление «физического» типа

Исходя из приведенных выше теоретических и практических соображений, каждой переменной программы, которой может быть приписан некоторый физический смысл, программистом может быть присвоен универсальный «физический» тип из величин СИ. Данный тип в таблице компилятора представлен в виде одного масштабного коэффициента в формате IEEE-754 и занимающего восемь байт, а также девяти показателей степени основных и вспомогательных величин в смысле единиц СИ

Поскольку вычисления в «инженерных» задачах проводятся и с дробными степенями, например, `Z=SQRT (X**2+Y**2)`, то показатели степени в универсальном «физическом» типе представлены рациональными (и возможно неправильными) дробями. При этом байт занимает числитель и байт знаменатель. Таким образом, показатели степени в трансляторе могут быть представлены числами от -128 до 127, а в целом каждый «физический» тип переменной занимает в таблице транслятора 8+9*2=26 байт.

15.6. Действия с «физическими» типами при вычислении выражений

При анализе выражений в программе и генерации операций над переменными с «физическими» типами компилятор производит очевидные действия, чтобы получить текущий «физический» тип всего выражения.

- При умножении переменных показатели степеней их «физических» типов складываются. Поскольку степени представлены дробями, то и складываются они как дроби.
- При делении переменных показатели степеней их «физических» типов вычитаются.
- При возведении переменной в степень-константу показатели степеней «физического» типа умножаются на эту константу.

При этом возведение в нецелую степень или в степень-переменную для «физического» типа считается ошибкой, за исключением случая представления степени-константы в виде дроби. Например, в выражении `X**(1e0/3e0)` показатели степени «физического» типа переменной `X` будут умножены на дробь 1/3, т.е. в данном случае знаменатели степеней будут умножены на три.

После каждого из перечисленных выше действий компилятором проводится попытка приведения дробей показателей степеней к правильному виду. И для удобства работы проверяется частный случай сокращения всех базовых величин, т.е. случая получения всех нулевых числителей показателей степеней. Тогда и весь «физический» тип просто обнуляется, выражение «теряет»

тип и становится «чисто» безразмерным, что упрощает и ускоряет последующие операции с типами.

При сложении, вычитании, сравнении и присваивании переменных с «физическими» типами, сами типы не меняются, а просто сравниваются на совпадение. При этом масштабные коэффициенты учитываются в выражениях и присваиваниях и поэтому в этих операциях могут не совпадать. Однако есть частные случаи присваивания или сравнения (например, векторов), когда присваивание или сравнение реализуется просто побайтным переписыванием или сравнением двух участков памяти. В таких случаях и масштабные коэффициенты «физических» типов у сравниваемых или присваиваемых переменных обязаны также совпадать.

15.7. Задание «физических» типов в исходном тексте программы

Для удобства задания «физических» типов с сохранением возможности трансляции уже написанных программ (без таких типов), были использованы символы квадратных скобок, ранее вообще не применявшиеся в языке PL/1, например:

```
declare V float(53) [м/с];
```

Внутри квадратных скобок могут быть записаны произвольные выражения, состоящие из скобок, знаков умножения, деления, возведения в степень, числовых констант и девяти имен базовых и вспомогательных величин СИ: **м, кг, с, а, к, моль, кд, рад, ср**. Для угловых величин допустимо также имя «ПИ». Например:

```
declare Mu float(53) [(1000*m)**3/c**2];
declare Fi float(53) [пи/180*рад];
```

Встроенная таблица «физических» типов в трансляторе имеет лишь девять первых заполненных строк перечисленными выше базовыми величинами, однако с помощью препроцессорного оператора замены %replace в неё можно добавлять произвольное число «нестандартных» типов, например:

```
%Replace
[км]      by [1000*м],
[час]     by [3600*с],
[миля]    by [1852*м],
[узел]    by [миля/час];
```

```
…
Declare
скорость          float(53) [км/час],
скорость_судна float(53) [узел];
```

При этом после выполнения оператора %Replace имена «км», «час», «миля» и «узел» становятся допустимыми только внутри квадратных скобок описания «физического» типа, в остальном тексте программы эти имена можно использовать для других целей.

15.8. Присваивание переменным начальных значений

При обработке оператора присваивания переменной с «физическим» типом транслятор проверяет, что показатели степени девяти величин СИ в представлении типа в левой и правой части присваивания совпадают. Но как быть при присваивании переменной начального значения не выражения, а, например, константы?

Но поскольку числовые константы, записываемые непосредственно в физических уравнениях не должны иметь собственной размерности, инициализация значений выполняется так:

```
Declare
//---- константы ----
g   float(53) value(9.81e0) [м/с**2],
v0  float(53) value(10e0)   [км/час],
//---- переменные ----
m  float(53) [кг],
v   float(53) [км/час],
F   float(53) [кг*м/с**2];
…
v=v0; F=g*m;
```

Исключение составляет константа 0. Её «физический» тип может быть любым:

```
v=0; F,m=0;
```

15.9. Чтение и запись переменных с «физическими» типами

Двоичный обмен операторами read/write осуществляется без каких-либо преобразований и поэтому переменные с «физическими» типами записываются и читаются из файлов точно так же, как и любые другие переменные.

А вот в другом случае появляется несколько непривычное свойство вывода переменных с «физическими» типами. Поскольку в общем случае текстового вывода выдаваемый объект — это выражение, то для переменных с «физическими» типами в нем автоматически будут учтены масштабные коэффициенты, а, следовательно, выражение всегда будет выведено в базовых единицах СИ, например, в метрах, даже когда выводится одна переменная и она задана в километрах или в миллиметрах.

Для того чтобы операторы put и get оставались «зеркальными» транслятор автоматически учитывает масштабный коэффициент переменной и при чтении её оператором get, т.е. при чтении также подразумевается, что значения хранятся в файлах в базовых единицах СИ.

Если все же необходим текстовый обмен без учета масштабных коэффициентов, то это можно реализовать, используя т.н. «наложенные» переменные, т.е. описав переменные без «физических» типов по тем же адресам памяти, что и переменные с «физическими» типами, например, в случае:

```
declare
X1 float(53) static init(10e0) [км],
X2 float(53) defined(X1);

put skip list(X1, X2);
```

будут выданы значения 10000 и 10.

15.10.Встроенная переменная ?TYPE для выражений

При добавлении в транслятор «физических» типов традиционно следовало бы добавить и встроенную функцию, например, с именем TYPE, возвращающую «физический» тип своего аргумента. Однако вместо этого введена встроенная переменная (строка) ?TYPE, автоматически принимающую значение «физического» типа (т.е. размерности) последнего вычисленного выражения.

Ведь вычислять «физический» тип для одной переменной с помощью встроенной функции довольно бессмысленно, поскольку эта же переменная описана с этим же «физическим» типом здесь же, в исходном тексте. «Физический» тип обычно нужен при выводе результатов в виде выражений, но тогда выражение транслятору придется вычислять два раза, например:

```
put skip list(s/t,type(s/t));
```

Поэтому для вывода удобнее использовать встроенную переменную, а не функцию, например, при выполнении оператора вывода в данном примере:

```
declare
s  float(53) static init(10) [км],
t  float(53) static init(5)  [час];
…
put skip list(s, ?type,t, ?type, s/t, ?type);
```

будут выданы следующие значения:

```
     1.00000000000000E+004 М  1.80000000000000E+004 С
5.55555555555555E-001 М*С**-1
     конец программы
```

15.11.«Физические» типы формальных параметров подпрограмм

Формальные параметры подпрограмм, как и обычные переменные, естественно, также могут иметь «физический» тип с масштабным коэффициентом. При вычислении и подстановке фактического параметра транслятор выполняет те же действия, что и при обычном присваивании, т.е. сравнивает показатели степеней базовых величин и умножает на масштабный коэффициент.

Однако если параметром является не скалярная величина, а сразу агрегат данных, масштабный коэффициент формального и фактического параметров должны строго совпадать и в этом случае на коэффициент ничего не умножается.

Также «физический» тип может иметь возвращаемое подпрограммой значение:

```
declare
//---- модуль радиус-вектора ----
Vmod entry((3) float(53)[км]) returns(float(53)[км]);
```

Поскольку на практике используются и подпрограммы с полиморфными типами, допускается задание в формальных параметрах неопределенного «физического» типа (с помощью знака «?»), при котором проверки на совпадение и учет масштабных коэффициентов отключаются.

```
declare
//---- модуль любого вектора ----
Vmod0 entry((3) float(53)[?]) returns(float(53)[?]);
```

Еще один особый случай представляет встроенная функция квадратного корня SQRT. Она тоже имеет неопределенный «физический» тип формального аргумента, но возвращает всегда такой же тип как у фактического аргумента и в

степени ½.

Это правило нельзя распространить на любые подпрограммы с неопределенными типами, поскольку в общем случае они могут возвращать «физические» типы, не обязательно совпадающие с типом входного аргумента.

15.12.Отладочные средства для «физических» типов

Кроме встроенной переменной ?TYPE, есть ещё два отладочных средства. Это возможность вывести всю таблицу переменных программы, включая теперь и их «физические» типы с масштабными коэффициентами и собственно сообщения при компиляции о несоответствии типов в вычисляемых выражениях. В данном случае в небольшом тесте, приведенном ниже, в операторе сравнения «физические» типы справа и слева оказались не равны.

```
test:proc main;

%replace
[км]  BY [1000*м],
[час] BY [3600*с];

declare

s  float(53) static init(10) [км],
t  float(53) static init(5)  [час],
v  float(53) [км/час];

if s*t>v*t then stop;
…
```

Вывод таблицы переменных с «физическими» типами и сообщения о несоответствии «физических» типов в операции сравнения:

```
a 00000000 TEST ПРОЦЕДУРА ПАРАМЕТРЫ(0) ОБЩЕЕ НЕИЗМЕНЯЕМОЕ
БЛОК В СТРОКЕ 3, ВЫДЕЛЕНО ПАМЯТИ 8 БАЙТ
c 00000000 S ВЕЩ. ЧИСЛО ДВОИЧНОЕ (53) СО ЗНАЧЕНИЕМ ПОСТОЯННОЕ [1.0000000E+03*М]
c 00000000 T ВЕЩ. ЧИСЛО ДВОИЧНОЕ (53) СО ЗНАЧЕНИЕМ ПОСТОЯННОЕ [3.6000000E+03*С]
c 00000000 V ВЕЩ. ЧИСЛО ДВОИЧНОЕ (53) ВРЕМЕННОЕ [2.7777777E-01*М*С**-1]
*КОНЕЦ*
  13 c      if s*t>v*t then stop;
РЗМ. НЕ=                ?
М*С # М
```

15.13.Пример применения «физических» типов

Для примера приведены фрагменты реальной программы. Исходная программа не содержала «физических» типов.

а) Потребовалось ввести ряд переменных с типами вместо ранее безразмерных констант, например:

```
// СКОРОСТЬ ВРАЩЕНИЯ ЗЕМЛИ
WEarth  float(53) static init(0.000072921158e0)      [РАД/С],

// ДВА ПИ
Dpi     float(53) static init(6.2831853071795864e0) [РАД],

// Epsilon/Mu
Em      float(53) static init(66072.1866e0)              [KM**2],

// ГРАВИТАЦИОННЫЙ ПОТЕНЦИАЛ
Mu      float(53) static init(398600.4e0)                [KM**3/C**2],

// ПОЛУОСЬ ДЛЯ СРЕДНЕЙ ВЫСОТЫ 358 КМ
Am      float(53) static init(6736e0)                    [KM],

//ЭКВАТОРИАЛЬНЫЙ РАДИУС ЗЕМЛИ
Re      float(53) static init(6378.137e0)                [KM],
```

б) Потребовалось убрать ряд приведений масштабов переменных, например, было:

```
//---- ПРИВЕДЕНИЕ МАСШТАБОВ К КМ И +-180 ----

do i=1 to 3;
  vsg(i)=vsg0(i)/1000e0;       // ПЕРЕВЕЛИ В КМ
  vrg(i)=vrg0(i)/1000e0;       // ПЕРЕВЕЛИ В КМ
  if Lamseans(i) > 180e0 then Lamseans(i)-=360e0;
end i;
```

Стало:

```
//---- ПРИВЕДЕНИЕ МАСШТАБОВ К КМ И +-180 ----

do i=1 to 3;
  vsg(i)=vsg0(i);     // АВТОМАТИЧЕСКИЙ ПЕРЕВОД В КМ
  vrg(i)=vrg0(i);     // АВТОМАТИЧЕСКИЙ ПЕРЕВОД В КМ
  if Lamseans(i) > У_180 then Lamseans(i)-=У_360;
end i;
```

Было:

```
BetaBal=60e0*(per2-per1)/(te2s-te1s)/2e0;   // ПЕРИОД БЫЛ В МИНУТАХ
```

Стало:

```
BetaBal=(per2-per1)/(te2s-te1s)/2e0;          // ПЕРИОД БЫЛ В МИНУТАХ
```

в) Некоторые произвольно разбитые на части формулы потребовалось свернуть, чтобы не вводить промежуточных переменных с нестандартной размерностью, например, было:

```
//---- ТЕКУЩИЙ РАДИУС ДЛЯ РАСЧЕТА ВЫСОТЫ ----

Radius  = J3-et1+dz1*dz1+J4*cos(ArgLat+ArgLat);
Radius *= Axe*(1e0+J7*tp);
```

Стало:

```
//---- ТЕКУЩИЙ РАДИУС ДЛЯ РАСЧЕТА ВЫСОТЫ ----

Radius  = Axe*(1e0+J7*tp)* (J3-et1+dz1*dz1+J4*cos(ArgLat+ArgLat));
```

Однако большей частью исходный текст остался без изменений. В результате тестовых расчетов были получены те же результаты, что и у исходной программы, при компиляции сообщений о несоответствии типов не выдавалось, например, самая громоздкая из использованных формул дала корректную размерность длины:

```
//---- ОЦЕНКА ПОЛУОСИ ПО ЗНАЧЕНИЮ ПЕРИОДА ----

do AxeOfPer=Am, i=1 to 4;                  // НАЧИНАЕМ С ВЫСОТЫ 358 КМ
 AxeOfPer=((Period/Dpi)**2*Mu)**(1e0/3e0)
      /(1e0-2e0/3e0*Em/AxeOfPer/AxeOfPer*(4e0*CosIncl*CosIncl-1e0));
end;
```

Однако одна логическая ловушка с точки зрения размерностей в программе все-таки нашлась:

```
//---- ШИРОТА В ГРАДУСАХ С УЧЕТОМ ЭЛЛИПСОИДНОСТИ ЗЕМЛИ ----

Fi = Fi + Alz*sin(2e0*Fi);
```

Здесь Alz — безразмерный коэффициент сжатия земного эллипсоида, равный 1/298.257. Получается, что в формуле угол Fi в радианах складывается с заведомо безразмерным синусом, т.е. несоответствие размерности. Так сказалось формальное представление углов условно размерной величиной «радиан», несмотря на то, что реальный радиан величина безразмерная. Для формального

же преодоления этого несоответствия в формуле пришлось дополнительно умножить синус на переменную с «физическим» типом «радиан» и единичным значением.

15.14.Преимущества использования расширения типов

Дополнительной типизация и назначение нового свойства переменным, определенного как «физические» типы в виде базовых и вспомогательных единиц Международной системы СИ позволило:

- автоматически проверять по размерности корректность записи уравнений, имеющих физический смысл;
- автоматически учитывать различные несистемные единицы измерений, при этом организовать сами вычисления всегда в семи базовых и двух вспомогательных единицах СИ.

Если внутреннее представление «абстрактного» типа — целое число, то с таким типом можно выполнять лишь простейшие действия, например, сравнивать на совпадение или определять предыдущий/следующий тип. Обычно автоматически при вычислениях такой тип не меняется.

В отличие от «абстрактных» типов, которых может быть бесконечное число, и отношения, между которыми должен задавать программист, число «физических» типов мало, и они независимы друг от друга по определению.

Кроме этого «физический» тип изначально хранит гораздо больше информации об объекте и автоматически меняется при вычислении выражений по вполне очевидным правилам. Масштабный коэффициент «физического» типа позволяет легко оперировать внесистемными единицами измерений и при этом автоматически сводить все вычисления к системным единицам.

Несмотря на то, что некоторые действия в программе из-за новых типов упрощаются, а некоторые встроенные функции (например, упомянутая выше sind) стали ненужными, в целом число операций в вычислениях может увеличиться (из-за добавления умножений на масштабные коэффициенты). Однако при этом проверки корректности программы облегчаются и её надежность возрастает.

Как пример важности таких проверок корректности можно привести неудачу миссии Mars Climate Orbiter 23 сентября 1999 года из-за расчетов в «не тех» единицах измерения. Вполне вероятно, что, если бы тогда используемые трансляторы имели бы механизм «физических» типов, подобный приведенному выше, а также автоматический вывод в системных единицах, можно было бы выполнять расчеты в злополучных «фунт-силах», но при этом избежать последующих разночтений с печальными последствиями.

16. ДОСТУП ИЗ PL/1 ПРОГРАММ К РЕГИСТРАМ ПРОЦЕССОРА

В данной реализации возможно прямое обращение (запись/чтение) к регистрам общего назначения процессора x86-64. Эта возможность делает программы привязанными к конкретному типу процессоров, однако иногда это единственный способ выполнить какое-то действие в программе, обычно связанное с внешней аппаратурой.

Совместно с обращением к функции UNSPEC, это расширение позволяет программировать прямо в машинных командах.

Для обращения к регистрам в программе необходимо описать требуемые регистры как переменные типа BIT или FIXED BINARY. Признаком регистра, а не обычной переменной является знак «?» перед именем регистра. Пример:

```
DCL
?AL FIXED (7),
?AH BIT (8);

?AL=12;
?AH='12'B4;
```

Следующий пример показывает возможность чтения из порта 378 в переменную X:

```
DCL
?DX     BIT (16),
?AL     FIXED (7),
X       FIXED (7);

?DX='0378'B4;          // ЗАНОСИМ В РЕГИСТР DX ЗНАЧЕНИЕ 378
UNSPEC ('EC'B4);       // КОМАНДА IN AL,DX
X=?AL;                 // ЗАПИСЫВАЕМ ИЗ AL В ПЕРЕМЕННУЮ X
```

17. ВЫЗОВ И РАБОТА КОМПИЛЯТОРА

Компилятор расположен в файле PLINK64.EXE. Вызов осуществляется директивой:

PLINK64 <имя>.PL1 [<параметры>] или
PLINK64 <имя>.PLI [<параметры>] или
PLINK64 * <имя> [<параметры>]

Если указан значок “*”, расширение файла с иходным текстом может быть любым, а не только .PL1 или .PLI.

<имя> - имя файла с исходным текстом. Можно указать путь для исходного файла, однако файл .OBJ запишется всегда в текущую папку;

<параметры> - необязательный список параметров, представляющий собой перечисление латинских букв без пробелов, например:

```
plink64 TEST.PL1 lf
```

В параметры могут также входить ключи отладочной печати (см. ниже) в виде цифры от 0 до 9.

Установленные ключи компиляции доступны в самой PL/1-программе при выполнении через встроенную переменную DCL ?КЛЮЧИ EXT BIT (32);

17.1. Параметры компиляции

Параметры могут быть заданы как в командной строке, так и установлены с помощью системного реестра Windows. В последнем случае, они сохраняются и действуют при последующих вызовах до новой установки в реестре. Однако, если задать параметр в реестре, а затем указать его же при вызове, параметр при работе будет отменен (инвертирован).

В реестре в разделе «HKEY_CURRENT_CONFIG\SOFTWARE» можно создать имя раздела «PL/1», а в нем подразделы с названиями «A» - «Z» типа REG_DWORD и присваивать им значения 0 или 1. Подраздел с именем «%» и типом REG_SZ может содержать значение в виде одной буквы, которая заменит знак «?» в имени файла в операторе %INCLUDE. Подраздел «ERR» может содержать текст, добавляемый к каждому сообщению об ошибке.

Параметры можно указать и в тексте самой программы буквой, перед которой стоят символы: "%_", например: %_f Однако, не все параметры можно задавать внутри программы, так, нельзя задать ключ D, поскольку файл .PRN не создан (он создается при запуске компилятора, а не при чтении программы).

Параметры можно отменить в теле программы командой %_-<буква>. Например, %_-K отменяет выдачу INCLUDE на экран. Параметры N, U и K можно задавать и отменять несколько раз. Конструкция:

```
%_u /* пример текста */
%_-u
```

выведет одну строку пример текста на экран во время компиляции.

A: ПУСК ПРОГРАММЫ

Компилятор автоматически устанавливает ключ F, затем компилирует, затем запускает LINK-KT, а затем (если не было ошибок) запускает полученный файл .EXE на исполнение в отдельном окне Windows.

B: ПРЕДУПРЕЖДЕНИЕ О МЕДЛЕННЫХ ПРЕОБРАЗОВАНИЯХ

Преобразования из FLOAT (24) в FLOAT(53) проводятся в PL/1-программе через промежуточное преобразование в текстовую строку, что довольно медленно. Установленный ключ заставит компилятор выдавать предупреждение «МЕДЛЕННО» при каждом таком найденном преобразовании в программе. Результат компиляции не отменяется. Смысл данного ключа – помочь найти все такие места для возможного устранения или выноса из циклов при ускорении расчетов.

C: ВЫВОД ПСЕВДОАССЕМБЛЕРА

Вывод генерируемого кода в виде команд псевдоассемблера. С помощью специального конвертора этот код может быть преобразован в настоящий ассемблер и затем оттранслирован транслятором RASM-KT.

D: ВЫВОД ПРОТОКОЛА В .PRN

Выводной поток компиляции будет помещен в файл с расширением .PRN.

E: ССЫЛКА ПО ИМЕНИ ВНУТРИ РЕКУРСИИ

Если этот ключ не задан и в программе есть рекурсивные процедуры (с атрибутом RECURSIVE), внутри которых, в свою очередь, есть вызов процедур, то все параметры таких вызовов передаются только по значению, т.е. для параметров создаются временные переменные. Это дает такой же эффект, как если бы Вы все параметры при вызове процедуры написали в круглых скобках, например: CALL F (X1,X2,X3); выполняется как

```
CALL F((X1),(X2),(X3));
```

Ключ E отменяет это правило и внутри рекурсивных процедур можно передавать параметры по имени. Если в программе нет рекурсивных процедур, значение ключа безразлично.

F: ВЫЗОВ LINK ПОСЛЕ ТРАНСЛЯЦИИ

Если компиляция прошла успешно, то сразу после нее будет вызван редактор связей LINK-KT для создания EXE-файла. Если компилируемая программа состоит из одного файла с процедурой, имеющей атрибут MAIN, данный ключ устанавливается автоматически, однако в этом случае, если LINK-KT не найдет всех процедур, сообщение об этом на экран не выдается, пока данный ключ не будет указан явно.

Пример: PLINK64 TEST.PL1 FH

G: ОТМЕНА СООБЩЕНИЯ «КОНЕЦ ПРОГРАММЫ»

В конце работы каждой PL/1-программы выдается сообщение «Конец программы». Если при компиляции задать этот ключ, выдача сообщения отменяется.

H: [] ВНУТРИ КОММЕНТАРИЯ РАЗРЕШАЮТ ВЛОЖЕННЫЕ КОММЕНТАРИИ

Если установлен данный ключ и внутри комментария встретился символ "[", то до встречи символа "]" все символы "*" заменяются на пробел. Это позволяет закомментировать фрагмент программы, в котором есть свои внутренние комментарии, например:

```
/*[
X=1;/*  пример внутреннего комментария */
]*/
```

I: ВЫВОД ПСЕВДОАССЕМБЛЕРА И ВЫВОД ИСХОДНОГО ТЕКСТА

Кроме строк исходного текста в листинге будет печататься сгенерированный машинный код и его представление на языке псевдо ассемблера, т.е. в стиле ассемлера RASM-KT.

J: ВЫВОД НЕДОСТУПНЫХ МЕСТ

Если компилятор обнаружит операторы, на которые невозможно передать управление (например, после оператора goto), будет выдана информация об этом. Если ключ не установлен - таких сообщений не выдается.

K: БЕЗ ВЫВОДА %INCLUDE

При компиляции на экран не выводятся операторы %INCLUDE и описания ключей.

L: ВЫВОД ИСХОДНОГО ТЕКСТА

Выдача листинга с номерами исходных строк и адресацией памяти, начиная с которого расположен код, сгенерированный из данной строки. Этот режим автоматически устанавливается, если был задан I. Глубина вложенности каждого блока индицируется латинской буквой.

M: РАЗРЕШЕНИЕ ПРЕДОПРЕДЕЛЕННЫХ %REPLACE

Русские эквиваленты ключевых слов разрешается загрузить перед компиляцией программы. Если ключ не установлен - это позволяет отменить использование русских ключевых слов, когда в программе они уже использованы как переменные.

N: ПОКАЗ ВЛОЖЕННОСТИ УРОВНЕЙ

Печать уровней вложенности блоков и процедур. Каждая пара PROCEDURE-END, BEGIN-END и DO-END считается за один уровень и отмечается буквами от «a» до «z». Начальный уровень главной программы – «a». Каждый BEGIN и DO увеличивают уровень на одну букву, каждая PROCEDURE - на две буквы. Используется при поиске нарушенных границ.

O: ОТКАЗ ОТ .OBJ

Отключение объектного кода и третьего прохода (файл .OBJ не будет создан).

P: ВКЛЮЧЕНИЕ ВСЕХ ИМЕН

В файл .OBJ будет включена специальная запись типа LOCSYM, содержащая имена всех меток и переменных в программе. Используя эту запись, LINK-KT создает файл имен .SYM в наиболее полном виде и этот файл при отладке позволит встроенному в PL/1-программу символьному отладчику установить связь между именами объектов программы и их адресами при выполнении программы.

Q: КОНТРОЛЬ ЦЕЛОЧИСЛЕННОГО ПЕРЕПОЛНЕНИЯ

Если этот ключ установлен, то после каждой арифметической операции с объектами типа FIXED BINARY компилятор начнет вставлять специальную команду, контролирующую переполнение. В случае переполнения сработает ситуация FIXEDOVERFLOW(3).

R: ОТЛАДКА

В файл .EXE будут вставлены специальные коды, отмечающие текущие строки и имена вызываемых процедур. В случае аварийного окончания программы, на экран будет выдана более подробная информация (см. ниже).

S: ВЫВОД РАСПРЕДЕЛЕНИЯ ПАМЯТИ

Вывод таблицы идентификаторов с указанием всех параметров (в том числе назначенных по умолчанию).

T: КОНТРОЛЬ ИНДЕКСОВ И SUBSTR

Этот ключ заставляет компилятор генерировать специальные команды проверки при каждом обращении в программе к элементу массива или к функции SUBSTR. При выполнении программы эти команды проверяют, соответствует ли текущее значение индекса описанию массива и допустимы ли операнды в SUBSTR. Если индекс выходит за границы массива, генерируется сигнал ERROR(16). Стандартная реакция программы на этот сигнал - окончание работы и выдача диагностики вида:

```
ОШИБКА (16) A:0040123B, ИНДЕКС ВНЕ ГРАНИЦ
(0040120A) СТЕК: 0040534F 00000000 0040123B # 004051BF 00000000 00401007
Конец программы
```

Если в SUBSTR(X,K,N) длина переменной X равна L и не выполняются условия 1<= K <= MAX(1,L) и 0 <= N <= L-K+1, то генерируется сигнал ERROR(19) ВНЕ SUBSTR.

Номер строки в программе можно найти по ключу R. Следует отметить, что включение этих проверок может существенно увеличить код программы и замедлить ее работу.

U: ВЫВОД ПРИ КОМПИЛЯЦИИ НА ЭКРАН

Этот ключ эквивалентен по действию ключу N, однако выдает только текст исходной строки (без однострочных комментариев). Его можно использовать для выдачи произвольного текста при компиляции на экран, например:

```
%_U
/* МОДЕРНИЗИРОВАННАЯ ВЕРСИЯ */
%_-U
```

V: ВЫВОД ВНУТРЕННЕГО КОДА

Вывод кода программы на промежуточном языке для контроля компилятора.

W: КОРОТКОЕ ВЫПОЛНЕНИЕ ЦЕПОЧЕК AND И OR

Если в программе есть операторы типа:

```
IF X1 & X2 & X3 ... THEN ...    или
IF X1 ! X2 ! X3 ... THEN ...
```

то при отсутствии этого ключа обязательно вычисляются все выражения X1, X2,... Если ключ W установлен, то генерируются дополнительные команды проверки и перехода. В первом же случае Xn='0'B (для AND) или Xn='1'B (для OR), остальные выражения Xn+1 уже не вычисляются, так как окончательное значение сразу известно.

X: ВЫВОД НЕИСПОЛЬЗУЕМОГО

Выдача списка переменных и подпрограмм, которые описаны, но к которым нет явного обращения в программе или в подпрограммах. "*" отмечены неиспользуемые подпрограммы, которые компилятор удалил из объектного модуля.

Y: СЛУЖЕБНЫЙ

Используется для контроля работы самого компилятора.

Z: ВЫРАВНИВАНИЕ ДАННЫХ

Выравнивание данных для повышения скорости работы программы. Объекты с нечетной длиной не выравниваются, объекты с длиной кратной слову – выравниваются по словам, объекты с длиной кратной двойному слову -

выравниваются по двойным словам, объекты с длиной кратной 8 байтам - выравниваются по 8 байт.

По умолчанию заданы ключи H, J, M, P, W, Z.

17.2. Ключи отладочной печати

Данное средство, похожее на условную трансляцию, не обязательно использовать только для отладочной печати. Однако часто требуется вывод какой-либо отладочной информации, который желательно включать и выключать, не трогая текст самой программы. Для этого можно в тексте программы (обязательно в начале строк) расставить конструкции из символа процента и цифры от 0 до 9, например:

```
…
%1 PUT SKIP LIST('ОТЛАДКА 1');
…
%2 PUT SKIP LIST('ОТЛАДКА 2');
```

По умолчанию эти строки будут рассматриваться как однострочные комментарии. Если при трансляции задать ключ 1 или 2, например:
PLINK64 TEST.PL1 1 или PLINK64 TEST.PL1 2 то в исходном тексте появится или первый оператор примера или сразу оба.

Идея здесь в том, чтобы задать уровень, до которого нужно транслировать строки. Сами строки, естественно, могут содержать любые операторы, а не только печать. Процент и ноль просто заменяются на пробелы (т.е. такая строка транслируется всегда).

Необходимый уровень можно задать и в самой программе конструкцией из процента, вопросительного знака и цифры от 0 до 9. Например, если задать в начале программы %?9, все строки, начинающиеся с %0-%9 будут транслироваться.

17.3. Предопределенные константы в PL/1-программе

Имеется встроенная в компилятор системная переменная ?DATE (не путать со встроенной функцией DATE), которая уже описана как DCL ?DATE CHAR (8) EXT. При сборке загрузочного модуля, LINK-KT подставит в эту переменную текущую дату сборки и, таким образом, можно выводить дату текущей версии Вашей программы просто оператором типа:

```
PUT SKIP LIST ('ВЕРСИЯ от',?DATE);
```

Имеется также ряд встроенных в компилятор констант, которые удобно использовать при отладочных выводах. Эти константы описаны в предопределенном %REPLACE со следующими значениями типа CHAR VAR:

- ?md – заменяется в тексте программы на имя компилируемого модуля. Пример использования:

```
PUT SKIP LIST ('Модуль',?MD);
```

- ?fl – заменяется в тексте программы на имя компилируемого файла. Пример использования:

```
PUT SKIP LIST ('Файл',?FL);
```

- ?vr или ?ver – заменяется в тексте программы на номер версии компилятора в виде текста. Пример использования:

```
PUT SKIP LIST ('Версия',?ver);
```

- ?ln или ?line – заменяется в тексте программы на текущий номер строки компилируемого модуля в виде текста. Пример использования:

```
PUT SKIP LIST (' Строка ',?LN);
```

- ?proc – заменяется в тексте программы на текущее имя компилируемой процедуры. Пример использования:

```
PUT SKIP LIST ('Процедура',?PROC);
```

17.4. Работа компилятора

Компилятор расположен в файле PLINK64.EXE и обрабатывает исходный текст за 3 прохода. Сам компилятор – 32-разрядная программа Windows.

На первом проходе обрабатываются все описания, и строится таблица всех объектов программы.

На втором проходе анализируется структура программы и создается ее специальное внутреннее представление (обратная польская запись).

На третьем проходе из внутреннего представления генерируется перемещаемый объектный код в формате OMF (Intel Technical Specification 121748-001), доработанного для x86-64, вычисляются значения переменных и констант, которые можно определить до начала работы программы и т.д.

Приблизительно 300 наиболее часто встречающихся конструкций заменяются вызовом специальных системных подпрограмм-«экстракодов» из системной библиотеки PL1LIB.L86. Все имена таких подпрограмм начинаются со знака «?» (см. приложение). В этой же библиотеке хранятся и все встроенные функции языка.

Номер каждого прохода (1, 2 или 3) высвечивается в верхней строке окна работы компилятора. Если текст берется из файла, вызванного оператором %INCLUDE, то высвечивается еще и признак "I".

Если первым параметром задать символ "$", например:
PLINK64 TEST.PL1 $I, то на экран не будет выдаваться информационная картинка компилятора.

Если были обнаружены ошибки, то выдается строка, вызвавшая ошибку, и краткое сообщение об ошибке (см. раздел 21). Ошибочное место в строке отмечается знаком "?". После выдачи сообщения о первой ошибке компиляция прекращается и выдается запрос: "ИДТИ ? (Esc-НЕТ)"

Теперь возможны следующие действия:

- нажать клавишу "0" или "Esc" - компиляция прекратится и будет выдано сообщение "КОМПИЛЯЦИЯ ПРЕКРАЩЕНА";
- нажать простую клавишу (например, пробел) - компиляция пойдет дальше, не останавливаясь после каждого сообщения об ошибках, и так до конца программы или до следующего нажатия клавиши;
- нажать управляющую клавишу (например, курсор вниз) - компиляция пойдет дальше и остановится на следующей ошибке; Если при этом компилировался файл из %INCLUDE, компиляция также пойдет до следующей ошибки, но еще на экран будет выдано имя текущего файла из %INCLUDE.

Большинство ошибок не фатальные, после их обнаружения компиляция продолжается. После 255 ошибок компиляция прекращается. Однако рекомендуем не продолжать трансляцию, а прекращать ее после первой же ошибки и исправлять эту ошибку, так как следующие ошибки могут быть «наведенными» первой ошибкой.

В конце работы компилятор выдает размер сегментов команд и данных, которые занимает сгенерированная программа, а также процент использования места в своей таблице идентификаторов (если больше 10%). При ключе «Y» выдается так же процент использования внутреннего стека в выражениях (если больше 25%).

17.5. Запуск PL/1-программ и разбор командной строки

После получения с помощью LINK-KT загрузочного модуля в виде файла с расширением .EXE, его можно исполнить директивой:

<имя> [<параметры>]

После окончания программы выдается сообщение "Конец программы". Однако это сообщение можно отменить ключем компиляции G, или если описать системную переменную:
DCL ?DO_KONEC BIT EXT; и присвоить ей значение не ноль.

Если параметры в командной строке являются именами файлов, их можно достать конструкцией в операторе OPEN:

... TITLE ('$1.$1') или ... TITLE('$2.$2')

Если необходимо достать всю строку параметров, можно использовать параметр главной процедуры:

```
TEST:PROC (GET_LINE) OPTIONS (MAIN);
DCL GET_LINE CHAR (*) VAR;
...
```

После начала работы программы в переменную GET_LINE попадет строка параметров из вызова, начиная с первого пробела после имени вызванной программы. Если параметров не было, переменная GET_LINE будет пустой (т.е. будет иметь нулевую длину).

17.6. Аварийное сообщение при выполнении программы

При работе программы могут возникнуть различные ошибочные ситуации, которые можно перехватить соответствующими ON-операторами. Если для данной ситуации ON-оператор не описан, но ситуация фатальная - работа программы прекращается и на экран выдается стандартное сообщение вида:

<СИТУАЦИЯ> (КОД) A:<АДРЕС> <НОМЕР СТРОКИ> <ОШИБКА> <ИМЯ ФАЙЛА> <СООБЩЕНИЕ>
ВХОД:<имя1>, <имя2>, ...
(<ПОСЛЕДНИЙ АДРЕС>) СТЕК:aaaa bbbb cccc dddd # eeee ffff gggg hhhh

<СИТУАЦИЯ> - одна из стандартных ситуаций языка,
(КОД) - стандартный код, расшифровывающий ситуацию.

A:<АДРЕС> абсолютный адрес в программе, где случилась ситуация. Определяется средствами операционной системы. Если ситуацию обнаружили встроенные средства PL/1 – адрес не выводится.

<НОМЕР СТРОКИ> - номер строки исходного текста модуля (если был задан режим компиляции R) и первые две буквы имени текущего модуля.

<ОШИБКА> - код ошибки в смысле операционной системы, обычно выдается при файловом обмене, например, код 2 – не найден файл, 3 – не найдена папка.

<ИМЯ ФАЙЛА> - конструкция типа ФАЙЛ:<имя1>=<имя2>, где <имя1> и <имя2> соответственно внутреннее представление файла в программе (имя переменной типа FILE) и внешнее представление файла (имя файла), например:

ФАЙЛ:SYSPRINT=CON

<СООБЩЕНИЕ> - текст, поясняющий ошибку, выдается в случае, если достаточно информации для ее анализа.

<ВХОД> - имена последних пяти вызванных процедур (если был задан режим компиляции R).

В третьей строке выдается последний адрес, где программа еще работала нормально (если был задан ключ R), а также содержимое восьми слов стека программы в шестнадцатеричном виде, сначала 4 верхние, затем знак # и затем 4 нижние. Если текущая глубина стека меньше 8, знак # не ставится.

Пример аварийной выдачи:

```
ОШИБКА (19) 0007 TE, ВНЕ ПОДСТРОКИ
(0040120A) СТЕК: 0040125E 00000000 00405191 00000000 00401007
Конец программы
```

В данном примере, произошел выход индекса за пределы строки при выполнении функции SUBSTR.

Ошибку обнаружили средства PL/1 в строке №0007 модуля, имя которого начинается с «TE» (в отладочной информации каждой строки есть место только для двух первых букв имени модуля).

Операционная система ошибок не обнаружила (адреса нет).

Последний адрес, где программа работала еще нормально 40120A.

В стеке находятся:

- адрес возврата из встроенной функции SUBSTR 40125E
- адрес окончания программы (адрес перехода на оператор STOP 405191
- адрес начала выполнения EXE-файла (всегда 401007)

18. ВНУТРЕННЕЕ ПРЕДСТАВЛЕНИЕ ДАННЫХ

Знание внутреннего представления данных необходимо для использования базированных переменных с наложением и для связи программ на PL/1 с программами на других языках, а также для отладки.

18.1. Представление FIXED BINARY

В зависимости от заданной точности, данные FIXED BINARY могут занимать один (FIXED BINARY(7)), два (FIXED BINARY(15)) или четыре (FIXED BINARY(31)) или восемь (FIXED BINARY(63)) байт памяти. Для процессоров x86 действует правило: младший байт слова находится по младшему адресу, таким образом константа '1234'b4 будет помещена в двух соседних байтах как 34 12. Старший бит в данных обозначает знак, если число отрицательное - оно пишется в дополнительном коде. Примеры чисел:

FIXED BINARY (7)

Число	представление
0	00
1	01
-1	FF
127	7F

FIXED BINARY (15)

Число	представление
0	00 00
1	01 00
-1	FF FF
127	7F 00

FIXED BINARY (31)

Число	представление
0	00 00 00 00
1	01 00 00 00
-1	FF FF FF FF
127	7F 00 00 00

FIXED BINARY (63)

Число	представление
0	00 00 00 00 00 00 00 00
1	01 00 00 00 00 00 00 00
-1	FF FF FF FF FF FF FF FF
127	7F 00 00 00 00 00 00 00

18.2. Представление FLOAT BINARY и FLOAT DECIMAL

В формате IEEE-754 FLOAT BINARY и FLOAT DECIMAL представляются одинаково.

Для чисел с обычной точностью биты используются следующим образом:

	знак	порядок	мантисса
Номера бит	31	30-23	22-0

В формате IEEE-754 мантисса всегда нормализована, т.е умножена так, чтобы ее первый разряд всегда был 1, раз это так, этот разряд можно не хранить. Такой бит мантиссы называется неявным. Двоичная точка находится непосредственно справа от неявного бита. Порядок в формате IEEE-754 сдвинут на 127 (или 7F), так, что 80 означает +1, 7E означает -1 и т.д.

Разберем число в формате IEEE-754: 00 00 C0 3F.
Как битовая строка, это число выглядит так:

```
3     F     C     0     0     0     0     0
0011  1111  1100  0000  0000  0000  0000  0000
```

Старший бит показывает, что число положительное, а порядок без смещения 7F дает 0:

```
0     0111 1111          1000 0000 0000 0000 000
знак  порядок            мантисса
```

С учетом неявного бита и точки, мантисса выглядит так:

```
1. 100 0000 0000 0000 0000 0000
```

Переводим в десятичный вид:

```
2**0 + 2**-1 + 0 =1.5
```

Таким образом, 00 00 C0 3F в формате IEEE-754 это 1.5.

Для чисел с двойной точностью биты используются следующим образом:

	знак	порядок	Мантисса
Номера бит	63	62-52	51-0

Порядок здесь сдвинут на 1023 (или 3FF), так, что 400 означает +1, 3FE означает -1 и т.д.

Разберем число двойной точности: 00 00 00 00 00 C0 43 C0.

```
C     0     4     3     C     0     0     0
1100  0000  0100  0011  1100  0000  0000  0000 ...
знак  порядок           мантисса
```

С учетом неявного бита и точки, мантисса выглядит так:
1. 0011 1100 0000 0000 0000 0000 ...
Порядок:
4 0 4
0100 0000 0100 отнимаем 3FF, получаем 5
Учитываем порядок:
10011 1.100 0000 0000 0000 0000 ...
Переводим в десятичный вид:
2**5 + 2**2 + 2**1 + 2**0 + 2**-1 = 39.5 и учитываем знак минус

Таким образом, 00 00 00 00 00 C0 43 C0 в формате IEEE-754 это -39.5.

Замечание: *данные, с которыми работает процессор, тоже представляются в формате IEEE-754.*

18.3. Представление FIXED DECIMAL

Данные FIXED DECIMAL представляются в десятичном дополнительном упакованном коде BCD (Binary Coded Decimal). Каждая цифра BCD записывается в половину байта. PL/1 помещает младшую значащую пару цифр BCD по старшему адресу. Первая позиция BCD резервируется для знака. Знак для положительных чисел - ноль, для отрицательных - девять. Число байт для FIXED DECIMAL определяется заданной точностью и равно FLOOR((p+2)/2), где p допустима от 1 до 15.

Для примера, рассмотрим число 12345 с точностью 5, для которого PL/1 выделит FLOOR((5+2)/2)=3 байта памяти. Число будет записано как: 45 23 01

Отрицательное число записывается в дополнительном до 9 коде, дополнительный код здесь получается вычитанием из 9 и прибавлением единицы. Например, -2 имеет код (9-2)+1=8 и с точностью 2 будет записано как 98, а с точностью 5 будет выглядеть: 98 99 99.

18.4. Представление CHARACTER

Данные представляются в двух видах:

- для строки фиксированной длины, байты символов помещаются последовательно от младшего адреса к старшему.

- для строки переменной длины (VARYING), сначала пишется байт длины (от 0 до 254), а затем байты символов, как в предыдущем случае. Таким

образом, строка переменной длины занимает число байт, равное максимальной длине плюс еще один байт длины.

Например, строка 'Walla Walla Wash' в первом случае будет представлена как: W a l l a W a l l a W a s h
57 61 6C 6C 61 20 57 61 6C 6C 61 20 57 61 73 68
а во втором случае:
W a l l a W a l l a W a s h
10 57 61 6C 6C 61 20 57 61 6C 6C 61 20 57 61 73 68

18.5. Представление BIT

Битовые строки, в зависимости от заданной длины, представляются или одним (1-8 битов) байтом или двумя байтами (9-16 битов) или четырьмя (17-32 бита) или восьмью (33-64 бита). Помните, что действует правило: младший байт слова находится по младшему адресу. Если число бит не равно 8 или 16 или 32, лишние биты справа игнорируются. Примеры представления строк:

Запись в PL/1	Описание	Представление в памяти
'1'b	BIT (8)	00000001
'1'b	BIT (16)	0000001 00000000
'A0'b4	BIT (8)	10100000
'A0'b4	BIT (16)	10100000 00000000
'1234'b4	BIT (16)	00110100 00010010
'12345678'b4	BIT (32)	01111000 01010110 00110100 00010010

18.6. Представление POINTER

Данные представляются в восьми байтах - там находится адрес памяти переменной.

18.7. Представление ENTRY и LABEL

Данные представляются в восьми байтах для LABEL (адрес метки) и для ENTRY (смещение процедуры).

18.8. Представление FILE

Каждый идентификатор файла в PL/1 ассоциируется со статически выделяемой для него структурой блока параметров файла (FPB) и динамически выделяемой (во время OPEN) и освобождаемой (во время CLOSE) структурой управляющего блока файла (FCB), в конце которой размещается и буфер обмена

для данного файла, если файл отображается на память - буфера нет.

Файл в PL/1 фактически представляет собой указатель на структуру FPB, причем после открытия файла, первые восемь байт FPB содержат указатель на FCB.

Можно описать переменные типа POINTER и оператором инициации (INIT) присвоить им значения файлов. Напомним, что напрямую пересылать значения файлов можно только в переменные типа FILE VARIABLE.

18.9. Структура блока параметров файла (FPB)

Смещ	Длина	Содержимое
+00	8	указатель на FCB, результат обращения к ALLOCATE внутри OPEN. Показывает на выделенные 35H+<длина буфера> байт, после CLOSE эта память опять освобождается
+08	2	позиция в строке для STREAM, первоначально 1
+0A	2	номер строки для STREAM, первоначально 1
+0C	2	номер страницы для STREAM, первоначально 1
+0E	1	считанный, но еще не обработанный символ
+0F	8	значение стека (RSP) после возврата из ON-оператора
+17	8	адрес возврата после ON-оператора
+1F	2	размер строки для STREAM, берется из OPEN (LINESIZE)
+21	2	размер страницы для STREAM, берется из OPEN (PAGESIZE)
+23	4	размер записи для KEYED, берется из OPEN ENV(F)
+27	4	размер буфера, берется из OPEN ENV(B), если -1 - то буфера нет и файл отображается на память. Если буфер задан, он не может быть короче размера записи (если запись есть)
+2B	2	атрибуты файла из OPEN (см. следующую таблицу)
+2D	1	длина внутреннего (в PL/1) имени файла
+2E	1-31	внутреннее (в PL/1) имя файла

18.10.Атрибуты файла из OPEN

Номер бита в поле 1B	Признак атрибута
XXXX XXXX XXXX XXX1	ENV (опция A)
XXXX XXXX XXXX XX1X	PRINT
XXXX XXXX XXXX X1XX	KEYED
XXXX XXXX XXXX 1XXX	DIRECT
XXXX XXXX XXX1 XXXX	SEQUENTIAL
XXXX XXXX XX1X XXXX	UPDATE
XXXX XXXX X1XX XXXX	OUTPUT
XXXX XXXX 1XXX XXXX	INPUT
XXXX XXX1 XXXX XXXX	RECORD
XXXX XX1X XXXX XXXX	STREAM
XXXX X1XX XXXX XXXX	FILE
XXXX 1XXX XXXX XXXX	TITLE
XXX1 XXXX XXXX XXXX	LINESIZE
XX1X XXXX XXXX XXXX	PAGESIZE
X1XX XXXX XXXX XXXX	ENVIROMENT

В своей программе Вы можете описать приведенную ниже структуру для непосредственного доступа ко всем внутренним данным переменной типа FILE.

```
declare
 1 STRUC_FPB  based,
  2 FCB           ptr,          /* указатель на FCB */
  2 COL           fixed,        /* текущий номер позиции в строке */
  2 LINENO        fixed,        /* текущий номер строки */
  2 PAGENO        fixed,        /* текущий номер страницы */
  2 SYMBOL        char,         /* считанный, еще не обработанный символ */
  2 SP_ON         bit(64),      /* стек после возврата из ON-оператора */
  2 RET_ON        bit(64),      /* адрес возврата после ON-оператора */
  2 LINESIZE      fixed,        /* размер строки */
  2 PAGESIZE      fixed,        /* размер страницы */
  2 F             fixed(31),    /* размер записи */
  2 B             fixed(31),    /* размер буфера */
  2 ATTRIBUTS     bit(16),      /* атрибуты */
  2 INT_NAME      char(31) varying; /* имя файла в PL/1 */
```

18.11.Структура управляющего блока файла (FCB)

Смещ	Длина	Содержимое
+0	1	тип файла 0 - INPUT, 1 - OUTPUT, 2 – UPDATE (INPUT/OUTPUT)
+1	1	драйвер 0 - текущий, 1 - A:, 2 - B: и т.д.
+2	8	внешнее имя файла (дополняется до 8 пробелами)
+A	3	расширение файла (дополняется до 3 пробелами)
+D	4	номер открытого файла (HANDLE)
+11	4	размер записи для работы по ключу (KEY) или для KEYTO
+15	4	счетчик в буфере, когда достигает размера буфера, производится обмен буфера с диском, после чего этот счетчик сбрасывается в ноль. Если счетчик ноль и текущий размер буфера ноль - возникает ситуация ENDFILE
+19	4	текущий размер буфера, после очередного обмена сюда пишется действительное число считанных или записанных байт
+1D	4	размер файла в байтах при открытии
+21	4	текущий номер записи (значение KEY или KEYTO)
+25	4	адрес нулевой записи в файле, каждый раз прибавляется к адресу записи (т.е. к размеру записи, умноженному на номер записи). При открытии файла устанавливается в ноль.
+29	8	смещение буфера, при открытии устанавливается на FCB+35H
+31	4	число записей в файле, устанавливается один раз при открытии, корректируется при записи оператором WRITE KEYFROM
+35	...	буфер при открытии, если был задан -1, то эта память для него не выделяется

В своей программе Вы можете описать приведенную ниже структуру для непосредственного доступа ко всем внутренним данным этой, динамически создаваемой, структуры.

```
declare
 1 STUC_FCB    based,
  2 TYPE       fixed(7),    /* 0 - INPUT, 1 - OUTPUT, 2 - UPDATE */
  2 DRV        fixed(7),    /* драйвер 0 - текущий, 1 - A: ... */
  2 NAME_1     char(8),     /* внешнее имя файла */
  2 NAME_2     char(3),     /* расширение файла */
  2 HANDLE     fixed(31),   /* номер открытого файла */
  2 REC_SIZE   fixed(31),   /* размер записи */
  2 COUNT      fixed(31),   /* текущая позиция в буфере */
  2 BUF_SIZE   fixed(31),   /* текущий размер буфера */
  2 SIZE       fixed(31),   /* размер файла при открытии */
  2 N_REC      fixed(31),   /* текущий номер записи */
  2 N0_OFF     bit(32),     /* смещение нулевой записи */
  2 BUF_OFF    bit(64),     /* смещение буфера */
  2 MAX_REC    fixed(31);   /* число записей в файле */
```

Пример доступа к внутренним структурам данных типа FILE

```
 ...
declare
  P1      ptr static init(FILE_1),
  P2      ptr static init(FILE_2),
  FPB     ptr,
  FILE_1  file,
  FILE_2  file;
 ...
open file(FILE_1) print title('C:\TEST\README.TXT');
FPB=P1;
put skip list(FPB->LINENO);
put skip list(FPB->PAGENO);
put skip list(FPB->FCB->REC_SIZE);
put skip list(FPB->FCB->COUNT);
 ...
open file(FILE_2) input title('$1.$1');
FPB=P2;
put skip list(FPB->INT_NAME);
put skip list(FPB->FCB->SIZE);
 ...
```

18.12.Размещение агрегатов данных

PL/1 пишет агрегаты данных последовательными участками байт. Массив из BIT (1) не будет выравниваться, но каждый бит будет писаться с начала нового байта. Элементы массива располагаются последовательно, так, что правый индекс меняется быстрее всего. Например: declare A (2,2,2); расположится в памяти:

1,1,1	1,1,2	1,2,1	1,2,2	2,1,1	2,1,2	2,2,1	2,2,2
1	2	3	4	5	6	7	8

18.13.Внутреннее представление формата EDIT

В данной версии компилятора нельзя задать формат для EDIT с заранее неизвестными длинами полей. Т.е. нельзя вместо формата F (10,5) записать конструкцию типа F(X+10,Y/2). Формат располагается среди команд, а не данных. Однако, зная внутреннее строение формата его можно менять в программе.

Формат начинается байтом 80, после которого следует коэффициент повтора, затем собственно данные формата и байт конца 81. Данные формата состоят из байта типа и затем одного или двух байтов значений. В случае шаблона за типом следует сама строка шаблона.

Тип	элемент	параметры
0	F	Два байта – общая длина и длина дробной части
1	E	Два байта – общая длина и длина дробной части
2	P	Байт длины строки и затем сама строка шаблона
3	A	Байт длины
4	B	Байт длины
5	B1	Байт длины
6	B2	Байт длины
7	B3	Байт длины
8	B4	Байт длины
9	TAB	Байт количества табуляций
A	LINE	Байт номера строки
B	X	Байт числа пробелов
C	SKIP	Байт числа пропусков
D	COL	Байт номера позиции
E	R	-
F	PAGE	Байт числа пропусков страниц

Замечание: *байт FF означает значение, принятое по умолчанию.*

Пример размещения формата в памяти:

```
(       SKIP,  F (10,5),   B4 (4),X (3), 10 (     B1,     A,    TAB (6),A(50) )  );

80 00 0C FF 00 0A 05 08 04 0B 03 80 0A 05 FF 03 FF 09 06 03 32 81 81
    |                                        |
        Коээфициенты повторения
```

Пример динамического изменения формата (номера колонки с 5 до 16) в программе:

```
T:PROC MAIN;

R1:FORMAT (SKIP (3),COL (5),F (10,5));

DCL
X           FLOAT STATIC INIT(12345.6789e0),
P           PTR    DEF (R1),
P1 (100) BIT (8) BASED (P);

PUT SKIP EDIT (X)(R (R1));        // НОМЕР КОЛОНКИ 5
P1(6)='10'B4;
PUT SKIP EDIT (X)(R (R1));        // НОМЕР КОЛОНКИ ИЗМЕНИЛСЯ
END;
```

Вывод программы:

```
      123.78900


                  123.78900
Конец программы
```

19. СОГЛАШЕНИЕ О ПЕРЕДАЧЕ ПАРАМЕТРОВ МЕЖДУ ПРОГРАММАМИ НА PL/1 И НА ЯЗЫКЕ АССЕМБЛЕРА

При передаче параметров, компилятором в памяти создается таблица адресов параметров в виде непрерывного массива, где под каждый адрес отводится восемь байт. Адрес самого массива помещается в регистр RBX.

```
CALL F (X1,X2,X3, ... Xn);
```

Список адресов параметров в памяти:

```
АДРЕС X1 -> RBX
АДРЕС X2
АДРЕС X3
...
АДРЕС Xn
```

Таким образом, адрес 1-ого параметра в ассемблерной подпрограмме можно достать как [RBX], 2-ого как [RBX]+8, 3-его как [RBX]+16 и т.д.

Например, достать первый байт первого параметра в регистр AL можно командами:

```
MOV RSI, [RBX]
MOV AL, [RSI]
```

В случае если ассемблерная подпрограмма является подпрограммой-функцией и должна возвращать (кроме параметров) значение, оно передается следующим образом:

- если процедура-функция типа BINARY FIXED(1-7), то значение помещается в AL;
- если процедура-функция типа BINARY FIXED(8-15), то значение помещается в BX;
- если процедура-функция типа BINARY FIXED(16-31), то значение помещается в EBX;
- если процедура-функция типа BINARY FIXED(32-63), то значение помещается в RBX;
- если процедура-функция типа BINARY FLOAT, то значение помещается в стек в виде 4-х (или 8) байтов формата IEEE-754 младшим байтом мантиссы вперед (при этом в стеке всегда выделяется 8 байт);
- если процедура-функция типа DECIMAL FIXED, то значение помещается в стек в виде двоично-десятичных байтов, последний - знак;

- если процедура-функция типа CHAR, то значение помещается в стек в виде байтов текста, в AL помещается длина строки;
- если процедура-функция типа CHAR VAR, то значение помещается в стек в виде байтов текста, а затем в AL записывается байт длины строки;
- если процедура-функция типа BIT(1-8), то значение помещается в AL;
- если процедура-функция типа BIT(8-16), то значение помещается в BX;
- если процедура-функция типа BIT(17-32), то значение помещается в EBX;
- если процедура-функция типа BIT(33-64), то значение помещается в RBX;
- если процедура-функция типа PTR, ENTRY или LABEL, то значение помещается в RBX, причем, если возврат идет оператором GOTO, обрабатывающая программа должна восстановить стек;

Внешние переменные (с атрибутом EXT) внутри ассемблерной программы необходимо описывать как отдельные сегменты, входящие в группу DGROUP, например, в PL/1-программе:

```
DCL  1 STRUC (1:100) EXT,
         2 X1  BIT (8),
         2 X2  BIT (8);
                  ...
```

В ассемблерной программе:

```
STRUC      DSEG  COMMON
@STRUC    RB 0
DGROUP GROUP STRUC

CSEG A8
...
LEA ESI, @STRUC  ; загрузили в ESI адрес STRUC
LLODSB                ; достали X1(1) в сумматор
...
```

20. ПОДПРОГРАММЫ РАБОТЫ С ДИНАМИЧЕСКОЙ ПАМЯТЬЮ

В системной библиотеке PL1LIB.L86 имеется несколько подпрограмм для работы с динамической памятью - «кучей», они работают со словами (2 байта), поскольку память всегда выделяется по адресу, кратному 2:

TOTWDS - выдает общее число слов (dw) динамической памяти, описание в PL/1: DCL TOTWDS RETURNS (FIXED (63)).
Рекомендуется использовать эту подпрограмму при анализе ситуации ERROR (7);

MAXWDS - выдает размер в словах (dw) максимального непрерывного куска свободной динамической памяти, описание в PL/1:
DCL MAXWDS RETURNS (FIXED (63)).
Рекомендуется использовать эту подпрограмму при анализе ситуации ERROR (7). Обе функции возвращают -1 в случае какой-либо ошибки.

STKSIZ - возвращает текущий размер стека в байтах. По умолчанию весь стек 10000H байт. Описание в PL/1: DCL STKSIZ RETURNS (FIXED (31)).

?TEST_MEM – полная проверка целостности памяти. В случае разрушения «кучи» возникает ситуация ERROR (38) с указанием адреса, где проверка еще показывала правильность. Описание в PL/1: DCL ?TEST_MEM ENTRY;

?TEST_MEM_KVANT – частичная проверка целостности памяти. В случае разрушения «кучи» возникает ситуация ERROR (38) с указанием адреса, где проверка еще показывала правильность. В отличие от предыдущей процедуры проверяет не всю «кучу», а лишь очередные 100 элементов из нее. Предназначена для постоянного вызова при работе программы, так как в отличие от предыдущей процедуры работает за определенный и всегда одинаковый промежуток времени. Дойдя до конца, начинает проверку заново. Описание в PL/1:
DCL ?TEST_MEM_KVANT ENTRY;

21. СООБЩЕНИЯ ОБ ОШИБКАХ ПРИ КОМПИЛЯЦИИ

21.1. Выдача сообщений об ошибках

При обнаружении ошибок, на экран выдается строка, вызвавшая ошибку и краткое сообщение. Ошибочная позиция отмечается знаком "?". После обнаружения первой ошибки, процесс компиляции приостанавливается и выдается запрос: ИДТИ ? (Esc-НЕТ), при нажатии "ESC" или "0" компиляция прекращается. Если ошибок не было, нажатие клавиш не влияет на работу компилятора.

При фатальных ошибках (отмеченных ниже *) компиляция прекращается.

21.2. Общие ошибки

БОЛЬШЕ 1 МБАЙТА ФАЙЛ *	Файл с исходным текстом (без учета %INCLUDE) не должен превышать мегабайта. Каждый %INCLUDE тоже не должен превышать мегабайта.
ДИСК ПОЛОН *	Все пространство на диске заполнено. Удалите лишние файлы.
ЗНАЧЕНИЕ НЕДОПУСТИМО	Число превышает разрядность 2**31-1, т.е. выходит за пределы четырех байт.
КОМПИЛЯТОР *	Компилятор ошибся (может быть из-за ошибок, встретившихся ранее).
КОМПИЛЯЦИЯ ПРЕКРАЩЕНА*	Число ошибок превысило 255 или компиляция прервана с клавиатуры.
КОНЕЦ ФАЙЛА *	Ошибки Windows при работе с файлом .OBJ
МАЛ СТЕК *	Разрушена память компилятора
НЕ КЛАССА УПРАВЛЯЕМОЙ ПАМЯТИ	Массивы с динамически изменяемыми границами могут быть только для переменных класса CTL (CONTROLLED)
НЕ ЧИТАЕТСЯ *	Ошибка чтения файла
НЕЛЬЗЯ СОЗДАТЬ *	Часто, когда транслятор запущен сразу в двух окнах Windows.
НЕТ КАВЫЧКИ *	Нет баланса апострофов в строковых константах.
НЕТ ФАЙЛА *	Не найден файл с исходным текстом
НЕОЖИДАННЫЙ КОНЕЦ ФАЙЛА *	Конец файла был обнаружен перед логическим концом программы. Обычно из-за нечетного числа скобок, границ комментариев или кавычек. Используйте ключ "N" для проверки DO и END.
ОЧЕНЬ ДЛИННАЯ ЛЕКСЕМА	Длина идентификатора больше 31 символа или длина строки больше 254 символов. Возможно,

	пропущен апостроф
ОШИБКА ВСТАВКИ	Неправильно сформирован оператор %INCLUDE, его общий вид: %INCLUDE 'D:<путь> <имя>';, где D выбранный диск, а <имя> - полное имя файла с расширением. Может быть внутри не закрыт комментарий ?
СТРОКА > 245	Строка исходного текста превысила 245 символов и была усечена.

21.3. Ошибки при первом проходе компилятора

БАЛАНС	Скобки в выражении не сбалансированы (число "(" не равно числу ")").
БЛОК В СТРОКЕ X ПЕРЕМЕННАЯ: V УЖЕ НЕТ ПАМЯТИ	Блок, начинающийся со строки X, содержит переменную V, при попытке размещения в памяти которой размер блока превысил допустимый объем памяти
БЛОКОВ > 31	Следующий уровень PROCEDURE, DO и BEGIN блоков превышает 31. Упростите программу
ДЛИНА	Длина данного элемента превышает максимальный размер для этого элемента. Возможно, пропущен апостроф в строковой константе.
ЕСТЬ ВНЕШ	Предупреждение, что уже есть EXTERNAL переменная с таким именем.
ЕСТЬ В ПЛ	Предупреждение, что в языке есть встроенная функция с точно таким же именем. Могут быть ошибки, если в других модулях программы используется встроенная функция, а не данная переменная
КОНСТАНТА	Встреченная константа в формате, неправильном для этого типа констант
КОНФЛИКТ	Заданные в операторах DECLARE атрибуты противоречивы
МЕТКА	Неправильная метка. Метка может быть только с постоянным индексом.
НЕ ОПИСАНО: *	Перечисленные параметры процедур не были описаны.
НЕ СДЕЛАНО	Попытка использовать средства PL/1, не реализуемые данной версией компилятора.
НЕ ТАМ %3	Оператор %REPLACE неправильно размещен относительно структуры блока. Эти операторы

	должны стоять в самом внешнем блоке и до использования констант.
НЕ ФУНКЦИЯ	Атрибут BUILTIN применен к идентификатору, который не является встроенной функцией.
НЕ ЧИСЛО	В указанной позиции формата требуется числовая константа.
НЕ КОСВЕНН.	Указанный элемент используется в программе как элемент типа VARIABLE, но он не имел этого атрибута при описании.
НЕТ БАЗЫ ДЛЯ НАЛОЖЕНИЯ *	Для указанной переменной не соблюдены требования к атрибуту DEFINED (см. раздел 7.1.6).
ПЛОХ КОГДА	Неправильный оператор в теле ON-оператора: а) нельзя использовать для выхода оператор RETURN; б) если есть операторы DO и IF, требуется ограничить тело ON-оператора словами BEGIN и END.
ПЕРЕПОЛНЕНА ТАБЛИЦА *	Эта программа не может быть скомпилирована как один модуль. Разбейте ее на более мелкие модули.
РЕКУРСИЯ	Рекурсивная процедура содержит неправильное вложение блоков. Разрешается использовать только простые DO-операторы в таких процедурах.
СИНТАКСИС	Синтаксическая ошибка в указанном операторе.
СЛИШКОМ МНОГО CTL-ПЕРЕМЕННЫХ	Число переменных класса CTL по умолчанию 8. Это число нужно увеличить через строку реестра «CTL»
СТРАННО ОПИСАНИЕ	В описании переменная без атрибутов.
СТРУКТУРА	Указанная структура имеет слишком много уровней (более 255). Разбейте на подструктуры.
УЖЕ БЫЛО	Указанная переменная уже объявлялась в DCL в этом же блоке или метка встретилась более одного раза.
ШАБЛОН	Ошибка в записи шаблона (формат P).

21.4. Ошибки при втором проходе компилятора

БАЗА ПЛОХА	Неправильная ссылка на базируемую переменную: а) указатель определяет ссылку на не базируемую переменную; б) переменная указана в DCL как BASED (X), где X не простой указатель переменной или простой указатель вызова функции. Возможно только описание типа: BASED(P) или BASED(Q()).
БАЛАНС	Скобки в выражении не сбалансированы (число «(« не равно числу «)»).
БЕЗ ОТВЕТА	Попытка вернуть значение процедуре, которая описана без атрибута RETURNS, или наоборот - нет RETURN внутри процедуры-функции (ее имя выдается на экран).
БИТ ВЕЛИК	Значение битовой подстроки выходит за границу строки. Последний аргумент в SUBSTR для битовых строк должен быть константой (числом) в диапазоне от 1 до 64.
ВХОДНОЙ	Для фактического параметра, отмеченного "*", создана копия (он не может быть выходным). Это предупреждение.
ИМЯ КОНЦА	Имя после оператора END не соответствует имени процедуры.
КОНФЛИКТ	Исключающие друг друга атрибуты в OPEN.
МАЛО ПАМЯТИ	Не хватило места для внутреннего представления программы.
МЕТКА	Неправильно поставленная метка.
МНОГО ПАР.	Одна из ошибок: а) индекс массива не соответствует описанию; б) параметр процедуры является элементом массива; в) более 15 пар индексов, задающих границы массива; г) несоответствие граничных пар для массивов; д) формальный и фактический параметры не соответствуют друг другу.
НЕТ ОСНОВЫ	В этом контексте требуется базируемая переменная.
НЕ ЦЕЛОЕ	В этом контексте требуется целое выражение (FIXED BINARY). Эта ошибка не отменяет

	создание .OBJ.
НЕ "БИТ"	Параметр UNSPEC не 8-ми, не 16-ти и не 32-ти битовая строка.
НЕ МЕТКА	Неправильная метка в GOTO.
НЕ "ПРОЦ"	Неправильное имя в CALL.
НЕ ОПИСАН	Переменная не объявлена внутри данного блока или процедуры.
НЕ ТЕКСТ	В данном контексте требуется строка, а не выражение.
НЕ СДЕЛАНО	Попытка использовать средства PL/1, не реализуемые данной версией компилятора.
НЕ СКАЛЯР	В этом контексте требуется скалярное выражение, а не массив.
НЕ ПОСТОЯН	Использован оператор INIT при описании переменной без атрибута STATIC и EXTERNAL.
НЕ ФУНКЦИЯ	Ссылка на несуществующую встроенную функцию PL/1.
НЕ "ФАЙЛ"	В операторах работы с файлами указана переменная, не являющаяся переменной типа FILE или FILE VARIABLE.
НЕ ФОРМАТ	В операторах PUT EDIT или GET EDIT нет формата.
НЕТ ЗНАЧЕН	Там, где требуются вычисления, было использовано не вычисляемое выражение.
НЕТ КЛЮЧА	Выражение, используемое в операторах с KEYTO, KEYFROM или KEY не переменная типа FIXED BINARY.
НУЖНО ИМЯ	В этом контексте требуется идентификатор.
НУЖЕН БИТ	В данном контексте требуется битовое выражение.
НУЖНА ПРМ	В этом контексте требуется переменная.
НУЖЕН УКАЗ	В этом контексте требуется переменная типа указатель.
ОПС ВНУТРИ ЦИКЛА	Из-за особенностей реализации оператор DCL нельзя помещать внутри цикла REPEAT.
ПАРАМЕТРЫ	Фактический параметр не соответствует формальному параметру.
ПАР. ЦИКЛА	Переменная оператора DO может быть только скалярной переменной.
ПЕРЕВОД	Константа не может быть преобразована к требуемому типу.
ПЕРЕПОЛНЕНА ТАБЛИЦА*	Не хватает места для таблицы идентификаторов.

ПЛОХОЕ ЗНЧ	Число параметров в INITIAL не совпадает с числом переменных, которые надо инициализировать
ПЛОХИ ПАР.	Неправильный аргумент во встроенной функции или индекс массива, заданный константой, находится вне границ массива.
РЗМ. НЕ=	«Физические» типы в выражении не совпали.
СЛИШКОМ ДЛИННОЕ ВЫРАЖЕНИЕ *	Выражение слишком сложное для компилятора. Упростите программу.
СТРАННО	Предупреждающее сообщение на некоторые конструкции, в которых возможна ошибка, например: X=X=1; (возможно здесь подразумевалось X=X+1)
СТРУКТУРА	Ссылка на переменную не соответствует описанной структуре. (Обычно из-за двусмысленной ссылки на подструктуру).
ТИПЫ НЕ =	Несовместимость типов в операции. Проверьте DCL и правила преобразования типов. Может быть, вследствие неправильного присвоения значений элементам структуры.
ТИП ОТВЕТА	Выражение в операторе RETURN несовместимо с атрибутами в RETURNS соответствующей процедуры.
ОСТАТОК > 0	Показывает, что в результате вычислений выражение типа FIXED BINARY имеет ненулевой масштабный коэффициент (SCALE). Для данной версии — это недопустимо. Если выражение включает деление (X/Y) - заменить на DIVIDE (X,Y,P,0). Эта ошибка не отменяет создание .OBJ.

21.5. Ошибки при третьем проходе компилятора

ВОЗМОЖЕН ВЫХОД БЕЗ ЗНАЧЕНИЯ ИЗ ФУНКЦИИ ...	В указанной процедуре-функции формально можно попасть на последний END без выполнения оператора RETURN.
ДЕЛЕНИЕ НА 0	В программе есть деление на константу, равную нулю.
ЕСТЬ ДРУГОЕ ВНЕШНЕЕ	Есть такая же внешняя переменная, но с другими атрибутами.
НЕДОСТУПНО МЕСТО ЗА ...	Предупреждающее сообщение, что в программе были команды (за RETURN или за GOTO), на

	которые невозможно передать управление. Компилятор выбросил эти команды.
НЕ СДЕЛАНО	Ошибка в промежуточной операции из-за ошибки компилятора.
ОЧЕНЬ ДЛИННОЕ ВЫРАЖЕНИЕ *	Слишком сложное выражение в программе. Упростите. Место в программе можно найти, задав ключ L.
ОШИБКА С РЕГИСТРОМ	Ошибка в промежуточной операции из-за ошибки компилятора. Возможно, были неверные преобразования типов.
ПЕРЕПОЛНЕН ВНУТРЕННИЙ СТЕК	Ошибка в компиляторе или попытка использовать не реализованные функции.
ПЛОХ РАБ.ФАЙЛ	Ошибка в промежуточном файле из-за ошибки компилятора.
ПОВТОРЯЮЩИЙСЯ ИНДЕКС У МЕТКИ	Попались метки с одинаковым индексом.
ПЛОХ РАЗМЕР КОМАНДЫ	Ошибка в промежуточной операции из-за ошибки компилятора.
ПОТЕРЯ ОБЪЕКТА	Ошибка в промежуточной операции из-за ошибки компилятора.

Ошибки на 3 проходе компилятора обычно редко выдаются и часто свидетельствуют о разных непредвиденных случаях или неправильной работе самого компилятора. Желательно сообщать о таких случаях разработчикам компилятора. Однако и некоторые неправильные операторы программы иногда могут дать такую диагностику. Например, компиляция оператора

if 1<x<y then y=x; может выдать:

"ПЕРЕПОЛНЕН ВНУТРЕННИЙ СТЕК".

22. СООБЩЕНИЯ ОБ ОШИБКАХ ПРИ ИСПОЛНЕНИИ

22.1. Фатальные ошибки

Очень часто причиной таких ошибок является выход индекса за границу массива. Для динамического контроля индексов можно задать ключ компиляции "T".

СТЕРТ СПИСОК ФАЙЛОВ	Список активных файлов был разрушен во время выполнения. Попытка закрыть все активные файлы в конце работы потерпела неудачу. Часто из-за выхода индекса за границы диапазона.
ПЕРЕПОЛНЕН СТЕК УСЛОВИЙ	Переполнился стек, где запоминаются точки возврата по ON-операторам, часто из-за многократного прохождения управления через ON-оператор и отсутствия соответствующих операторов REVERT.

22.2. Ошибки, которые можно перехватить ON-операторами

22.2.1. Ошибки с типом 0

ОШИБКА(1) "ПЕРЕВОД" *	Преобразование между типами данных не выполнено.
ОШИБКА(2) "МНОГО ВВОДА/ВЫВОДА" *	Стек ввода/вывода превысил 16 одновременно выполняемых операций. Упростите.
ОШИБКА(3)	Аргумент встроенной функции вне допустимого диапазона.
ОШИБКА(4) "КОНФЛИКТ X"	Файл явно или неявно открыт с одним набором атрибутов, а в дальнейшем используется в операторах, требующих других атрибутов. Значение X может быть ТЕКСТОВЫЙ/НЕ ТЕКСТОВЫЙ ПООЧЕРЕДНЫЙ/ПРЯМОЙ ДЛЯ ВВОДА/ДЛЯ ВЫВОДА ИНДЕКСНЫЙ АТРИБУТ Ошибка может быть в случае, если файл, содержащий код ASCII обрабатывается операторами READ и WRITE, а параметр в FROM или INTO не CHARACTER VARYING.
ОШИБКА(5) "МНОГО ФОРМАТОВ" *	Уровень вложенности форматов превышает 32.

	Упростите.
ОШИБКА(6) * "ОШИБКА В ФОРМАТЕ"	При обработке форматов встретился неправильный символ
ОШИБКА(7) "ВСЯ ПАМЯТЬ ИСЧЕРПАНА" *	Нет свободного места в памяти или забыта переменная ?MEMORY. Не могут выполняться ALLOCATE, OPEN, рекурсии.
ОШИБКА(15) "ФАЙЛ УЖЕ ОТКРЫТ"*	Попытка открыть уже открытый файл. В отличие от стандарта «G» сообщается об ошибке.
ОШИБКА(16) * "ИНДЕКС ВНЕ ГРАНИЦ"	Текущее значении индекса вне границ описания массива (см. описание ключа компиляции T).
ОШИБКА(17) * "СТРОКА ИСЧЕРПАНА"	исчерпан аргумент STRING в операторе GET.
ОШИБКА(19) "ВНЕ ПОДСТРОКИ" * или "СТРОКА>255" *	операнды SUBSTR вне строки или при склейке размер строки превысил 255 байт.
ОШИБКА(35)	Все ошибки, обнаруженные средствами Windows
ОШИБКА(36) " РАЗРУШЕН СТЕК СОПРОЦЕССОРА"	При вычислениях FLOAT в FPU стало более 8 значений или ошибка в командах.
ОШИБКА(37) "ПАМЯТЬ КУЧИ РАЗРУШЕНА" *	В операторе FREE определен адрес памяти вне допустимого диапазона (обычно из-за ссылок на неинициализированный указатель).
ОШИБКА(38) * "ПАМЯТЬ РАЗРУШЕНА"	Динамическая память разрушена. Часто из-за выхода индекса за границу или переполнения стека. Выдается при проверке процедурой ?TEST_MEM. Может быть при отсутствии MAIN в программе.
ОШИБКА(39) "РЕКУРСИВНЫЙ ОБМЕН *	Обнаружен рекурсивный вывод, обычно бывает, если в операторе PUT есть обращение к функции, в теле которой тоже есть оператор PUT.
ОШИБКА(40) "КОМПЛЕКСНАЯ ФУНКЦИЯ НЕ РЕАЛИЗОВАНА"	Данная встроенная функция не реализована для комплексного аргумента.
ОШИБКА(41) 'ЗАПУСК ПОТОКА'	Ошибка при попытке запустить процедуру в параллельном режиме.
ОШИБКА(42) 'МНОГО ПОТОКОВ'	Запущено более 32 процедур в параллельном режиме одновременно.
ОШИБКА(43) 'НЕ НАЙДЕНА ИМПОРТИРУЕМАЯ ПРОЦЕДУРА'	Вероятнее всего, имя подпрограммы в описании указано неверно, например, SetConsoleTitle вместо правильного SetConsoleTitleA

22.2.2. Ошибки типов 1-8

ЦЕЛОЕ ПЕРЕПОЛНЕНИЕ	Десятичная операция сложения или умножения с фиксированной точкой вырабатывает значение, превышающее 15 десятичных цифр или точность. Попытка записать переменную в память с недостаточной точностью. Если установлен ключ Q, то вызывается выходом результата арифметической операции за границы 2**n-1 для FIXED BINARY (n).
ВЕЩ. ПЕРЕПОЛНЕНИЕ(1)	Операции с плавающей точкой дали результат, не представимый в формате IEEE-754.
АНТИПЕРЕПОЛНЕНИЕ (1)	Операции с плавающей точкой дали результат, величина которого мала для представления в формате IEEE-754.
ДЕЛЕНИЕ НА 0 (1)	Десятичное деление или операция MOD столкнулись с делением на 0.
ДЕЛЕНИЕ НА 0 (2)	Деление (или MOD) с плавающей точкой на ноль.
ДЕЛЕНИЕ НА 0 (3)	Целое деление (или MOD) на ноль.
КОНЕЦ ФАЙЛА	Попытка считывания после конца файла
НЕ НАЙДЕН ФАЙЛ	Файл не может быть найден на диске (для ввода) или создан для вывода.
КЛЮЧ(1)	Обнаружен неправильный ключ в операторе вывода.
КЛЮЧ(2)	Обнаружен неправильный ключ в операторе ввода.
КОНЕЦ СТРАНИЦЫ	Зафиксирована ситуация "Конец страницы". (Замечание: если в программе не была описана ситуация ON ENDPAGE, по заполнению текущей страницы работа программы не прерывается).

23. ВЗАИМОДЕЙСТВИЕ С WINDOWS

Работа в операционной среде Windows часто подразумевает:

- Использование процедур (Win API) из динамически подключаемых библиотек (DLL);
- Использование данных типа «ресурсы»;
- Широкое использование графического вывода;

Строго говоря, нет необходимости во всех задачах обязательно использовать перечисленные средства. Например, программа:

```
test:proc main;
put ('Hello world');
end test;
```

выведет сообщение в стандартное окно (консоль).

Для вычислительных задач или задач, выдающих результаты в файл можно не использовать средства Windows, однако для ряда задач это совершенно необходимо.

23.1. Процедуры Windows из динамических библиотек

23.1.1. Описание процедур Win API в программе

Для обращения к процедурам (Win API) Windows их надо описать в программе с атрибутом IMPORT (отсутствует в стандарте). Параметрами таких процедур могут только указатели PTR, данные типа CHAR VAR, FIXED BINARY (31/63) и BIT (32/64). Например:

```
DCL SLEEP ENTRY (FIXED (31)) IMPORT;
SLEEP(10000); // ПАУЗА 10 СЕКУНД
```

23.1.2. Реализация подключения процедур Win API

В данной реализации использование таких процедур затруднено тем, что для Win API допустимы большие и малые буквы в именах, а здесь – только большие. Поэтому пришлось разработать специальный механизм указания имен для LINK-KT. Кроме описания процедур в программе еще необходимо поместить их имена в специальный ассемблерный модуль, составленный и транслируемый по правилам, описанным ниже.

Для большей части процедур из библиотек вроде GDI32.DLL, USER32.DLL, и KERNEL32.DLL, т.е. для наиболее часто используемых процедур, такой модуль уже составлен и включен в PLINK64.EXE.

Вместе с созданием EXE-файла PLINK64.EXE создает и SYM-файл, содержащий имена и адреса переменных. После компиляции этот файл используется обычно для отладки программы с помощью встроенного отладчика. В конце SYM-файла имеется раздел (с названием PE), содержащий имена процедур Win API.

Если Вы описали процедуру с атрибутом IMPORT и она попала в этот раздел SYM-файла - значит, она будет подключена при запуске программы.

Если же процедура с атрибутом IMPORT описана в программе, но отсутствует в SYM-файле – значит, ее нет в PLINK64.EXE и необходимо создать свой ассемблерный модуль, где указать нужные процедуры и библиотеки.

Называться такой модуль должен обязательно WINDOWS.A86.

Транслироваться такой модуль должен директивой:

```
PLINK64 WINDOWS.A86 $+
```

что позволит не переводить малые буквы в большие.

Полученный модуль WINDOWS.OBJ вместе с другими OBJ-файлами формирует EXE-программу, например: PLINK64 TEST,WINDOWS

Следует убедиться, что в SYM-файле появилось имя требуемой процедуры.

23.1.3. Правила составления имен процедур в модуле WINDOWS.A86

Пример:

```
PUBLIC LIB@DLL EQU 20 SHL 16

PUBLIC ADDA EQU LIB@DLL+001
AddA EQU NOT ADDA
PUBLIC ADDW EQU LIB@DLL+002
AddW EQU NOT ADDW
```

В данном примере описаны две функции AddA и AddW из библиотеки LIB.DLL. Фактически имена представляют собой некоторые константы-числа, записанные по правилам, по которым LINK-KT подставит их в EXE-файл.

Для библиотек выделены номера с 20 до 31 (остальные использованы в самом PLINK64.EXE). Номера самих процедур произвольны, рекомендуется располагать их в алфавитном порядке и назначать номера 1,2,3 … Имена процедур указаны для PL/1 (в примере ADDA и ADDW) и для Win API (в примере AddA и AddW). Таким образом, имя ADDA является «псевдонимом» имени AddA, т.е. в PL/1-программе процедуры Win API можно назвать как угодно,

например, по-русски. Отличие имени от «псевдонима» можно использовать, когда в имени есть малые буквы или оно больше 31 символа.

23.1.4. Подключение процедур в Win API при исполнении

В системную библиотеку PL1LIB.L86 добавлены служебные функции, которые могут использовать и прикладные программы.

```
?ЗАГРУЗИТЬ_ИЗ_DLL ENTRY (CHAR (*)VAR, CHAR(*)VAR,PTR);
```

Данная функция динамически загружает указанную библиотеку из системной папки SYSTEM32 и записывает адрес указанной функции в переменную, адрес которой указан, как третий параметр. Например,

```
//=== ИНИЦИАЛИЗАЦИЯ БИБЛИОТЕК DIRECT 3D ===

//---- ДИНАМИЧЕСКИ ЗАГРУЖАЕМ ФУНКЦИЮ ИНИЦИАЛИЗАЦИИ ----

DCL DIRECT3DCREATE9 ENTRY (FIXED (31)) RETURNS (PTR) IMPORT;

?ЗАГРУЗИТЬ_ИЗ_DLL('D3D9.DLL','Direct3DCreate9',ADDR (DIRECT3DCREATE9));

//---- САМО ОБРАЩЕНИЕ К ПРОЦЕДУРЕ СОЗДАНИЯ ИНТЕРФЕЙСА ----

P_ИНТЕРФЕЙС=DIRECT3DCREATE9(D3D_SDK_VERSION);

IF P_ИНТЕРФЕЙС=NULL THEN
   PUT SKIP LIST ('НЕ МОГУ НАЙТИ МЕТОДЫ D3D',STOP);
```

23.2. Использование строковых переменных в Win API

Имеются различия в представлении строк в PL/1 и Win API:

- Строка не имеет байта длины в начале
- Строка оканчивается нулевым байтом

Для преодоления указанных отличий, удобно использовать наложение данных типа CHAR на данные типа CHAR VAR, например:

```
DCL
MOVEFILEA ENTRY (PTR,PTR,BIT (32)) IMPORT ETURNS (FIXED (31)),
S1 CHAR (*) VAR,
S2 CHAR (*) DEF (S1+1),
S3 CHAR (*) VAR,
```

```
S4 CHAR (*) DEF (S3+1);

S1='TEST1.DAT^@';
S3='TEST2.DAT^@';
MOVEFILEA(ADDR (S2),ADDR(S4),'0000000A'B4);
```

В примере окончание строк нулевым кодом обеспечивается символом ^@, а убирание байта длины – наложением со смещением 1. В данном примере можно было и не использовать данные CHAR VAR, однако в реальных программах их использование для строк Win API удобнее, чем просто CHAR.

Однако, если описать ту же процедуру Win API, но не с параметрами PTR, а прямо CHAR (*) или CHAR(*) VAR таких преобразований не требуется.

23.3. Работа с ресурсами

Данные типа «ресурсы» фактически никак не связаны с компилятором PL/1-KT и полностью обеспечиваются работой LINK-KT. Для подключения ресурсов к EXE-файлу, необходимо указать в списке собираемых модулей оттранслированный компилятором «ресурсов» файл с расширением .RES, например: PLINK64 TEST1,TEST2 DATA.RES

Реально, в редакторе связей имеется только простейшая обработка «ресурсов», которая реализована следующим образом:

а) Предполагается что один или несколько файлов «ресурсов» (имеющие расширение .RC) уже оттранслированы стандартным транслятором ресурсов RC.EXE в результате чего, получены те же один или несколько файлов «ресурсов», но уже не в текстовом, а в двоичном виде (они имеют расширение .RES).

б) С помощью стандартного редактора связей ILINK32.EXE (включающего в себя «настоящий» редактор «ресурсов» RLINK32.DLL) собирается общий файл, например, следующей командной строкой:

```
ILINK32 MAIN.OBJ, X0.RES, NUL,,,X1.RES, X2.RES, X3.RES
```

т.е. здесь редактором связей ILINK32.EXE три файла «ресурсов» X1.RES, X2.RES, X3.RES объединяются с объектным модулем MAIN.OBJ в единый выполняемый 32-разрядный EXE-файл, с названием X0.RES.

с) Несмотря на то, что получившийся X0.RES это EXE-файл, а не файл, содержащий только «ресурсы», именно он указывается (всегда последним) при работе LINK-KT, например:

PLINK64 Y1.OBJ, Y2.OBJ X0.RES

Поскольку в качестве MAIN.OBJ используется заглушка, например, написанная на Си «пустая» программа, она всегда будет занимать в X0.RES одно и то же фиксированное место. LINK-KT просто отбрасывает эту заглушку, а все остальное дописывает в получающийся 64-разрядный файл Y1.EXE как секцию «ресурсов» PE-формата, пересчитав все смещения в этой секции для нового файла.

Таким образом, вся работа с «ресурсами» ведется, по существу, старыми и стандартными 32-разрядными средствами. В результате этой работы создается единственная и готовая секция «ресурсов», формально входящая в 32-разрядный выполняемый модуль, у которого нет выполняемой части, и поэтому всегда начинающаяся по абсолютному адресу 600H.

Использованные для этого стандартные старые 32-разрядные средства:

rc.exe, ilink32.exe, lnkdfm50.dll, rcdll.dll, rlink32.dll, rw32core.dll

фактически не являются частью системы программирования на PL/1.

23.4. Работа с реестром Windows

Некоторые параметры компиляции можно задавать не только из команной строки, но и через реестр Windows. Для этого необходимо создать раздел:

«HKEY_CURRENT_CONFIG\SOFTWARE\PL/1».

В этом разделе можно создать подразделы с названиями «A» - «Z» типа REG_DWORD и присваивать им значения 0 или 1 (т.е. ключи компиляции).

Если один и тот же ключ задан и в реестре и в командной строке – он инвертируется. Например, если в реестре задан раздел «Q» со значением 1 и в командной строке задан он же: PLINK64 TEST.PL1 Q то реальное значение ключа будет ноль.

Подраздел с именем «%» и типом REG_SZ может содержать значение в виде одной буквы, которая заменит знак «?» в имени файла в операторе %INCLUDE.

Подраздел «ERR» может содержать текст, добавляемый к каждому сообщению об ошибке.

Подраздел «CTL» задает максимально допустимое число служебных указателей для переменных класса CONTROLLED (по умолчанию 8).

Подраздел «DEFAULT» задает мантиссы по умолчанию для FIXED FLOAT DECIMAL (по умолчанию 00 0F 18 07) т.е. 15 24 7.

24. ОТЛИЧИЯ PL/1-КТ ОТ СТАНДАРТА ПОДМНОЖЕСТВА "G"

24.1. Отсутствующие возможности

- Основные количественные ограничения реализации компилятора связаны с использованием системы команд процессоров x86-64 и проявились в максимально возможном размере для следующих типов данных:
  а) BINARY FIXED — не более 63 бит (от FIXED(1) до FIXED(63))
  б) BIT — не более 64 бит (от BIT(1) до BIT(64))
  в) CHARACTER — не более 254 байт (от CHAR(1) до CHAR(254))
  г) BINARY FLOAT — не более 64 бит (от FLOAT(1) до FLOAT(53))

- Из-за особенностей реализации LINK-KT общий размер EXE-файла (команды + данные) не должен превышать 16 Мбайт. Эти ограничения не касаются объема данных, выделяемых оператором ALLOCATE и размера ресурсов.

- Шаблон PICTURE реализован только как спецификация вывода PUT EDIT;

- Встроенную функцию STRING можно использовать только внутри PUT и GET, аргумент STRING может быть только переменной типа CHAR или CHAR VAR.

- Представление битовых строк с переменной длиной не реализовано.

- Массивы с меняющимися при выполнении программы границами реализованы с ограничениями и требуют специальной подготовки в программе.

- Атрибут описания DEFINED реализован с ограничениями: база должна быть описана раньше, не может иметь атрибута EXT и не может быть элементом массива или не верхним уровнем структуры. Смещение относительно базы задается не атрибутом POSITION, а прямым указанием внутри DEFINED, например: dcl x fixed def(y+100);

- Не реализован оператор типа: A=scalar; , где A - переменная типа массив, а scalar - скалярная переменная, исключение A=0. Можно сравнивать массивы на «равно-не равно» с нулем одним оператором.

- Атрибут структуры LIKE может ссылаться только на описанную выше структуру. Имя после LIKE нужно указывать обязательно полностью.

- Требуется, чтобы третий параметр функции SUBSTR для битовых строк был обязательно константой.

- Отсутствуют некоторые встроенные функции множества "G" (ATANH, COPY и некоторые другие).

24.2. Дополнительные возможности

- Добавлены комплексные числа, но с ограничениями: они применяются только для чисел формата FLOAT, преобразования между комплексными и действительными не проводятся, комплексные константы нужно писать в кавычках.

- Добавлены функции ASCII и RANK.

- Добавлены операторы LEAVE и CONTINUE.

- В операторе ALLOCATE можно явно указывать число выделяемых байт.

- В препроцессорной функции %REPLACE возможно перечисление элементов через запятую, а не единственный оператор.

- С помощью %INCLUDE можно указать свою собственную библиотеку.

- Ключевое слово MAIN главной программы можно указывать без OPTIONS.

- Для текстовых файлов возможен ввод/вывод переменных типа CHARACTER VARYING с помощью функций READ и WRITE.

- Для оператора GET EDIT возможен формат A без указания длины.

- Допускается использование управляющих символов в строковых константах и переменных.

- Допускаются описания типа:

```
DCL X (10) CHAR (10) STATIC INIT ((10) '0123456789');
```

вместо требуемого стандартом «G» описания:

```
DCL X (10) CHAR (10) STATIC INIT ((10) (1) '0123456789');
```

- Введена возможность писать вместо DO открывающую скобку {, а вместо END - закрывающую скобку }, например:
  ```
  IF X=1 THEN {; Y=2; Z=3; };
  ```

- В операторе GO TO можно опускать слово "TO", например GO M;

- В списке вывода оператора PUT можно указывать оператор STOP и SKIP, например:
  ```
  PUT DATA (X,Y,SKIP,STOP);
  ```

- Допускается задавать начальные значения указателей, отличные от NULL, например:
  ```
  DCL X1     FIXED,
  X2         BIT (16) BASED (P),
  P          PTR STATIC INIT (X1);
  ```
  Можно указывать элементы структур:
  ```
  DCL P PTR STATIC INIT (X1.X2.X3);
  ```

- Допускается использовать в качестве начального значения встроенную функцию DIMENSION (обязательно в сокращенном виде - DIM), например:
  ```
  DCL X1 (1:100) FIXED,
  LEN            FIXED STATIC INIT (DIM (X1,1));
  ```

- Встроенные функции LBOUND, HBOUND и DIMENSION с нулевым значением второго параметра возвращают размер элемента типа CHARACTER или CHARACTER VARYING:
  ```
  DCL X1 (1:100) CHAR (45);
  Y=DIM (X1,0); /* присвоит Y значение 45 */
  ```

- Для встроенных функций LBOUND, HBOUND и DIMENSION допускается отсутствие второго параметра (тогда принимается 1).

- Допускается задавать код ответа для Windows в операторе STOP.

- Допускается задавать ключевое слово KEY вместо KEYFROM.

- Допускается опускать ключевое слово ERROR, например вместо оператора SIGNAL ERROR(150); можно писать SIGNAL(150);

- Введен оператор RESIGNAL.

- Допускается ставить однострочные комментарии (действующие до конца строки). Такой комментарий начинается с символов "//", например:
  X+1; // пример однострочного комментария

- Если второй параметр функции преобразования CHARACTER отрицательная константа, считается, что это длина строки ответа, начиная справа (а не слева).

- Допускается использовать в операторе CALL процедуру-функцию, возвращающую FIXED BINARY или BIT.

- Ключевое слово CALL можно опускать.

- Функция ROUND для битовых строк выполняет циклический сдвиг.

- Можно ставить префикс одноразовости "1" перед DO-группой.

- В операторе присваивания допустим список переменных.

- После символьных констант можно указать коэффициент повторения в круглых скобках, например: S='-*-'(10);

- Цикл DO REPEAT; ... END; обозначает бесконечный цикл.

- Цикл DO TO 100; использует «невидимую» переменную и выполнится 100 раз.

- Можно перечислять несколько условий в одном ON-операторе.

- Добавлена встроенная функция TRIM, убирающая пробелы и нулевые коды из начала и конца символьных строк.

- Имеется возможность обнулять все локальные переменные процедуры одним оператором присваивания, указывая в левой части оператора идентификатор этой процедуры.

- Добавлены операторы присваивания для арифметических действий типа X+=1; X-=1; X*=2; X/=2;

- Разрешен оператор DIMENSION в описании массивов.

- Добавлена функция DATE4Y, возвращающая все четыре цифры текущего года.

- Добавлена функция REPLACE, заменяющая фрагменты строк.

- Добавлен квалификатор W в текстовых константах для обозначения кодировки «кириллица Windows».

- Добавлена возможность явного задания числа байт в операторах READ и WRITE.

- Добавлен атрибут IMPORT для процедур Win API.

- После END оператора цикла можно писать переменную цикла или слова из заголовка REPEAT/WHILE. Но тогда переменная цикла не должна быть элементом структуры.

- Добавлена возможность напрямую обращаться к регистрам процессора и писать любые коды в программе с помощью функции UNSPEC.

- Добавлены русские эквиваленты ключевых слов с помощью предопределенного оператора %REPLACE.

- При составлении «сложных» заголовков циклов из нескольких частей, каждый новый заголовок должен писаться через запятую, например:
  DO I=3,5,I+2; … END;
  При этом допускается менять и сам параметр цикла по ходу выполнения цикла, например
  DO I=1 TO 100, J=1 TO 100, K=100 TO 1 BY -1; … END;

24.3. Некоторые замечания по реализации

ON-оператор не освобождает память от переменной, обращение к которой вызвало данную ON-ситуацию. Так, нельзя закрыть файл при обмене с которым возникла ON-ситуация, если это ON ENDFILE.

Реализация процесса рекурсивных процедур такова:

- при входе в стек копируется СТАТИЧЕСКИЙ ФРЕЙМ, содержащий адреса используемых объектов в памяти класса AUTOMATIC;
- при выходе рекурсивная процедура восстанавливает все данные предыдущего экземпляра из стека обратно в СТАТИЧЕСКИЙ ФРЕЙМ.

Если рекурсивная процедура вызывает подпрограммы и обращается к объектам в памяти класса AUTOMATIC, фактически работа ведется с адресами из СТАТИЧЕСКОГО ФРЕЙМА. Если подпрограмма вызывает свою собственную рекурсивную процедуру, возможны непредсказуемые результаты. Кроме этого, непредсказуемый результат будет в случае, если в параметрах рекурсивной процедуры обращаться к внешним переменным по значениям. Такие параметры необходимо заключать в круглые скобки.

Для преобразования DECIMAL во FLOAT используется промежуточное преобразование DECIMAL в CHAR и CHAR во FLOAT. Таким образом, выражение X=1/3; где X - FLOAT будет выполняться медленнее, чем X=1/3e0;

Компилятор проверяет, что формально можно попасть на конец тела процедуры-функции (не выполнив оператор RETURN) и выдает предупреждение, если это так.

При выходе из блока, ограниченного BEGIN; и END; установленные в этом блоке ON-операторы не отменяются.

При задании параметров процедуры константами, для этих констант не создаются рабочие переменные внутри процедуры. Однако это можно сделать, если заключить константы-параметры в круглые скобки.

Не поддерживаются условные операторы для переменных типа BINARY FIXED(31), если при их суммировании или вычитании получаются промежуточные результаты больше чем 2147483647 по абсолютной величине.

Для запоминания ON-операторов выделяется 32-х словный стек. Этот стек

общий для всех блоков. Таким образом, в любой точке программы возможно наличие не более чем 32 открытых ON-операторов, иначе будет выдано сообщение: ПЕРЕПОЛНЕНИЕ СТЕКА УСЛОВИЙ.

В результате работы компилятора получается перемещаемый объектный код в формате OMF (Intel Technical Specification 121748-001) с доработкой, связанной с введением в x86-64 относительной адресации для данных.

LINK-KT позволяет включать в имена внешних процедур и данных кириллические символы, однако случае создания библиотек DLL нет гарантии, что в других системах, использующих эти библиотеки, такие имена допустимы.

25. ОТЛИЧИЯ PL/1-КТ ОТ IBM OS PL/I И "ПОЛНОГО" PL/I (ANSI X3.53)

1. Несколько процедур, BEGIN-блоков и DO-групп не могут оканчиваться единственным END.

2. Процедуры не могут иметь несколько точек входа.

3. Функции должны возвращать скалярные значения.

4. Длина символьной строки в описании RETURNS должна быть целой константой.

5. BEGIN-блок не может иметь атрибута OPTIONS.

6. Набор символов (малые, русские буквы и псевдографические знаки) отличается от полного набора ANSI.

7. Внутри комментария не должно быть комбинации символов /* */ (если не установлен ключ компиляции "H").

8. Префиксы разрешения/запрещения обработки исключительных ситуаций не допускаются.

9. Коэффициент повторения может быть только у строчных констант и пишется сзади (в ANSI PL/I - впереди).

10. Не разрешается использовать неописанные переменные.

11. Длины строковых переменных и структур при описании должны быть целыми константами, исключение - длина строки формального параметра (допустим символ "*").

12. Нижняя и верхняя границы массивов должны быть целыми константами.

13. Не должно быть индексированных меток перед ключевыми словами PROCEDURE и FORMAT, не допускаются любые метки перед DECLARE, THEN и ELSE.

14. Массив, занимающий не непрерывный участок памяти (из-за того, что он часть структуры) не может быть аргументом процедуры.

15. Выражения в IF и DO WHILE должны быть битовыми строками длиной 1.

16. Для выхода из ON-оператора нельзя использовать RETURN (а только LEAVE ON).

17. Не поддерживаются следующие операторы:
    DISPLAY, EVENT, EXIT, FLOW, HALT, ENTER, NOFLOW, ORDER, OTHERWISE, PRIORITY, REENTRANT, RELEASE, REORDER, REPLY, RETCODE, SELECT, SYSTEM, RASK, TRKOFL, UNLOCK, WHEN.
    Оператор EXIT может быть заменен на оператор CALL <имя> STOP;

18. В операторе ALLOCATE не поддерживается опция IN.

19. Не поддерживается атрибут CONNECTED.

20. Не разрешается присваивание оператором BY NAME.

21. В операторах PUT и GET не поддерживается атрибут COPY, оператор GET DATA не допускает точки с запятой и произвольного порядка данных.

22. В операторе PUT опции ALL, FLOW и SNAP не допускаются.

23. Список форматов не может содержать переменные и выражения.

24. В операторе READ опции LOCATE и IGNORE не поддерживаются.

25. В операторе REWRITE не может быть опции FROM.

26. Не поддерживаются не выровненные битовые строки (независимо от атрибутов ALIGNED и UNALIGNED строки всегда имеют тип ALIGNED).

27. Не поддерживаются следующие атрибуты файлов:
    ADDBUFF, BUFFERED, BUFND, BUFNI, BUFOFF, BUFSP, CTL360, D, DB, FETCH, FS, GENKEY, INDEXAREA, INDEXED, KEYLENGTH, KEYLOC, LEAVE, NCP, NOLOCK, NOWRITE, PASSWORD, REREAD, REUSE, SCALARVARYING, SIS, SKIP, SYSIN, SYSPRINT, TOTAL, TP(M/R), VS, VSAM.

28. Оператор DEFAULT не поддерживается.

29. Следующие атрибуты неявно поддерживаются, но явно не могут быть заданы: CONSTANT, MEMBER, NONVARYING, PRECISION, REAL,

STRUCTURE.

30. Следующие атрибуты не поддерживаются:
AREA, BIT VARYING, CONDITION, GENERIC, LOCAL, OFFSET, POSITION (вместо POSITION используется константа со знаком "+").

31. Для переменных класса DEFINED нельзя использовать атрибут ISUB.

32. В операторе DECLARE нельзя использовать опцию REFER.

33. Начальная инициация переменных допустима только для класса памяти STATIC.

34. Коэффициент повторения в операторе INITIAL должен быть целой константой.

35. Значения INITIAL могут быть строковыми или арифметическими константами, а также встроенными функциями NULL и DIM, для указателей (POINTER) могут быть заданы имена переменных и элементов структур.

36. FIXED DECIMAL может иметь ненулевой масштабный коэффициент, но не может иметь отрицательного масштабного коэффициента.

37. Не поддерживаются следующие атрибуты процедур:
ARG, ASSEMBLER, BACKWARD, CALL, COBOL, EXCLUSIVE, FORTRAN, IRREDUCIBLE, NOMAP, NOMAPIN, NOMAPOUT, REDUCIBLE, TRANSIENT, U, WHEN.

38. В шаблонах при выводе не поддерживаются форматы A, E, I, K, R, T, X, Y.

39. Функции MIN и MAX должны иметь только два аргумента.

40. Встроенные функции не могут вернуть не скалярный результат.

41. Не поддерживаются следующие встроенные функции и переменные:
ADD, AFTER, ALL, ANY, ATANH, BEFORE, COMPILETIME, COMPLETION, COMPLEX, CONJ, COPY, COUNT, COUNTER, CURRENTSTORAGE, DATAFIELD, DECAT, DOT, EMPTY, EVERY, HIGH, IMAG, LOW, OFFSET, ONCHAR, ONCOUNT, ONFIELD, , ONSOURSE, PARMSET, PLIRETV, POINTER, POLY, PRECISION, PRIORITY, PROD, REAL, REPEAT, REVERSE, SAMEKEY, SOME, STATUS, STORAGE, SUBTRACT, SUM, VALID.

42. Не поддерживаются следующие псевдопеременные:
COMPLETION, IMAG, ONCHAR, PRIORITY, REAL, STATUS.

43. Операторы должны возвращать скалярный результат.

44. Выражения над массивами не допускаются, однако массивы можно присваивать один в другой, сравнивать на "равно" и "не равно", а также обнулять одним оператором присваивания и сравнивать с нулем.

45. Оператор деления "/" без потери остатка недопустим для переменных FIXED BINARY.

46. Индексы массивов в выражениях должны быть целыми.

47. Процедура может содержать только один список аргументов.

48. Арифметические операторы и встроенные математические функции должны иметь арифметические операнды.

49. Операторы отношения требуют или двух арифметических операндов или операндов одного типа.

26. Список внутренних функций, используемых компилятором

Для реализации программ на языке PL/1 компилятор использует следущие служебные функции и подпрограммы («экстракоды»):

Название функции	№ модуля	описание
?ADDIO	114#	ДОБАВЛЕНИЕ В СВЯЗАННЫЙ СПИСОК ОТКРЫТЫХ ФАЙЛОВ
?ADDRBUFKEY	025#	СЛУЖЕБНЫЙ УКАЗАТЕЛЬ НА БУФЕР ДЛЯ ПЕРЕХВАТА ВЫДАЧИ
ALLOCATE выделение памяти		
?ALLOA	119#	Оператор ALLOCATE с выравниванием
?ALLOC_NZ	119#	Не обнулять выделенную память
?ALLOC_ZR	119#	Обнулять выделенную память
?ALLOP	119#	ВЫДЕЛЕНИЕ ПАМЯТИ
?АДР_ИСКЛ	114#	ОБРАБОТЧИК ИСКЛЮЧЕНИЙ
?BADFM	024#	ВЫДАЧА ОШИБКИ В ФОРМАТЕ ВВОДА-ВЫВОДА
?BDPTR	001#	ОБРАБОТЧИК ИСКЛЮЧЕНИЯ «НЕВЕРНЫЙ УКАЗАТЕЛЬ»
?BEGIN	114#	УКАЗАТЕЛЬ НА НАЧАЛО СВОБОДНОЙ ПАМЯТИ (НАЧАЛО СПИСКА КУЧИ В ALLOCATE)
?BITMAP	OVR#	
ФУНКЦИЯ "INDEX" ДЛЯ:		
?BIX08	113#	BIT (8)
?BIX16	113#	BIT (16)
?BIX32	113#	BIT (32)
?BIX64	113#	BIT (64)
?ВЫВОД_СИМВОЛА	114#	ВЫВОД ОДНОГО СИМВОЛА В КОНСОЛЬНОЕ ОКНО
?CIOOP	005#	ЗАКРЫТЬ ОПЕРАЦИЮ ОБМЕНА
?CLOSE	025#	ОПЕРАЦИЯ ЗАКРЫТИЯ ФАЙЛА
?CMEMRY	114#	ДЛИНА ВСЕХ КОМАНД В ПРОГРАММЕ (КОНСТАНТА)
?CNVER	123#	СИГНАЛ "ОШИБКА ПЕРЕВОДА"
?COMMAND_LINE	114#	КОМАНДНАЯ СТРОКА ПРИ ВЫЗОВЕ ПРОГРАММЫ
РАБОТА С КОМПЛЕКСНЫМИ		
?CPL_1	146#	ОБРАБОТКА ОПЕРАЦИИ 1 ОПЕРАНД КОМПЛЕКСН. 2 – ДЕЙСТВ.
?CPL_2	146#	ОБРАБОТКА ОПЕРАЦИИ 2 ОПЕРАНД КОМПЛЕКСН. 1 – ДЕЙСТВ.
?CPL_F	146#	ОБРАБОТКА ОПЕРАЦИИ ОБА ОПЕРАНДА КОМПЛЕКСНЫЕ
Ф-ЦИЯ "SUBSTR" ТИПА "CHAR"		
?CS2AD	080#	Ф-ЦИЯ "SUBSTR" ТИПА "CHAR" С ДВУМЯ ОПЕРАНДАМИ
?CS3AD	080#	Ф-ЦИЯ "SUBSTR" ТИПА "CHAR" С ТРЕМЯ ОПЕРАНДАМИ
?CT2AD	099#	SUBSTR" ТИПА "CHAR" С ДВУМЯ ОПЕРАНДАМИ И КОНТРОЛЕМ ДЛИНЫ
?CT3AD	099#	SUBSTR" ТИПА "CHAR" С ТРЕМЯ ОПЕРАНДАМИ И КОНТРОЛЕМ ДЛИНЫ
?CTРОКА	114#	ТЕКУЩИЙ НОМЕР СТРОКИ ПРИ ВЫПОЛНЕНИИ ПРОГРАММЫ
?CTRIM	158#	ФУНКЦИЯ TRIM ДЛЯ CHAR НЕ VAR
ФУНКЦИЯ "INDEX" ДЛЯ:		
?CXCC1	107#	ДВУХ ПАР-ОВ ТИПА "CHAR" ИЗ ПАМЯТИ (БЕЗ СТЕКА)
?CXCCM	107#	ДВУХ ПАР-ОВ ТИПА "CHAR" ИЗ ПАМЯТИ (ЧЕРЕЗ СТЕК)
?CXCMS	103#	ПАР-ОВ ТИПА "CHAR" ИЗ ПАМЯТИ И СТЕКА
?CXCVM	102#	ПАР-ОВ ТИПА "CHAR" И "CHAR VAR" ИЗ ПАМЯТИ
?CXSCM	101#	ПАР-ОВ ТИПА "CHAR" ИЗ СТЕКА И ПАМЯТИ
?CXSTS	104#	ПАР-ОВ ТИПА "CHAR" ИЗ СТЕКА
?CXSVM	101#	ПАР-ОВ ТИПА "CHAR VAR" ИЗ СТЕКА И ПАМЯТИ
?CXVCM	105#	ПАР-ОВ ТИПА "CHAR VAR" И "CHAR" ИЗ ПАМЯТИ
?CXVMS	100#	ПАР-ОВ ТИПА "CHAR VAR" ИЗ ПАМЯТИ И СТЕКА

?CXVVM	106#	ПАР-ОВ ТИПА "CHAR VAR" ИЗ ПАМЯТИ
?СЧТ	001#	ОБМЕН С ФАЙЛАМИ И ПРОВЕРКИ (ДАННЫЕ ДЛЯ "STRING" :ДЛИНА И ПРИЗНАК "CHAR VAR", АДРЕС И СЧЕТЧИК СТРОКИ)
?DABSF	076#	Ф-ЦИЯ "ABS" ДЛЯ "DECIMAL"
?DADOP	067#	СЛОЖЕНИЕ "DECIMAL" В СТЕКЕ
?DCEIL	078#	Ф-ЦИЯ "CEIL" ДЛЯ "DECIMAL"
?DCMOP	071#	СРАВНЕНИЕ "DECIMAL" В СТЕКЕ (С ОЧИСТКОЙ СТЕКА)
?DCOMP	071#	СРАВНЕНИЕ "DECIMAL" В СТЕКЕ (БЕЗ ОЧИСТКИ СТЕКА)
?DCRET	074#	ВОЗВРАТ С "DECIMAL"
?DD	114#	РЕЗЕРВИРУЕТ ПАМЯТЬ ПОД КОНСТ. 'XX.XX.XX'
?DDVOP	068#	Ф-ЦИЯ ДЕЛЕНИЯ ДЛЯ "DECIMAL" ИЗ СТЕКА
?DEBUG	120#	
?DEXOP	079#	ВОЗВЕДЕНИЕ В СТЕПЕНЬ ДЛЯ "DECIMAL"
?DFLOR	078#	Ф-ЦИЯ "FLOOR" ДЛЯ "DECIMAL"
?DLDOP	069#	ЗАГРУЗКА "DECIMAL" В СТЕК
?DMAXF	075#	Ф-ЦИЯ "MAX" ДЛЯ "DECIMAL"
?DMINF	075#	Ф-ЦИЯ "MIN" ДЛЯ "DECIMAL"
?DMODF	068#	Ф-ЦИЯ "MOD" ДЛЯ "DECIMAL"
?DMUOP	066#	УМНОЖЕНИЕ "DECIMAL" В СТЕКЕ
?DNGOP	065#	СМЕНА ЗНАКА У "DECIMAL"
?DOONE	115#	ВЫПОЛНЕНИЕ ЦИКЛА СТРОГО ОДИН РАЗ
?DOVER	073#	ПЕРЕПОЛНЕНИЕ ДЛЯ "DECIMAL"
?DROUN	077#	Ф-ЦИЯ "ROUND" ДЛЯ "DECIMAL"
?DSIOP	072#	СИГНАТУРА "DECIMAL" В СТЕКЕ
?DSTOP	070#	ИЗВЛЕЧЕНИЕ "DECIMAL" ИЗ СТЕКА
?DSUOP	064#	ВЫЧИТАНИЕ "DECIMAL" В СТЕКЕ
?DS_PL	114#	СЕГМЕНТ ДАННЫХ DS:
?DT	012#	ПРИЗНАК "GET DATA" ИЛИ "PUT DATA"
?EDITF	018#	ЗАПИСЬ ОДНОГО ФОРМАТА ДЛЯ "EDIT"
?EDTIV	022#	ВВОД ПО "EDIT" "CHAR VAR"
?EDTOB	021#	"OUTPUT EDIT BIT"
?EDTOV	019#	"OUTPUT VAR"("OUTPUT EDIT" ИЗ СТЕКА ПО ФОРМАТУ)
?ENPOP	020#	ВЫВОД ПО ШАБЛОНУ
?ERMSG	114#	СООБЩЕНИЕ И ВЫХОД ИЗ ПРОГРАММЫ
?FA4_L	031#	СЛОЖЕНИЕ ДВУХ "FLOAT" 1 - В СТЕКЕ, 2 - В ПАМЯТИ
?FA4_M	031#	СЛОЖЕНИЕ ДВУХ "FLOAT" В ПАМЯТИ
?FA4_R	031#	СЛОЖЕНИЕ ДВУХ "FLOAT" 2 - В СТЕКЕ, 1 - В ПАМЯТИ
?FA4_S	031#	СЛОЖЕНИЕ ДВУХ "FLOAT" В СТЕКЕ
?FABSF	036#	Ф-ЦИЯ "ABS" ДЛЯ "FLOAT"
?FAF_L	031#	СЛОЖЕНИЕ ДВУХ "FLOAT" 1 - В СТЕКЕ, 2 - В ПАМЯТИ
?FAF_M	031#	СЛОЖЕНИЕ ДВУХ "FLOAT" В ПАМЯТИ
?FAF_R	031#	СЛОЖЕНИЕ ДВУХ "FLOAT" 2 - В СТЕКЕ, 1 - В ПАМЯТИ
?FAF_S	031#	СЛОЖЕНИЕ ДВУХ "FLOAT" В СТЕКЕ
?FC44L	030#	СРАВНЕНИЕ ДВУХ "FLOAT" 1 - В СТЕКЕ, 2 - В ПАМЯТИ
?FC44M	030#	СРАВНЕНИЕ ДВУХ "FLOAT" В ПАМЯТИ
?FC44R	030#	СРАВНЕНИЕ ДВУХ "FLOAT" 2 - В СТЕКЕ, 1 - В ПАМЯТИ
?FC44S	030#	СРАВНЕНИЕ ДВУХ "FLOAT" В СТЕКЕ
?FCEIL	040#	Ф-ЦИЯ "CEIL" ДЛЯ "FLOAT"
?FD4_L	029#	ДЕЛЕНИЕ ДВУХ "FLOAT" 1 - В СТЕКЕ, 2 - В ПАМЯТИ
?FD4_M	029#	ДЕЛЕНИЕ ДВУХ "FLOAT" В ПАМЯТИ
?FD4_R	029#	ДЕЛЕНИЕ ДВУХ "FLOAT" 2 - В СТЕКЕ, 1 - В ПАМЯТИ
?FD4_S	029#	ДЕЛЕНИЕ ДВУХ "FLOAT" В СТЕКЕ
?FDF_L	029#	ДЕЛЕНИЕ ДВУХ "FLOAT" 1 - В СТЕКЕ, 2 - В ПАМЯТИ

?FDF_M	029#	ДЕЛЕНИЕ ДВУХ "FLOAT" В ПАМЯТИ
?FDF_R	029#	ДЕЛЕНИЕ ДВУХ "FLOAT" 2 - В СТЕКЕ, 1 - В ПАМЯТИ
?FDF_S	029#	ДЕЛЕНИЕ ДВУХ "FLOAT" В СТЕКЕ
?FE40M	035#	СИГНАТУРА "FLOAT" В ПАМЯТИ
?FE40S	034#	СИГНАТУРА "FLOAT" В СТЕКЕ
?FEX07	041#	ВОЗВЕДЕНИЕ В СТЕПЕНЬ "FIXED (7)" ПЕРЕМ. "FLOAT"
?FEX15	041#	ВОЗВЕДЕНИЕ В СТЕПЕНЬ "FIXED (15)" ПЕРЕМ. "FLOAT"
?FFLOR	040#	Ф-ЦИЯ "FLOOR" ДЛЯ "FLOAT"
?FFXOP	051#	ВОЗВЕДЕНИЕ В СТЕПЕНЬ "FLOAT" ПЕРЕМ. "FLOAT"
?FM4_L	028#	УМНОЖЕНИЕ ДВУХ "FLOAT" 1 - В СТЕКЕ, 2 - В ПАМЯТИ
?FM4_M	028#	УМНОЖЕНИЕ ДВУХ "FLOAT" В ПАМЯТИ
?FM4_R	028#	УМНОЖЕНИЕ ДВУХ "FLOAT" 2 - В СТЕКЕ, 1 - В ПАМЯТИ
?FM4_S	028#	УМНОЖЕНИЕ ДВУХ "FLOAT" В СТЕКЕ
?FMAXF	038#	Ф-ЦИЯ "MAX" ДЛЯ "FLOAT"
?FMF_L	028#	УМНОЖЕНИЕ ДВУХ "FLOAT" 1 - В СТЕКЕ, 2 - В ПАМЯТИ
?FMF_M	028#	УМНОЖЕНИЕ ДВУХ "FLOAT" В ПАМЯТИ
?FMF_R	028#	УМНОЖЕНИЕ ДВУХ "FLOAT" 2 - В СТЕКЕ, 1 - В ПАМЯТИ
?FMF_S	028#	УМНОЖЕНИЕ ДВУХ "FLOAT" В СТЕКЕ
?FMINF	037#	Ф-ЦИЯ "MIN" ДЛЯ "FLOAT"
?FMODF	039#	Ф-ЦИЯ "MOD" ДЛЯ "FLOAT"
?FN40M	032#	ИЗМЕНЕНИЕ ЗНАКА "FLOAT" В ПАМЯТИ
?FN40S	033#	ИЗМЕНЕНИЕ ЗНАКА "FLOAT" В СТЕКЕ
?FP40	151#	УПАКОВКА "FLOAT" В DX:CX
?FPBIN	002#	РАБОТА С FPB (ДОСТАТЬ FPB ПО АДРЕСУ В СТЕКЕ)
?FPBIO	003#	ОБМЕН ЧЕРЕЗ FPB
?FPBOU	002#	РАБОТА С FPB (ЗАНЕСТИ FPB ПО АДРЕСУ В СТЕКЕ)
?FPEX2	141#	ВОЗВЕДЕНИЕ "FLOAT" В СТЕПЕНЬ ДВОЙКИ
?FPRET	026#	ВОЗВРАТ ИЗ ПРОЦЕДУРЫ С "FLOAT" В СТЕКЕ
?FPSHF	140#	СДВИГ ПЕРЕМЕННОЙ "FLOAT"
?FREOP	119#	ОСВОБОЖДЕНИЕ ПАМЯТИ (ОПЕРАТОР FREE)
?FROUN	149#	Ф-ЦИЯ "ROUND" ДЛЯ "FLOAT"
?FS4_L	027#	ВЫЧИТАНИЕ ДВУХ "FLOAT" 1 - В СТЕКЕ, 2 - В ПАМЯТИ
?FS4_M	027#	ВЫЧИТАНИЕ ДВУХ "FLOAT" В ПАМЯТИ
?FS4_R	027#	ВЫЧИТАНИЕ ДВУХ "FLOAT" 2 - В СТЕКЕ, 1 - В ПАМЯТИ
?FS4_S	027#	ВЫЧИТАНИЕ ДВУХ "FLOAT" В СТЕКЕ
?FSF_L	027#	ВЫЧИТАНИЕ ДВУХ "FLOAT" 1 - В СТЕКЕ, 2 - В ПАМЯТИ
?FSF_M	027#	ВЫЧИТАНИЕ ДВУХ "FLOAT" В ПАМЯТИ
?FSF_R	027#	ВЫЧИТАНИЕ ДВУХ "FLOAT" 2 - В СТЕКЕ, 1 - В ПАМЯТИ
?FSF_S	027#	ВЫЧИТАНИЕ ДВУХ "FLOAT" В СТЕКЕ
?FTRNC	040#	Ф-ЦИЯ "TRUNC" ДЛЯ "FLOAT"
?FU40	150#	РАСПАКОВКА "FLOAT"
?GETKY	025#	ОПЕРАЦИЯ ОБМЕНА (ДОСТАТЬ ТЕКУЩИЙ НОМЕР ЗАПИСИ)
?GETND	019#	ДОСТАТЬ ЦИФРУ ИЗ БУФЕРА МАНТИССЫ ("OUT EDIT" ИЗ СТЕКА ПО ФОРМАТУ)
?GNCPR	007#	ОПЕРАЦИЯ "GET CHAR"
?GNFMT	018#	ДОСТАТЬ ТЕКУЩИЙ ФОРМАТ
?GNVOP	012#	"GET CHAR VAR" ОПЕРАНД
?GNVPR	013#	"GET CHAR VAR"
?HANDLE1	114#	Указатель на открытый стандартный вывод
?HANDLE2	114#	Указатель на открытый стандартный ввод
?IAB07	057#	Ф-ЦИЯ "ABS" ДЛЯ "FIXED (7)"
?IAB15	057#	Ф-ЦИЯ "ABS" ДЛЯ "FIXED (15)"

?IAB31	057#	Ф-ЦИЯ "ABS" ДЛЯ "FIXED (31)"
?IAB63	057#	Ф-ЦИЯ "ABS" ДЛЯ "FIXED (63)"
?IE10N	062#	СИГНАТУРА "FIXED (7)"
?IE20N	063#	СИГНАТУРА "FIXED (15)"
?IE40N	063#	СИГНАТУРА "FIXED (31)" и "FIXED (63)"
?IEXOP	061#	ВОЗВЕДЕНИЕ В СТЕПЕНЬ "FIXED"
?IMAXF	058#	Ф-ЦИЯ "MAX" ДЛЯ "FIXED"
?IMDOP	059#	Ф-ЦИЯ "MOD" ДЛЯ "FIXED"
?IMINF	058#	Ф-ЦИЯ "MIN" ДЛЯ "FIXED"
?INPFM	023#	ЧТЕНИЕ ФОРМАТА ДЛЯ "INPUT" И ОБРАБОТКА УПРАВЛЕНИЯ
?IROUN	060#	Ф-ЦИЯ "ROUND" ДЛЯ "FIXED"
ОПЕРАЦИИ С КЛЮЧАМИ		
?KEYOP	006#	ОПЕРАНД KEY В WRITE/READ ДЛЯ 16 РАЗР.
?KEYP4	006#	ОПЕРАНД KEY В WRITE/READ ДЛЯ 32 РАЗР.
?KEYT4	006#	ОПЕРАНД KEYTO ДЛЯ 32 РАЗР. ВЕРСИИ PL/1
?KEYTO	006#	ОПЕРАНД KEYTO ДЛЯ 16 РАЗР. ВЕРСИИ PL/1
?LNFIL	004#	ДЛИНА ФАЙЛА
?LN_FL	004#	ДЛИНА ИМЕНИ ПРИ ОТКРЫТИИ ФАЙЛА
?MAX_MEMORY	114#	МАКСИМАЛЬНО ДОПУСТИМЫЙ АДРЕС ПАМЯТИ
?MEMRY	114#	СЛУЖЕБНАЯ ПЕРЕМЕННАЯ (СОДЕРЖИТ РАЗМЕР ПРОГРАММЫ)
?OFCOP	118#	СБРОС ON-ОПЕРАТОРА ПРИ ВЫХОДЕ ИЗ ПРОЦЕДУРЫ
?OIOOP	004#	ОТКРЫТИЕ ФАЙЛА-ОПЕРАНДА
?OIOPR	004#	ОТКРЫТИЕ ФАЙЛА
?ONCOP	121#	УСТАНОВКА ON-ОПЕРАТОРА
?ONCPC	121#	ПРОВЕРКА НА УСТАНОВЛЕННЫЙ ON
?OPNFIL	025#	ВНУТРЕННЯЯ ОПЕРАЦИЯ ОТКРЫТИЯ ФАЙЛА
?OUTFM	017#	ЧТЕНИЕ И ВЫПОЛНЕНИЕ ОЧЕРЕДНОГО ФОРМАТА
?OVERF	125#	СИГНАЛ "ПЕРЕПОЛНЕНИЕ"
?PAGOP	010#	ОПЕРАЦИЯ ПЕРЕВОДА СТРАНИЦЫ
?PI	056#	ПИ С ОДИНАРНОЙ ТОЧНОСТЬЮ
?PI_2	056#	ПИ С ДВОЙНОЙ ТОЧНОСТЬЮ
?PNBOP	008#	"PUT VAR" ОПЕРАНД "BIT"
?PNC	017#	ЧТЕНИЕ ФОРМАТА ДЛЯ "OUTPUT" (ДЛЯ ОДНОГО СИМВОЛА)
?PNCOP	008#	"PUT VAR" ОПЕРАНД "CHAR"
?PNCPR	009#	"PUT CHAR"
?PNVOP	008#	"PUT VAR" ОПЕРАНД "CHAR VAR" ДЛЯ "WRITE"
?PPUSH	114#	ПЕРЕСЫЛКА ПАРАМЕТРОВ В СТЕК ДЛЯ ПРОЦЕДУРЫ WINDOWS
?PROV	055#	ПРОВЕРКА ПЕРЕПОЛНЕНИЯ ПРИ РАСЧЕТЕ ТРАНСЦЕНД. Ф-ЦИЙ
?PTR_NAMES	114#	УКАЗАТЕЛЬ НА БУФЕР ИМЕН ПОСЛЕДНИХ ВЫЗВАННЫХ ПРОЦЕДУР
ФУНКЦИИ ПРЕОБРАЗОВАНИЙ:		
?QB08C	126#	"BIT (8)" В "CHAR"
?QB16C	126#	"BIT(16)" В "CHAR
?QB32C	126#	"BIT (32)" В "CHAR"
?QB64C	126#	"BIT (64)" В "CHAR"
?QCB08	127#	"CHAR" В "BIT (8)"
?QCB16	127#	"CHAR" В "BIT (16)"
?QCB32	127#	"CHAR" В "BIT (32)"
?QCB64	127#	"CHAR" В "BIT (64)"
?QCCO1	108#	ПРЕОБРАЗОВАНИЕ СТРОКИ К ЗАДАННОЙ ДЛИНЕ ВЫРАВНИВАНИЕ СПРАВА
?QCCOP	108#	ПРЕОБРАЗОВАНИЕ СТРОКИ К ЗАДАННОЙ ДЛИНЕ
?QCDOP	128#	"CHAR" В "DECIMAL"
?QCFOP	144#	"CHAR" В "FLOAT"
?QCIOP	129#	"CHAR" В "FIXED BINARY"

?QDCOP	130#	"DECIMAL" В "CHAR"
?QDDOP	131#	"DECIMAL" К ЗАДАННОЙ ТОЧНОСТИ
?QDDSL	132#	ВЫРАВНИВАНИЕ "DECIMAL" ВЛЕВО
?QDDSR	133#	ВЫРАВНИВАНИЕ "DECIMAL" ВПРАВО
?QDI07	134#	"DECIMAL" В "FIXED BINARY (7)"
?QDI15	134#	"DECIMAL" В "FIXED BINARY (15)"
?QDI31	134#	"DECIMAL" В "FIXED BINARY (31)"
?QDI63	133#	"DECIMAL" В "FIXED BINARY (63)"
?QFCMS	147#	"FLOAT" В "CHAR" ИЗ ПАМЯТИ В СТЕК
?QFCMW	143#	ЗАГЛУШКИ ДЛЯ "FLOAT" ДВОЙНОЙ ТОЧНОСТИ
?QFCSS	148#	"FLOAT" В "CHAR" И СТЕКЕ
?QFF12	142#	"FLOAT (24)" В "FLOAT(53)"
?QFF21	142#	"FLOAT (53)" В "FLOAT(24)"
?QFI_3	142#	"FLOAT" В "FIXED (63)
?QFI_1	146#	"FLOAT" В "FIXED (31)
?QFI_5	146#	"FLOAT" В "FIXED (15)
?QFI_7	146#	"FLOAT" В "FIXED (7)
?QI07B	135#	"FIXED BINARY (7)" В "BIT"
?QI07D	137#	"FIXED BINARY (7)" В "DECIMAL"
?QI0_F	145#	"FIXED BINARY (7)" В "FLOAT"
?QI15B	135#	"FIXED BINARY(15)" В "BIT"
?QI15D	137#	"FIXED BINARY (15)" В "DECIMAL"
?QI1_F	145#	"FIXED BINARY (15)" В "FLOAT"
?QI31B	135#	"FIXED BINARY (31)" В "BIT"
?QI63B	135#	"FIXED BINARY (63)" В "BIT"
?QI31D	137#	"FIXED BINARY (31)" В "DECIMAL"
?QI63D	137#	"FIXED BINARY (63)" В "DECIMAL"
?QI3_F	145#	"FIXED BINARY (31)" В "FLOAT"
?QI6_F	145#	"FIXED BINARY (63)" В "FLOAT"
?QIC31	136#	"FIXED BINARY (31)" В "CHAR"
?QIC63	136#	"FIXED BINARY (63)" В "CHAR"
?QICOP	136#	"FIXED BNARY(15)" В "CHAR"
?QIOOP	002#	РАБОТА С FPB (ЗАПИСЬ ТЕКУЩЕГО FPB)
?RDBUFF	025#	ОПЕРАЦИИ ОБМЕНА (ЧИТАТЬ БУФЕР)
?RDBYTE	025#	ОПЕРАЦИИ ОБМЕНА (ЧИТАТЬ БАЙТ)
?RECLST	114#	СПИСОК ЗАПОМНЕННЫЙ РЕКУРСИЕЙ (НАЧАЛО СПИСКА)
?RECOV	118#	ВЫХОД ИЗ РЕКУРСИВНОЙ ПРОЦЕДУРЫ
?REVOP	121#	ОТМЕНА ON-ОПЕРАТОРА
?RET	147#	ОБРАБОТКА КОНСТАНТ ДЛЯ ДИНАМИЧЕСКИХ МАССИВОВ
?RNIOP	014#	ОПЕРАЦИЯ "READ"
?RSBLK	118#	ВОССТАНОВИТЬ БЛОК РЕКУРСИВНОЙ ПРОЦЕДУРЫ
?SACVM	090#	СКЛЕЙКА "CHAR" И "CHAR VAR"
?SASVM	089#	ЗАПИСЬ "CHAR VAR" ИЗ СТЕКА В ПАМЯТЬ
?SAVVM	090#	СКЛЕЙКА "CHAR VAR" И "CHAR VAR"
СРАВНЕНИЕ:		
?SCCCM	088#	ДВУХ "CHAR" В ПАМЯТИ
?SCCMS	087#	"CHAR" В ПАМЯТИ И СТЕКЕ
?SCCVM	084#	"CHAR" И "CHAR VAR" В ПАМЯТИ
?SCSCM	086#	"CHAR" В СТЕКЕ И ПАМЯТИ
?SCSTS	083#	"CHAR" В СТЕКЕ С ОЧИСТКОЙ СТЕКА
?SCSVM	086#	"CHAR VAR" В СТЕКЕ И ПАМЯТИ
?SCVCM	085#	"CHAR VAR" И "CHAR" В ПАМЯТИ
?SCVMS	081#	"CHAR VAR" В ПАМЯТИ И СТЕКЕ
?SCVVM	082#	ДВУХ "CHAR VAR" В ПАМЯТИ

?SERV1	114#	Запись имен процедур при ключе R
?SERV2	114#	Проверка выхода индекса за границы массива при ключе T
?SERV3	114#	Проверка выхода индекса за границы массива при ключе T
?SETKY	025#	ОПЕРАЦИЯ ОБМЕНА (УСТАНОВИТЬ ТЕКУЩИЙ НОМЕР ЗАПИСИ)
?SG87	114#	ПРОВЕРКА И ИНИЦИАЦИЯ FPU
?SIG	120#	ОБРАБОТКА СИГНАЛА С СОКРАЩЕННЫМИ ПАРАМЕТРАМИ
?SIGNAL	120#	ОБРАБОТКА СИГНАЛА ПРЕРЫВАНИЯ (СИГНАЛ С ОПЕРАНДАМИ В ПАМЯТИ)
?SIGOP	120#	ОБРАБОТКА СИГНАЛА ПРЕРЫВАНИЯ (СИГНАЛ С ОПЕРАНДАМИ-РЕГИСТРАМИ)
?SIOOP	001#	ОТКРЫТИЕ ОПЕРАЦИИ ОБМЕНА
?SJSCM	091#	ПРИКЛЕИВАНИЕ К "CHAR" В СТЕКЕ "CHAR"
?SJSTS	092#	СКЛЕИВАНИЕ ДВУХ СТРОК В СТЕКЕ
?SJSVM	091#	ПРИКЛЕИВАНИЕ К "CHAR" В СТЕКЕ "CHAR VAR"
?SKPOP	011#	ОПЕРАЦИЯ "SKIP" ДЛЯ ОПЕРАНДА
?SKPPR	011#	ПРОГОН ИЛИ ПРОПУСК СТРОК ("SKIP")
?SLCT2	093#	ЗАПИСЬ СТРОКИ "CHAR" ИЗ СЕГМЕНТА КОМАНД
?SLCTS	093#	ЗАПИСЬ СТРОКИ "CHAR" В СТЕК
?SLVTS	093#	ЗАПИСЬ СТРОКИ "CHAR VAR" В СТЕК
?SMCCM	097#	ПЕРЕПИСЬ "CHAR" В "CHAR"
?SMCVM	096#	ПЕРЕПИСЬ "CHAR" В "CHAR VAR"
?SMVCM	097#	ПЕРЕПИСЬ "CHAR VAR" В "CHAR"
?SMVVM	096#	ПЕРЕПИСЬ "CHAR VAR" В"CHAR VAR"
?SP	114#	АДРЕС ПОСЛЕДНЕГО БЕЗОШИБОЧНОГО МЕСТА
?SSCFS	094#	ИЗВЛЕЧЕНИЕ "CHAR" ИЗ СТЕКА
?SSVFS	095#	ИЗВЛЕЧЕНИЕ "CHAR VAR" ИЗ СТЕКА
?STACK	114#	ВЕРШИНА СТЕКА
?START	114#	СТАНДАРТНОЕ НАЧАЛО ПРОГРАММЫ
?STOPX	114#	СТАНДАРТНОЕ ОКОНЧАНИЕ ПРОГРАММЫ
?STRNG	001#	ОБМЕН СО СТРОКОЙ STRING
?STVRB	114#	ПРИСВАИВАНИЕ "ENTRY"В "FUNCTION VARIABLE"
?SUBIO	114#	ВЫБРАСЫВАНИЕ ИЗ СВЯЗАННОГО СПИСКА ОТКРЫТЫХ ФАЙЛОВ
?SVBLK	117#	СОХРАНИТЬ СОДЕРЖИМОЕ ДАННЫХ РЕКУРСИВНОЙ ПРОЦЕДУРЫ
?SYSIN	001#	ПРОВЕРКА СОВМЕСТИМОСТИ ОПЕРАЦИЙ (ВВОД)
?SYSPR	001#	ПРОВЕРКА СОВМЕСТИМОСТИ ОПЕРАЦИЙ (ВЫВОД)
?TEST_MEM	119#	Проверка целостности памяти
?TEST_MEM_KVANT	119#	Частичная проверка целостности памяти
?TST387	152#	ПРОВЕРКА НА НАЛИЧИЕ СОПРОЦЕССОРА 80387
?UADOP	147#	СЛОЖЕНИЕ ПРИ ПЕРЕВОДЕ "FLOAT"В "CHAR"
?UDV10	147#	ДЕЛЕНИЕ НА 10 ПРИ ПЕРЕВОДЕ "FLOAT"В "CHAR"
?UML10	147#	УМНОЖЕНИЕ НА 10 ПРИ ПЕРЕВОДЕ "FLOAT"В "CHAR"
?UNDER	125#	СИГНАЛ «АНТИПЕРЕПОЛНЕНИЕ» - ИСЧЕЗНОВЕНИЕ ПОРЯДКА
?USLOP	147#	СДВИГ МАНТИССЫ ВЛЕВО ПРИ ПЕРЕВОДЕ "FLOAT" В "CHAR"
?VCRET	098#	ВОЗВРАТ ФУНКЦИИ СО СТРОКОЙ
?VEROP	112#	Ф-ЦИЯ "VERIFY"
?VS2AD	080#	Ф-ЦИЯ "SUBSTR CHAR VAR" С ДВУМЯ ОПЕРАТОРАМИ
?VS3AD	080#	Ф-ЦИЯ "SUBSTR CHAR VAR" С ТРЕМЯ ОПЕРАТОРАМИ
?VT2AD	099#	Ф-ЦИЯ "SUBSTRCHAR" С ДВУМЯ ОПЕРАТОРАМИ
?VT3AD	099#	Ф-ЦИЯ "SUBSTR CHAR" С ТРЕМЯ ОПЕРАТОРАМИ
?VTRIM	158#	ФУНКЦИЯ TRIM ДЛЯ CHAR VAR
?WNIOP	015#	ОПЕРАЦИЯ "WRITE"
?WRBUFF	025#	ОПЕРАЦИЯ ОБМЕНА (ПИСАТЬ БУФЕР)
?WRBYTE	025#	ОПЕРАЦИЯ ОБМЕНА (ПИСАТЬ БАЙТ)
?WRCHR	016#	ВЫВОД СИМВОЛА С ПОДСЧЕТОМ
?XL2OP	109#	"TRANCLATE" С ДВУМЯ ОПЕРАНДАМИ

?XL3OP	110#	"TRANCLATE" С ТРЕМЯ ОПЕРАНДАМИ
?ZEROD	124#	СИГНАЛ "ДЕЛЕНИЕ НА 0"
?ИСКЛ	114#	РЕАКЦИЯ НА ИСКЛЮЧЕНИЕ В PL/1
?ИМЕНА	114#	Имена последних вызванных процедур
?ИМЯ11	114#	Имя первого файла из командной строки
ФУНКЦИИ:		
ACOS	050#	"ACOS"
ALLWDS	138#	ВЫДЕЛЕНИЕ ЗАДАННОГО ЧИСЛА СЛОВ
ASIN	049#	"ASIN"
ATAN	048#	"ATAN"
ATAN2	048#	"ATAN2"
ATAND	048#	"ATAND"
ATAND2	048#	"ATAND2"
COLLATE	111#	"COLLATE"
COS	044#	"COS"
COSD	044#	"COSD"
COSSIN	157#	ОДНОВРЕМЕННО СИНУС И КОСИНУС ОДНОЙ ОПЕРАЦИЕЙ
COSDSIND	157#	ОДНОВРЕМЕННО СИНУС И КОСИНУС ОДНОЙ ОПЕРАЦИЕЙ
COSH	053#	"COSH"
DATE	116#	ВЫДАЧА ТЕКУЩЕЙ ДАТЫ
ERF	155#	"ERF"
ERFC	155#	"ERFC"
EXP	045#	"EXP"
FILEOPEN	122#	"FILEOPEN"
FONT1	153#	ВСТРОЕННЫЙ ШРИФТ
FONT2	154#	ВСТРОЕННЫЙ ШРИФТ
FONT3	155#	ВСТРОЕННЫЙ ШРИФТ
FONT4	156#	ВСТРОЕННЫЙ ШРИФТ
FTC	139#	ПРЕОБРАЗОВАНИЕ "FLOAT" В "CHAR (17) VAR" БЕЗ Е
GAMMA	156#	"GAMMA"
GETDAY	116#	ЧТЕНИЕ ДНЯ НЕДЕЛИ
LINENO	122#	"LINENO"
LOG	046#	"LOG"
LOG10	046#	"LOG10"
LOG2	046#	"LOG2"
MAXWDS	138#	РАБОТА С ДИНАМИЧЕСКОЙ ПАМЯТЬЮ (МАКСИМАЛЬНОЕ ЧИСЛО СЛОВ)
ONCODE	122#	"ONCODE"
ONFILE	122#	"ONFILE"
ONKEY	122#	"ONKEY"
ONLOC	122#	"ONLOC"
ONTERM	122#	"ONTERM"
PAGENO	122#	"PAGENO"
RANDOM	157#	"RANDOM"
SIN	043#	"SIN"
SIND	043#	"SIND"
SINH	052#	"SINH"
SQRT	042#	"SQRT"
STKSIZ	138#	РАБОТА С ДИНАМИЧЕСКОЙ ПАМЯТЬЮ (РАЗМЕР СТЕКА)
TALLY	104#	"TALLY"
TAN	047#	"TAN"
TAND	047#	"TAND"
TANH	054#	"TANH"
TIME	116#	ВЫДАЧА ВРЕМЕНИ
TOTWDS	138#	РАБОТА С ДИНАМИЧЕСКОЙ ПАМЯТЬЮ (ОБЩЕЕ ЧИСЛО СЛОВ)

27. РЕДАКТОР СВЯЗЕЙ LINK-KT

27.1. Общие понятия

LINK-KT это редактор связей, переводящий перемещаемый объектный код в EXE-файл, который в дальнейшем будет выполняться под управлением Windows. Файлы объектного кода могут быть получены в результате работы трансляторов RASM-KT и PL/1-KT, которые выдают перемещаемый код в формате объектного модуля фирмы INTEL (Intel Technical Specification 121748-001), но с доработкой для x86-64).

В данной реализации LINK-KT работает в два этапа:

- Сначала создается EXE-файл в стиле MS DOS, но с 64-разрядной адресацией;
- Затем этот EXE-файл переводится в формат PE, который и будет выполняться в Windows, причем для поиска процедур (Win API) Windows используется SYM-файл, созданный на первом этапе.

27.2. Параметры, влияющие на создание PE-файла

При создании PE-файла LINK-KT может сделать его по правилам драйвера для WINDOWS, если в командной строке указать ключ «/sys».

При создании PE-файла LINK-KT может сделать его без стандартного окна выдачи (консоли), если в командной строке указать ключ «/graph».

При создании PE-файла LINK-KT может сделать динамически загружаемую библиотеку, если в командной строке указать ключ «/dll».

При создании PE-файла LINK-KT может сделать дополнительный файл для режима расширенной отладки, если в командной строке указать ключ «/deb».

При создании PE-файла LINK-KT может добавить в него файл ресурсов, если указать этот файл в конце командной строки с расширением «.RES». Однако, такой файл ресурсов должен сначала пройти обработку (см. раздел 23.3).

27.3. Входные и выходные данные процесса редактирования

LINK-KT различает 2 типа исходных файлов:

- OBJ - это объектный модуль с расширением .OBJ, который получается после работы компилятора PL/1-KT и ассемблера RASM-KT.
- L86 - файл библиотеки объектных модулей с расширением .L86, состоит из нескольких файлов OBJ, записанных в библиотеку редактором LIB-KT.

LINK-KT создает 4 файла:

- выполняемый (.EXE) - готовый исполняемый файл формата PE, содержащий всю необходимую информацию для Windows.
- файл таблицы имен (.SYM) - список меток и переменных из объектных модулей с атрибутами и адресами (нужен для встроенного отладчика);
- план-файл (.MAP - содержит информацию о загрузке выполняемого файла;
- файл с атрибутами переменных в терминах PL/1-программы для расширенной отладки (.TRF).

LINK-KT выводит все неопределенные им переменные на экран. Неопределенными считаются объекты, на которые есть ссылки, но которые не описаны ни в одном из модулей. Однако можно включить режим READ, при котором редактор попытается найти на диске файлы с именами, как у неопределенных объектов и расширением .OBJ.

После окончания работы выдается размер каждой секции выполняемого файла.

При работе на первом этапе для создания EXE-файла в стиле MS DOS LINK-KT оперирует объектами, перечисленными ниже:

- СЕГМЕНТ - блок команд или данных не более 16 Мбайт, это наименьший неделимый элемент с которым работает LINK-KT.
- ИМЯ СЕГМЕНТА - идентификатор по правилам RASM-KT (не более 40 знаков). LINK-KT объединяет все сегменты с одним именем из разных модулей.
- ИМЯ КЛАССА - идентификатор по правилам RASM-KT (не более 40 знаков). LINK-KT использует класс для размещения сегментов в текущей секции.
- ТИП ВЫРАВНИВАНИЯ - указатель типа границы начала сегмента. Может быть байт, слово, параграф (16 байт) или специальное выравнивание для

IA-32.

- ТИП ОБЪЕДИНЕНИЯ - определяет как LINK-KT может объединять части сегмента с одним именем из разных модулей. Обычно используют два типа: общий (COMMON) и смежный (PUBLIC).
- СЕКЦИЯ - логический блок выполняемого файла (может быть до 8). Секция может быть любой длины до 8 Мбайт, но общая длина всех секций не больше 8 Мбайт. LINK-KT может обработать 8 секций с именами: CODE, DATA, EXTRA, STACK, X1, X2, X3 и X4.
- ГРУППА - набор сегментов с общей длиной не более 16 Мбайт и адресуемых через один сегментный регистр

Если программа написана на языке PL/1, все эти параметры выбираются автоматически, для ассемблера их надо назначать самому.

27.4. Вызов LINK-KT

Вызов осуществляется директивой:

PLINK64 [<имя>=] <имя1> [, <имя2>, ..., <имяN>]

Если <имя> указано, то создается выполняемый файл <имя>.EXE, иначе файл будет называться <имя1>.EXE, например:

```
plink64 myfile=part1, part2, part3
```

27.5. Процесс редактирования

Редактирование связей при создании промежуточного EXE-файла в стиле MS DOS включает два этапа или прохода - сборка сегментов из модулей и размещение их в выполняемом файле.

На первом этапе LINK-KT выбирает сегменты с одинаковыми именами из файлов и объединяет их, учитывая атрибуты размещения. При необходимости, сегменты размещаются с начала слова или параграфа (16 байт) или в случае задания атрибута 32ALIGN более сложным образом.

На втором этапе LINK-KT объединяет сегменты в группы, учитывая тип выравнивания для определения промежутков.

После этого, LINK-KT помещает каждый сегмент в одну из 8 секций, выполняя следующие правила:

- сегменты, входящие в группу CGROUP помещаются в секцию команд;
- сегменты, входящие в группу DGROUP помещаются в секцию данных;
- остальные сегменты размещаются в соответствии со своим именем класса;

- не опознанные в 3-х первых случаях сегменты не попадают в EXE-файл.

27.6. Параметры LINK-KT

При вызове LINK-KT можно задавать различные параметры, влияющие на процесс редактирования. Имя параметра можно сокращать. Параметры могут ставиться после каждого имени файла, заключенные в квадратные скобки. Если параметр имеет под параметр, он также заключается в скобки, например:

plink64 test1[map], test2[nolocals]

Параметры могут быть для всего LINK-KT или для отдельных файлов.

27.7. Управление содержимым секций

Можно указать имя секции и перечислить какие сегменты и группы должны в нее входить, изменяя при этом правила умолчания. Примеры:

[code [group[name1,name2]] - группы name1 и name2 помещаются в секцию CODE.

[extra[segment[a1, a2]] - сегменты a1, a2 и a3 помещаются в секцию EXTRA.

Сокращения названий секций:

- CODE - C
- DATA - D
- STACK - ST
- EXTRA - E

27.8. Управление размером секций

Размер каждой секции определяется суммарным размером входящих в нее сегментов. Однако можно указать максимально возможный размер секции или задать дополнительный размер.

Параметр ADDITIONAL или AD выделяет добавочную память в конец секции. Добавочная память указывается в параграфах (16 байт), обычно это нужно для создания своих буферов, например: plink64 test [data [add [1000]]] к концу секции данных добавится 1000H*16 байт памяти.

Параметр MAXIMUM или M задает максимальный возможный размер памяти для секции. Память задается в параграфах, при этом секция не может превышать указанный размер, например: pink64 test [data [m [2000]]]

размер секции DATA не должен превышать 2000H*16 байт.

Параметр ORIGIN или O задает фиксированный размер памяти для секции. Память задается в параграфах, при этом секция будет занимать указанный размер, например: plink64 test [data [o [2000]]]
размер секции DATA в файле .EXE будет 2000H*16 байт.

27.9. Управление включением данных

Параметр FILL или F указывает, что нужно обнулить и включить начальные данные в конец секции выполняемого файла.

Параметр NOFILL или NOF указывает, что не нужно включать начальные данные в конец секции выполняемого файла.

27.10.Управление генерацией стандартного начала .EXE

Параметр NOPREFIX или NOP - не генерировать префиксный код в начале выполняемого файла, такой код обязательно нужен для программ на PL/1. Код обычно отменяется для программ, написанных на ассемблере.

27.11.Управление поиском ненайденных модулей

Параметр READ или R - все ненайденные модули будут искаться на текущем диске (или на диске, определенным параметром $O) как файлы с расширением .OBJ. Например, вместо PLINK64 TEST1,TEST2 можно указать PLINK64 TEST1 [R] и при этом редактор будет сам искать файл TEST2.OBJ.

27.12.Создание план-файла

Параметр MAP или M - указывает, что нужно создать план-карту загрузки выполняемого файла в MAP-файле. Здесь записывается информация обо всех группах и сегментах. По умолчанию такой файл не создается.

Параметр имеет 5 подпараметров:

- OBJMAP / NOOBJMAP - записывать или не записывать информацию об объектных файлах (.OBJ)
- L86MAP / NOL86MAP - записывать или не записывать информацию о библиотечных файлах (.L86)
- ALL - заносить всю информацию в MAP-файл

Пример: plink64 f1 [map [all]], s, gr.l86 [s,map [nol86map]]

27.13.Управление содержимым файла имен

Для работы встроенного отладчика может быть создан файл имен .SYM, содержащий имена меток и переменных и их адреса. По умолчанию такой файл создается, генерацию можно отменить директивой $Sz (см. ниже).

- Параметр LIBSYMS или LI указывает, что нужно включать идентификаторы из библиотечных файлов в файл таблицы идентификаторов (.SYM).
- Параметр NOLIBSYMS или NOLI отменяет включение идентификаторов из библиотечных файлов в файл таблицы идентификаторов (.SYM). Этот параметр действует по умолчанию
- Параметр LOCALS или LO указывает, что нужно включать локальные идентификаторы в файл таблицы идентификаторов (.SYM). Этот параметр действует по умолчанию.
- Параметр NOLOCALS или NOLO отменяет включение локальных идентификаторов в файл таблицы идентификаторов (.SYM).

Используя данные параметры, можно включить в файл .SYM имена только интересующих Вас подпрограмм, а не всех модулей.

27.14.Управление включением библиотечных модулей

Параметр SEARCH или S показывает, что нужно включать при редактировании в выполняемый файл только те модули из библиотеки, на которые есть ссылки, если этот параметр не указан, все файлы из библиотеки будут включены в выполняемый файл. Этот параметр автоматически устанавливается для всех библиотек, вызываемых по умолчанию (в том числе для PL1LIB.L86).

27.15.Задание командной строке в файле

Параметр INPUT или I указывает, что всю командую строку для вызова LINK-KT (кроме самого слова "plink64") можно записать в файл и вызывать по имени этого файла, можно указывать и комментарии по правилам RASM-KT, например в файле TEST.INP записаны строки:

```
memtest=test1, test2,test3,
;---- служебные программы ----------
```

iolib.l86[s],math.l82[s],test4,test5[local]

вызов LINK-KT производится в виде: plink64 test [i]
Замечание: *если расширение не указано, ищется файл с расширением .INP*

Параметр ECHO или EC указывает, что если вызов производился с помощью параметра INPUT, то на экран будет выдано содержание файла с командной строкой, например: plink64 test [i,echo]

27.16.Управление генерацией файлов

Можно указать место откуда брать исходные или куда записывать полученные файлы. Параметр задается в виде $td, где t - тип файла, а d – буква диска, типы файлов следующие:

- C - выполняемый файл (.EXE)
- L - библиотечный файл (.L86)
- M - план-файл (.MAP)
- O - объектный файл (.OBJ)
- S - таблица символов (.SYM).

Буква диска - буквы от A до P или специальные:
* - текущий диск, X - экран, Y - принтер, Z - отмена генерации файла.
По умолчанию задано $MZ, $S*. Пример: plink64 part1 [$sz],p[$od]

27.17.Ошибки, выдаваемые при работе LINK-KT

БОЛЕЕ ОДНОЙ ГЛАВНОЙ ПРОГРАММЫ	в нескольких процедурах стоит OPTIONS (MAIN).
В ФОРМАТЕ OMF8086/87 ОШИБКА N	найдена ошибка в файле OBJ, например он затерт. Список ошибок OMF приведен в следующей таблице
ГРУППА БОЛЬШЕ 16M	в одной группе больше 16 Мбайт.
ЗАДАНО СЛИШКОМ МНОГО МОДУЛЕЙ ИЗ БИБЛИОТЕКИ	ссылок на библиотечные модули более 256.
КОМАНДНАЯ СТРОКА ОЧЕНЬ ДЛИННА	очень много символов в INPUT-файле.
МНОГОКРАТНОЕ ОПИСАНИЕ	указанный объект определен как PUBLIC более чем в одном модуле.
НЕЛЬЗЯ СОЗДАТЬ	Обычно из-за того, что файл EXE занят в другом окне Windows

НЕ НАЙДЕН ФАЙЛ	не найден файл на указанном диске.
НЕТ СТЕКОВОГО СЕГМЕНТА	нет настройки стека в выполняемом файле. Это предупреждающая информация, обычно при параметре NOP.
НЕ НАЙДЕН… ВНЕШНЕЕ ИМЯ	ссылка на объект, который не описан ни в одном из модулей.
НЕТ МЕСТА ДЛЯ ФАЙЛОВ ПО УМОЛЧАНИЮ	слишком много модулей приходится искать в режиме READ
НЕ НАЙДЕНА ГРУППА	указанная в командной строке группа не найдена.
НЕ НАЙДЕН КЛАСС	указанный в командной строке класс не найден.
НЕ НАЙДЕН СЕГМЕНТ	указанный в командной строке сегмент не найден.
ОШИБКА ПРИ ЧТЕНИИ ДИСКА	невозможно прочитать файл.
ОШИБКА ПРИ ЗАПИСИ НА ДИСК	Обычно, если EXE занят в другом окне Windows
ОШИБКА В АТРИБУТАХ СЕГМЕНТА	ранее уже был сегмент с таким же именем, но с другими атрибутами.
ОШИБКА ПРИ КОМБИНИРОВАНИИ СЕГМЕНТА	сегменты не могут быть объединены как указано.
ОШИБКА ВЫЗОВА	ошибка в программной строке в указанном месте.
ОШИБКА В КЛАССЕ СЕГМЕНТА	уже был сегмент с таким именем, но с другим классом.
ОШИБКА РЕДАКТОРА	ошибка при подсчете индексов сегментов и групп, возможно из-за предыдущих ошибок.
ПЕРЕПОЛНЕНА ТАБЛИЦА	нет места для таблицы объектов, уменьшите число и длину общих идентификаторов в программах.
СЕГМЕНТ БОЛЕЕ 16М	сумма всех сегментов с одним и тем же именем больше 16 Мбайт
ССЫЛКА ВНЕ ГРАНИЦ	ссылка на метку вне данной локальной области.
СЛИШКОМ МНОГО МОДУЛЕЙ В БИБЛИОТЕКЕ	в библиотеке более 256 модулей.
ТАКОЙ FIXUPP НЕ РЕАЛИЗОВАН	ошибка в OBJ или модуль получен с помощью неизвестного редактору LINK-KT транслятора.
ТАКОЕ ВЫРАВНИВАНИЕ НЕ РЕАЛИЗОВАНО	ошибка в OBJ или выравнивание типа PAGE, UNNM, LTL

ТАКОЕ КОМБИНИРОВАНИЕ НЕ РЕАЛИЗОВАНО	ошибка в OBJ или модуль получен с помощью неизвестного редактору LINK-KT транслятора.
ТАКОЙ ТИП ГРУППЫ НЕ РЕАЛИЗОВАН	ошибка в OBJ или модуль получен с помощью неизвестного редактору LINK-KT транслятора.
ТАКАЯ ЗАПИСЬ НЕ РЕАЛИЗОВАНА	ошибка в OBJ или модуль получен с помощью неизвестного редактору LINK-KT транслятора.

27.18.Ошибки формата OMF 8086/8087

Ошибка	Причина
1	ДЛИНА ИМЕНИ БИБЛИОТЕКИ БОЛЬШЕ 8 СИМВОЛОВ
2	ДЛИНА ЗАПИСИ БОЛЕЕ 1024 БАЙТ
3	РАЗМЕР ДАННЫХ В LEDATA ИЛИ LIDATA БОЛЬШЕ ДЛИНЫ СЕГМЕНТА
4	НЕ СОВПАЛА КОНТРОЛЬНАЯ СУММА ЗАПИСИ
5	НЕВЕРНЫЙ ТИП ЗАПИСИ
6	БИБЛИОТЕЧНЫЕ ЗАПИСИ В ФАЙЛЕ OBJ
7	НЕТ СЛУЖЕБНЫХ ЗАПИСЕЙ В БИБЛИОТЕКЕ
8	НЕСУЩЕСТВУЮЩИЙ ИНДЕКС
9	НЕТ ГОЛОВНОЙ ЗАПИСИ В МОДУЛЕ
10	ДЛИНА ИДЕНТИФИКАТОРА БОЛЕЕ 40 СИМВОЛОВ
11	ЗАПРЕЩЕННЫЙ СИМВОЛ В ИДЕНТИФИКАТОРЕ
12	НЕВЕРНАЯ ДЛИНА ЗАПИСИ
13	НУЛЕВОЙ КОЭФФИЦИЕНТ ПОВТОРЕНИЯ В LIDATA
14	НЕВЕРНАЯ СТРУКТУРА ЗАПИСИ LIDATA
15	НЕВЕРНЫЙ ТИП MODEND
16	ЗАПРЕЩЕННЫЙ МЕТОД ДЛЯ СТАРТОВОЙ МЕТКИ В MODEND
20	МЕТОД В FIXUPP БОЛЬШЕ F3
21	МЕТОД В FIXUPP БОЛЬШЕ F6
22	ПОЛЕ LOC В FIXUPP БОЛЕЕ 4
23	НЕВЕРНЫЙ НОМЕР ПОЛЯ В FRAME FIXUPP
24	НЕ ЗАДАН THREAD ДЛЯ FRAME FIXUPP
25	НЕДОПУСТИМЫЙ МЕТОД В FRAME FIXUPP
26	НЕ ЗАДАН THREAD ДЛЯ TARGET FIXUPP
27	ЗАДАН СЕГМЕНТ ПРИ ВНУТРИСЕГМЕНТНОМ МЕТОДЕ FIXUPP
28	ВЫХОД ЗА ГРАНИЦУ СЕГМЕНТА В FIXUPP

28. РЕДАКТОР БИБЛИОТЕКИ LIB-KT

28.1. Основные понятия

LIB-KT это программа, создающая файл-библиотеку из объектных модулей. Такие модули могут быть получены в результате работы компилятора PL/1-KT и ассемблера RASM-KT. Редактор LIB-KT как и все другие утилиты расположен в файле PLINK64.EXE и вызывается директивами:

PLINK64.EXE # <командная строка>

После вызова, LIB-KT читает указанные в командной строке файлы и создает файл-библиотеку, файл перекрестных ссылок и файл-каталог модулей.

LIB-KT работает с файлами, имеющими следующие расширения:

- .INP - файл, где записана командная строка
- .L86 - исходный или результирующий файл-библиотека
- .MAP - файл-каталог модулей
- .OBJ - файл объектного модуля
- .XRF - файл перекрестных ссылок

Если при вызове расширение файлов не указано, то по умолчанию принимается:

- до знака равенства - расширение .L86,
- после знака равенства - расширение .OBJ.

Если в командной строке знака равенства нет - расширение по умолчанию .OBJ. LIB-KT работает с файлами .OBJ и .L86 совершенно одинаково, и определяет модуль или библиотеку по содержимому, а не по расширению.

Кроме имен файлов, в командной строке можно указывать параметры LIB-KT в квадратных скобках. Названия параметров можно сокращать до любой длины, но так, чтобы не получилось двусмысленности.

28.2. Создание новой библиотеки

Для создания библиотеки в командной строке указывается имя библиотеки, затем знак равенства и список включаемых в библиотеку файлов через запятую, например: plink64 # newlib=a,b,c

Файлы с объектным кодом могут идти в любом порядке.

28.3. Добавление файлов в существующую библиотеку

Для того, чтобы дописать файлы в имеющуюся библиотеку, необходимо в командной строке указать имя этой библиотеки справа и слева от знака равенства: plink64 # math=math.l86,sin,cos,tan

28.4. Замена модулей

Можно заменить один или несколько модулей в библиотеке, не меняя самой библиотеки. Общий вид директивы:

PLINK64 # <новое имя> = <старое имя> [REPLACE [<список>]]

<новое имя> - имя, под которым создается библиотека
<старое имя> - имя библиотеки, в которой заменяются модули, старое и новое имя может быть одним и тем же;

<список> - список пар имен через запятую в виде:
<имя модуля>=<имя файла>

пример: plink64 # math=math.l86 [replace [sqrt=newsqrt]]

здесь в библиотеке math тело модуля sqrt будет заменено на модуль из файла newsqrt.obj

Если <имя модуля> и <имя файла> совпадают, имя файла можно не указывать, например: plink64 # math=math.l86 [replace [sqrt]]

28.5. Уничтожение модулей

Общий вид директивы:

PLINK64 # <новое имя>=<старое имя> [DELETE [<список имен>]]

<список имен> - список имен уничтожаемых модулей через запятую, если это ряд соседних модулей, можно указать начальное и конечное имена через тире (минус).

примеры: plink64 # math=math.l86 [delete [add,sub,mul,div]]
plink64 # math=math.l86 [delete [add-div]]

28.6. Выбор модулей из библиотеки

Переписать модули из библиотеки в библиотеку можно следующей директивой:

PLINK64 # <новая библиотека>=<старая библиотека>[SELECT [<список имен>]]

Как и в предыдущем случае <список имен> - список имен модулей через запятую, или граничная пара имен через тире (минус). Получившуюся новую библиотеку можно склеить с другими библиотеками директивами добавления.

пример: plink64 # arith=math.l86 [select [add,mult]]

Замечание: *нельзя одновременно использовать REPLACE и SELECT вместе с DELETE.*

28.7. Создание файла перекрестных ссылок

Создать такой файл можно директивой PLINK64 # <имя> [XREF]

Файл создается с именем <имя>.XRF и содержит список всех глобальных и внешних объектов, а также имена сегментов в алфавитном порядке. Такой файл может быть создан для одного или для нескольких модулей или библиотек.

Модуль, в котором описано данное имя, отмечается знаком #, после имен сегментов ставится их тип в косых скобках, например /CODE/. В конце списка указывается число всех модулей.

28.8. Создание файла-каталога модулей

Такой файл создается директивой: PLINK64 # <имя> [MAP].

Каталог - это список модулей в алфавитном порядке, после каждого имени модуля идет список имен сегментов модуля с указанием их длины. В каталоге также имеется список глобальных объектов (PUBLIC) и внешних объектов (EXTERNAL).

Если указать параметр NOALPHA, имена будут не в алфавитном порядке, а так как они встречаются в библиотеке.

пример: plink64 # math.l86 [map,noalpha]

28.9. Создание частичных каталогов

Можно создать каталог, в котором будут указаны только имена модулей или сегментов или глобальных объектов или внешних объектов. Такие каталоги создаются директивами:

PLINK64 # <имя> [MODULES]
PLINK64 # <имя> [SEGMENTS]
PLINK64 # <имя> [PUBLICS]
PLINK64 # <имя> [EXTERNALS]

Замечание: *директивы MAP и XREF можно использовать вместе с SELECT.*
примеры:
plink64 # math.l86 [map,noalpha,select[sin-tan]]
plink64 # math.l86 [xref,select [sin,cos,tan]

28.10.Запись командной строки в файл

Командную строку для LIB-KT можно записать в файл с расширением .INP и затем запустить редактор директивой:

PLINK64 # <имя> [INPUT]

пример: в файле math.inp записаны строки
math=add [$oc],sub,mul,
;---- комментарий ----
div,cos,sin,
sqrt
обращение: plink64 # math [input]

При обращении можно указывать другие директивы:
plink64 # math [in, map, xref].
Кроме этого, можно указать параметр ECHO, выдающий на экран содержимое файла .INP.

28.11.Директивы ввода/вывода файлов LIB-KT

По умолчанию LIB-KT оперирует с файлами на текущем диске, однако можно явно указать диск для каждого файла, например:
plink64 # e:math=math.l86,d:sin,d:cos,c:tan

Задать букву диска для файлов MAP, XREF и OBJ можно с помощью параметра $td, где t-тип файла (M-MAP, O-OBJ, X-XREF), а d-буква диска, т.е. буквы от A до P или специальные символы (X - экран, Y - печать).

Пример: plink64 # trig[map, xref,$ocmyxy]=sin,cos,tan

Каждый параметр $td действует до начала следующего

пример: plink64 # biglib=a1[$oc], a2, ..., a50[$od], a51, ..., a100

28.12.Ошибки, выдаваемые при работе LIB-KT

В ФОРМАТЕ OMF ОШИБКА N	ошибка в объектном коде, например, затерт файл с объектным кодом. Коды ошибок такие же, как и для LINK-KT
ДИСК ПОЛОН	не хватает места для записи результатов.
НЕ НАЙДЕН ФАЙЛ МОДУЛЬ	при выполнении директив SELECT, DELETE, REPLACE не найден файл или модуль с таким именем.
МНОГОКРАТНОЕ ОПИСАНИЕ	глобальный объект описан более чем в одном модуле.
НЕЛЬЗЯ ЗАКРЫТЬ	LIB-KT не может закрыть файл, например он занят в другом окне Windows
НЕЛЬЗЯ СОЗДАТЬ	LIB-KT не может создать файл, например он занят в другом окне Windows
НЕ НАЙДЕН ФАЙЛ	не найден файл с таким именем.
НЕЛЬЗЯ ПЕРЕИМЕНОВАТЬ	LIB-KT не может переименовать рабочий файл .$$$, например, запущено два LIB-KT одновременно.
ОШИБКА ПРИ ЧТЕНИИ	редактор библиотеки не может прочитать файл.
ОШИБКА ВЫЗОВА	командная строка составлена не по правилам.
ПЕРЕПОЛНЕНА ТАБЛИЦА	не хватает места для имен общих идентификаторов.

29. УТИЛИТА ПРОВЕРКИ МЕЖМОДУЛЬНЫХ СВЯЗЕЙ LINT-KT

Для проверки соответствия внешних данных и параметров процедур, описанных в разных модулях можно использовать простую утилиту LINT-KT. Для ее работы необходимо откомпилировать все модули с ключами «S» и «D», например: plink64 main.pl1 sd и затем запустить утилиту в этой же папке.

Запуск самой утилиты: PLINK64 @

LINT-KT в текущей папке и во всех вложенных папках достает список файлов-протоколов компиляции (с расширением .PRN), откуда читает все внешние данные и параметры процедур, после чего просто сравнивает их между собой на тождество.

Протокол работы утилиты выдается на экран.

Таким образом, для программ, состоящих из большого числа PL/1-модулей, можно быстро и надежно убедиться в правильном использовании процедур и общих данных. Сам редактор связей не может провести такую проверку, поскольку имеет дело только с адресами и внешним представлением имен. Информация о типе данных в смысле структур PL/1 отсутствует в файлах .OBJ.

30. ВСТРОЕННЫЙ СИМВОЛЬНЫЙ ОТЛАДЧИК SID-KT

Интерактивный встроенный символьный отладчик SID-KT (Symbolic Instruction Debugger) является неотъемлемой частью системы программирования PL/1-KT и предназначен для отладки программ, разработанных в этой системе. Отладчик позволяет выполнять программу по командам, по подпрограммам или по строкам исходного текста, останавливаясь в заданных местах и при заданных условиях и показывая при этом имена переменных и меток.

Отладчик представляет собой служебную подпрограмму объемом около 70 Кбайт, которую редактор связи автоматически добавляет в каждый собираемый им EXE-файл. Отладчик может быть:

- активирован при запуске программы;
- активирован при возникновении исключения в программе;
- не активирован и не влиять на работу программы.

30.1. Необходимые данные для отладчика

Необходимый для отладки файл с таблицей символов всегда создается автоматически редактором связи LINK-KT. Этот текстовый файл имеет такое же имя, как и выполняемая программа и расширение .SYM. Если по каким-либо причинам .SYM-файл отсутствует, отладчик сможет работать и без него, однако тогда он не сможет показывать имена меток и переменных.

При этом файл .SYM содержит лишь начальные адреса переменных, но не содержит их типов в терминах PL/1. Поэтому имея только файл .SYM, отладчик может показать эту область памяти только в виде 16-ричных кодов или в формате IEEE-754.

Для того, чтобы выводить данные со свойствами в терминах PL/1 необходимо провести дополнительную подготовку.

Во-первых, все исходные PL/1-модули необходимо скомпилировать с ключами S и D, чтобы компилятор создал текстовые .PRN-файлы содержащие списки используемых в данном модуле переменных с их характеристиками уже в терминах PL/1.

Во-вторых, редактор связи LINK-KT нужно запустить с ключом /DEB. В этом случае, редактор связей просмотрит все имеющиеся в текущей папке файлы с расширением .PRN и дополнительно к .SYM-файлу создаст еще один уже двоичный файл с расширением .TRF, который содержит характеристики переменных, перечисленных в .SYM-файле в терминах PL/1.

Отладчик загружает .SYM-файл в память, чтобы иметь таблицу адресов подпрограмм и переменных. Если он обнаружит еще и .TRF-файл, он загружает в память и его, образуя общую с .SYM таблицу.

Теперь отладчик сможет выводить на экран содержимое переменных не только как просто участков памяти, но и в виде объектов с характеристиками в терминах PL/1.

Пример:

Пусть в программе имеется переменная X, описанная как

DCL X(1:5) FLOAT(53);

и оператор присваивания DO I=1 TO 5; X(I)=I; END;

Если запустить программу с отладчиком и остановиться после выполнения оператора присваивания, то по директиве D .X отладчик выведет на экран:

- без файла .TRF

```
-D .X

002B:0040A630 00 00 00 00 00 00 F0 3F-00 00 00 00 00 00 00 40
002B:0040A640 00 00 00 00 00 00 08 40-00 00 00 00 00 00 10 40
002B:0040A650 00 00 00 00 00 00 14 40-05 00 00 00 00 00 00 00
002B:0040A660 00 00 00 00 00 00 00 00-00 00 00 00 FF FF BF FF
002B:0040A670 74 A6 00 00 19 00 00 00-E7 1A 00 00 00 00 00 00
002B:0040A680 00 0D 0A 8A AE AD A5 E6-20 AF E0 AE A3 E0 A0 AC
002B:0040A690 AC EB 24 0D 0A 91 92 85-90 92 20 91 8F 88 91 8E
002B:0040A6A0 8A 20 94 80 89 8B 8E 82-24 0D 0A 8D 85 20 82 9B
002B:0040A6B0 84 85 8B 85 8D 80 20 8F-80 8C 9F 92 9C 24 66 06
002B:0040A6C0 00 00 0A 12 40 00 00 00-00 00 00 00 00 00 00 00
002B:0040A6D0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
002B:0040A6E0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

- с файлом .TRF

```
-D .X

002B:0040A630 .X(1:5)   FLOAT(53)
         1.00000000E+000         2.00000000E+000         3.00000000E+000
 4.00000000E+000
         5.00000000E+000
```

Замечание: *обратите внимание, что компиляция с ключами S и D может замаскировать ошибки компиляции, в том смысле, что они тоже будут выведены не на экран, а в файл с расширением .PRN.*

Рекомендуем в таких случаях использовать скрипт и выполнять компиляцию два раза: сначала с выводом ошибок на экран, а если их не было, то повторять компиляцию уже с ключами S и D:

```
PLINK64 TEST.PL1
IF ERRORLEVEL 1 GOTO :EOF
PLINK64 TEST.PL1 SD
PLINK64 TEST /DEB
```

30.2. Запуск программы с отладчиком

Поскольку отладчик является подпрограммой, «зашитой» в каждый EXE-файл, то для работы с ним требуется лишь активизация (и желательно наличие .SYM и .TRF файлов).

На практике отладчик обычно вызывают в двух случаях:

- перед выполнением программы, чтобы поставить контрольные точки, посмотреть начальные значения переменных и т.п.;
- при аварийном завершении программы, чтобы проанализировать текущие значения переменных.

Для этих двух случаев предусмотрены ключи «??» или «?», которые ставятся в начале командной строки при вызове PL/1-программы.

Используя ключ «??», Вы сразу попадаете в интерактивный режим отладчика до начала выполнения операторов PL/1-программы, но уже после работы служебной подпрограммы подготовки к исполнению программы, которая, например, выделяет память кучи и собственно активизирует отладчик.

С ключом «?» PL/1-программа начинает выполняться обычным образом, а в случае возникновения ошибки (исключения или особого случая) вызывается отладчик. При этом если программа завершилась успешно - отладчик так и не вызывается. Можно искусственно вызвать особый случай отладки и попасть в отладчик, если написать в PL/1 программе оператор UNSPEC('CC'B4);

Если в командной строке этих ключей не задано, отладчик не используется и никак не влияет на работу программы. В этом случае можно использовать «внешние» отладчики типа WinDbg. К сожалению, работать одновременно с двумя и более отладчиками невозможно.

Вызов программы с отладкой: <программа> [??/?] [<командная строка>]

Например, если обычный вызов без отладки: PROGRAM 1 2 3 4 5 'A'

то вызов отладчика вначале: PROGRAM ?? 1 2 3 4 5 'A'

вызов отладчика при аварийном завершении: PROGRAM ? 1 2 3 4 5 'A'

Замечание: *при запуске программы первые символы ? или ?? исключаются из командной строки и с точки зрения PL/1-программы доступная ей командная строка (как входной параметр главной процедуры) никогда не содержит таких символов. Сами символы ? или ?? необязательно должны отделяться пробелом от остальной командной строки.*

30.3. Стандартная информация отладчика

В результате активизации отладчика в начале программы, или в случае возникновения исключения при работе программы, Вы попадаете в интерактивный режим отладки, о чем говорит приглашение «-» и вывод стандартной информации, например:

```
Загрузка файла ссылок
#83
AX=0000  BX=0030  CX=7764  DX=7765  SP=0012  BP=0000  SI=0041  DI=0030
   0001     0014     7FA8     1DB0     FF48     0000     3200     3320
R8=0065  R9=0000 R10=0000 R11=7765 R12=0000 R13=0000 R14=0000 R15=0000 CTP
   0430     0012     0200     1DB0     0000     0000     0000     0000
CS=0033  DS=002B  SS=002B  ES=002B  FS=0053  GS=002B  IP=00401214  --I------
00401214 C7053A94000005000000          MOV    D PTR [0040A658],00000005  DS:0040
A658=00000000
-
```

Здесь выдано сообщение, что успешно загрузился файл ссылок (.SYM), а также к нему добавлено 83 объекта из файла .TRF. Таким образом, у отладчика имеются и адреса переменных и их типы.

Стандартная информация состоит в выводе текущего состояния регистров процессора и кода текущей команды с ее дисассемблированием. При этом данная команда еще не выполнилась и может быть выполнена при продолжении работы. Выводится также состояние флагов процессора, а если компиляция была с ключом R, то выводится также номер текущий строки исходного текста текущего модуля и первые две буквы его имени.

Обратите внимание, что выводятся только младшие половины 8-байтных регистров. В большинстве случаев в старших 4 байтах содержатся нули. Признаком нулей в старших половинах служит знак равенства, отделяющий имя регистра от его содержимого. Если старшие 4 байта содержат не нули, знак равенства заменяется на знак неравенства. Посмотреть весь регистр можно отдельной командой. Такой прием позволяет не забивать экран ненужной информацией, поскольку даже если старшая половина регистра содержит не нули, а часть адреса, эта часть меняется редко.

В примере выше, все старшие части всех регистров содержали нули.

В следующем примере отладчик остановился на 685 строке модуля, имя которого начинается с SI и старшая половина регистра RAX не равно нулю.

```
AX#0000  BX=01B5  CX=7764  DX=7765  SP=0012  BP=0000  SI=0042  DI=01B5
   0001     0015     7FA8     1DB0     FF48     0000     0200     4B87
R8=001C  R9=0000 R10=0000 R11=7765 R12=0000 R13=0000 R14=0000 R15=4953 CTP SI
   0430     0012     0200     1DB0     0000     0000     0000     02AD   685
CS=0033  DS=002B  SS=002B  ES=002B  FS=0053  GS=002B  IP=00401210  --I------
00401210 BEBC4B4100                    MOV    ESI,00414BBC
```

Регистр RAX можно посмотреть (и изменить) отдельной командой:

```
-rax

AX FFFFFFFF'00000001 .

-
```

Стандартная информация, которая выдается при каждом входе в отладчик, также может быть запрошена в любой момент по команде R без параметров.

30.4. Команды отладчика

В интерактивном режиме можно вводить различные команды отладчика, тем самым, влияя на состояние программы и ее работу. Большинство команд совпадает с «классическим» набором команд отладки, присутствующим почти во всех отладчиках и восходящим еще к временам MS DOS.

Почти все команды отладчика состоят из одной латинской буквы и параметров после нее. Ряд команд может иметь префикс «-». Ниже описываются команды, но не в алфавитном порядке, а в порядке их важности.

30.4.1. Команда Q. Выход из отладчика

Это, безусловно, самая важная команда задается просто как Q. Выход из отладчика произойдет также, если закончилась отлаживаемая программа самостоятельно или по клавишам CTRL+C.

С окончанием отладчика немедленно оканчивается и отлаживаемая программа.

30.4.2. Команда вывода подсказки отладчика

Краткий перечень команд выдается на экран, если ввести символ «?», а если ввести два этих символа подряд, выдается краткое описание форматов команд.

30.4.3. Ввод операндов в командах

В большинстве команд отладчика в качестве операндов задаются адреса памяти. Так же, как и в «старой доброй» программе DEBUG, адреса обычно задаются шестнадцатеричными числами, однако в данном отладчике (поскольку он все-таки символьный) операнды могут быть заданы и несколькими другими способами:

- как имена регистров (всегда с буквы E, например, EBX, реально будет взят RBX, но задать напрямую регистры R8-R15 нельзя);
- как имена переменных из PL/1-программы, тогда начинаются с точки;

- как десятичное число – тогда начинается с символа #;
- как символ ^, тогда означает содержимое по адресу из стека, т.е. [RSP], два ^^ - как [RSP]+8, ^^^ - как [RSP]+16 и т.д.;
- как @ - тогда это признак косвенной адресации;
- как сумма или разность двух операндов;
- как символьная строка из одного или более символов в кавычках (для некоторых команд, но это не адрес).

Несколько примеров:

нужно задать операнд 401000 (шестнадцатеричное), допустим, это и адрес переменной НАЧАЛО

```
401000      (в шестнадцатеричном виде по умолчанию)
#4198400  (в десятичном виде)
.НАЧАЛО  (в символическом виде)
ESI          (через регистр, если в регистре записано значение 401000)
```

Адрес может быть выражением:

```
400100+2
#256-#64
.НАЧАЛО-.КОНЕЦ
ESI+EDI
^
^^
@.НАЧАЛО
```

30.4.4. Команда D. Просмотр памяти

Команда имеет вид:

```
D[W/D/F/FF] <начало>  <конец>
D[W/D]        <начало> +<число>
```

где <начало> - это начальный адрес вывода, <конец> - конечный адрес вывода, <число> - число выводимых байт.

Содержимое памяти выводится в виде шестнадцатеричных кодов и соответствующим им текстовых строк. Если очередной байт имеет код меньше 20 или 7F или FF, в строке он заменяется на символ точки. Содержимое памяти может выдаваться байтами (команда D), словами (DW) двойными словами (DD),

в формате чисел с плавающей точкой (DF) и в виде чисел с плавающей точкой двойной точности (DFF)

Если <конец> или <число> не указаны, по умолчанию выводится 12 строк по 16 байт. (132 байта). Эту порцию можно изменить директивой

-D <число>,

например: -D 80 означает вывод по умолчанию 128 байт (80 шестнадцатеричное).

Если <начало> не указано, вывод идет с текущего адреса данных (этот адрес запоминается до следующей директивы D). Первоначально адрес по умолчанию равен 400000, что соответствует обычному началу доступной PL/1-программе памяти.

По команде DF (DFF) содержимое памяти представляется в формате сопроцессора обычной или двойной точности IEEE-754. При этом каждое число занимает в памяти соответственно 4 и 8 байт.

Примеры:

```
-d 400000

002B:00400000 4D 5A 90 00 03 00 00 00-04 00 00 00 FF FF 00 00 MZP.............
002B:00400010 B8 00 00 00 00 00 00 00-40 00 00 00 00 00 00 00 ╕.......@.......
002B:00400020 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
002B:00400030 00 00 00 00 00 00 00 00-00 00 00 00 70 00 00 00 ............p...
002B:00400040 0E 1F BA 0E 00 B4 09 CD-21 B8 01 4C CD 21 54 68 ..╢..┤.=!╕.L=!Th
002B:00400050 69 73 20 70 72 6F 67 72-61 6D 20 63 61 6E 6E 6F is program canno
002B:00400060 74 20 62 65 20 72 75 6E-20 69 6E 20 44 4F 53 20 t be run in DOS
002B:00400070 50 45 00 00 64 86 03 00-F9 D9 F7 60 00 00 00 00 PE..dЖ..·┘ў`....
002B:00400080 00 00 00 00 F0 00 22 00-0B 02 09 00 00 E1 01 00 ....Ё."......c..
002B:00400090 00 00 00 00 00 00 00 00-00 10 00 00 00 10 00 00 ................
002B:004000A0 00 00 40 00 00 00 00 00-00 10 00 00 00 02 00 00 ..@.............
002B:004000B0 06 00 01 00 06 00 01 00-06 00 01 00 00 00 00 00 ................

-dw 400000 +1f

002B:00400000 5A4D 0090 0003 0000-0004 0000 FFFF 0000         MZP.............
002B:00400010 00B8 0000 0000 0000-0040 0000 0000 0000         ╕.......@.......

-dd 400000 +1f

002B:00400000 00905A4D 00000003-00000004 0000FFFF     MZP.............
002B:00400010 000000B8 00000000-00000040 00000000     ╕.......@.......




-dd esp

002B:0012FF48 0040C0E0 00000000-00401007 00000000     p└@.......@.....
002B:0012FF58 7731556D 00000000-00000000 00000000     mU1w............
002B:0012FF68 00000000 00000000-00000000 00000000     ................
002B:0012FF78 00000000 00000000-00000000 00000000     ................
```

```
000000000040C0E0 0040C0E0 .?STOPX
```

Просмотр памяти через операнд регистр ESP (реально RSP) приводит к дополнительным действиям. Каждые 8 байт с вершины стека и вниз (всего 100H) проверяются на попадание в таблицу адресов подпрограмм, и потом отдельно выдается список адресов и ближайшие адреса меток и подпрограмм к ним. В данном примере в стеке найден адрес строго совпадающий со служебной подпрограммой ?STOPX. Если используется другой операнд, не ESP, таких дополнительных действий не выполняется.

Это сделано для упрощения анализа адресов возврата в стеке. Заметим также, что Windows требует всегда кратности стека 8. Попытка нарушить отладчиком кратность стека 8 приведет к краху и отладчика и всей задачи.

```
-dff 400000

002B:00400000 6.37066138E-314 1.39064994E-309 9.09080788E-322 3.16202013E-322
002B:00400020 0.00000000E+000 0.00000000E+000 0.00000000E+000 2.37663528E-312
002B:00400040-1.32170658E+063 3.67404926E+194 1.25013747E+243 5.76720549E+228
002B:00400060 1.24033552E+224 5.76071297E-153 4.90206491E-309 8.03773625E-315
002B:00400080 5.00743473E-308 2.61294074E-309 0.00000000E+000 8.69169476E-311
002B:004000A0 2.07226151E-317 1.08646184E-311 1.39079848E-309 3.23820505E-319
```

В последнем примере показан вывод памяти в формате IEEE-754. Наличие огромных степеней практически всегда показывает, что реально в этой области памяти находятся вовсе не числа с плавающей точкой.

30.4.5. Контрольные точки

В программе можно указать несколько мест, где необходимо остановить выполнение (например, чтобы посмотреть содержимое памяти). Такие места называются контрольными точками. Это должны быть адреса каких-то команд в программе, а, чаще всего, это будут адреса меток в программе. Если Вы пишете программу на PL/1, Вы можете заранее расставить метки в тех местах программы, где интересно остановиться. «Лишние» метки практически не влияют на выполнение EXE-файла.

Контрольные точки могут быть программные и аппаратные. Установка программной контрольной точки заключается в том, что в код команды по указанному адресу помещается специальная однобайтовая команда INT 3, при выполнении которой, управление попадет в отладчик. Таким образом, остановка программы и выход в отладчик происходит ДО выполнения команды по адресу контрольной точки. Аппаратные контрольные точки используют отладочные регистры процессора. Использование аппаратных контрольных точек позволяют не только остановиться в любом месте программы, но и остановиться по событию чтение/запись заданной ячейки памяти. Следует иметь в виду, что остановка по контрольной точке по данным происходит ПОСЛЕ выполнения команды,

выполнявшей чтение или запись. Однако аппаратная остановка по команде происходит тоже ДО выполнения команды.

Программные контрольные точки могут быть оперативные и постоянные. Оперативных контрольных точек может быть одна или две, и они указываются при каждом запуске или продолжении отлаживаемой программы командой G (см. ниже). Такие точки автоматически снимаются, если управление снова вернулось в отладчик, независимо от того, произошла ли остановка из-за того, что программа дошла до одной из этих точек или попала на одну из постоянных точек.

Постоянные контрольные точки устанавливаются и снимаются с помощью соответствующих команд, обычно перед запуском программы. Одновременно Вы можете установить 16 постоянных программных контрольных точек и четыре постоянных аппаратные точки.

Замечание: *современные процессоры позволяют установить до 8 контрольных точек, однако Windows не дает использовать дополнительные 4 отладочных регистра (их нет в контексте исключения), и можно использовать только 4 точки, как в 32-разрядных процессорах.*

30.4.6. Команда установки контрольной точки

Вид команды: K [Z]<адрес> [<число раз>] [тип] [размер] [условие]

Здесь Z необязательный признак звонка. Он позволяет установить контрольную точку со звонком. При прохождении программы через эту точку отладчик не будет выдавать на экран стандартную информацию, а вместо этого выдаст звуковой сигнал. Заметим, что можно регулировать длительность звонка, задав в команде не одну, а две и больше букв Z подряд.

<адрес> - адрес команды или переменной (обычно символьное имя), на которую устанавливается данная точка.

<число раз> - счетчик проходов через эту точку до остановки (по умолчанию равен 1).

<тип>- или K или W или R - один из трех возможных типов аппаратных контрольных точек (если тип не указан – задается программная контрольная точка):

K - аппаратная контрольная точка по команде по заданному адресу

W - аппаратная контрольная точка по записи по заданному адресу

R – аппаратная контрольная точка по чтению или записи. по заданному адресу.

<размер> - для контрольных точек по данным, задает размер объекта 1, 2 или 4 байта (по умолчанию 4).

<условие> - для контрольных точек по данным можно задать условие сравнения с константой. Для этого нужно задать знак больше, меньше, равно, не

равно (>, < , = , #) и саму константу. Такая возможность очень удобна для остановки при условии достижения какой-то переменной определенного значения при этом счетчик прохождений через контрольную точку автоматически задается большим числом.

Несколько примеров.

Допустим, программа ломается после длительной работы и неизвестно где. Удалось установить, что управление обязательно проходило чрез метку M100.

Ставим контрольную точку командой K .M100 1000000 и запускаем программу. На экран выдается каждое прохождение через метку и оказывается, что через 10457 прохождений происходит крах. Тогда запускаем программу заново, устанавливаем контрольную точку K .M100 10457 и дожидаемся остановки в отладчике. Теперь можно идти по шагам и уточнять место ошибки.

Пример установки контрольной точки по записи в переменную X значения пять: K .X W1 =5

Пример установки контрольной точки по записи в переменную X значения не равного пяти: K .X W1 #5

30.4.7. Команда снятия контрольной точки

Команда имеет вид: -K [<адрес>]

Если адрес не указан, снимаются все ранее установленные контрольные точки.

30.4.8. Команда просмотра контрольной точки

Команда имеет вид: K [<адрес>]

Если адрес не указан, выводится список всех контрольных точек.

Информация о контрольных точках имеет следующий вид:

```
1/0   0023:0040B8BC .BB W4
10/0 0023:0040122D .M K0
```

Здесь первое число – заданный счетчик, через дробь число раз, реально пройденных через эту точку, далее адрес контрольной точки и соответствующее ему символьное имя, затем тип контрольной точки (только для аппаратных точек) и, если было задано, условие сравнения с константой. Такая же информация выдается при прохождении через любую из контрольных точек.

30.4.9. Команда G. Запуск отлаживаемой программы

Команда имеет вид:

[-]G [=<начало>] [<1 контрольная точка>] [<2 контрольная точка>]

<начало> - это адрес, с которого надо начать выполнение программы, если он не указан, выполнение начнется с текущего адреса в RIP, что практически всегда и нужно.

<контрольные точки> - это оперативные контрольные точки (см. предыдущий раздел). Если Вы не установили постоянных контрольных точек командой K и не указали оперативных точек в команде G, то программа пойдет выполняться без контроля отладчика (если конечно в самой программе нет вызовов исключения отладки INT 3). Таким образом, если Вы вызвали программу с отладчиком и просто ввели G, то Ваша программа выполнится как при обычном вызове и присутствие отладчика фактически никак не скажется.

В программе можно также наставить команд INT 3, (на PL/1 это обращение к оператору UNSPEC ('CC'B4);) и каждая такая команда сработает как постоянная контрольная точка. Правда, эта контрольная точка никогда не снимется, и чтобы продолжить программу ее придется обойти, например, командой N

Примеры:

g =.m1 .m2
начать программу с метки m1 и остановиться, если дошли до m2
g .m2
начать программу с текущего места (например, сначала) и остановиться, если дошли до m2
g =.m1 .m2 .m3
начать программу с метки m1 и остановиться, если дошли до m2 или до m3

Если при выполнении программы после команды G, она проходит через контрольные точки, установленные командой K, на экран выдается сообщение об этом, однако такие сообщения можно отменить, если перед буквой G задать знак «-».

30.4.10. Команда T. Пошаговое выполнение программы (трассировка)

Команда имеет вид: [-]T <число>

По команде T исполнится текущая команда процессора, и управление вернется в отладчик. Этот режим, называемый трассировкой, использует аппаратную возможность процессора вырабатывать сигнал прерывания после выполнения одной команды (исключение пошагового режима).

После каждого шага, отладчик выдает стандартную информацию на экран. Задав <число> можно выполнить указанное число команд без остановки после каждой, причем такое выполнение можно прервать посередине, нажав CTRL+C. Используя префикс «-», можно отменить выдачу стандартной информации на промежуточных шагах.

30.4.11. Команда N. Переход через INT 3

Команда имеет вид: N

Когда программа остановилась на команде INT 3, перейти к следующей команде командой трассировки не удастся. В этом случае следует изменить на 1 значение RIP, что и сделает отладчик по этой команде. Перепрыгнуть через любую другую команду так не удастся, отладчик выдаст сообщение об ошибке.

30.4.12. Команда P. Пошаговое выполнение с пропуском подпрограмм

Команда имеет вид: [-]P <число>

Команда P без параметров аналогична команде T, за исключением того, что она выполняет за один шаг команды вызова подпрограмм (CALL) и «цепочечные» команды (типа REP MOVS). В этих случаях ставятся неявные контрольные точки. В остальных случаях команда действует как T.

Возможно, Вы будете использовать эту команду чаще всего, так как она позволяет пропускать служебные вызовы. Можно задать параметр <число>, чтобы выполнить без остановки соответствующее число команд P, а использование префикса «-» отменит выдачу на экран стандартной информации на промежуточных шагах.

Использование команды P может оказаться мощным средством поиска ошибочных подпрограмм. Вы доходите до вызова очередной подпрограммы, смотрите содержимое интересующих Вас данных до исполнения подпрограммы, затем командой P исполняете ее и смотрите в памяти результаты ее работы. Заметим, что команды -T <число> и -P <число> работают, конечно, гораздо медленнее, чем, если бы просто непрерывно выполнилось <число> команд программы. Ведь в этом случае, после каждой команды программы происходит выход в отладчик, а затем опять вход в текущее место программы.

30.4.13. Команда B. Построчное выполнение с пропуском подпрограмм

Команда имеет вид B <номер строки> или [-]B<количество строк>

Кроме пошагового исполнения отладчик имеет возможность при отладке исполнять PL/1-программу по строке исходного текста. Для этого следует использовать ключ компиляции R (по этому ключу в исполняемый файл занесется информация о строках), и тогда при отладке можно использовать команду B.

Команда B без параметров выполнит команды PL/1-программы, пока не изменится текущий номер строки исходного текста. Другими словами выполнит одну строку исходного текста.

Если же задать <номер строки> остановка произойдет, когда номер текущей строки совпадет с заданным, т.е. на первой команде заданной строки.

Есть другая форма команды, с префиксом «-», здесь параметр <количество строк> задает не номер, а сколько строк следует выполнить до остановки. –B без параметра считается командой –B 1.

30.4.14. Команда R. Вывод стандартной информации, запись регистров и флагов

Команда имеет вид: R <имя регистра> или R <имя флага>

По команде R без параметров на экран выводится стандартная информация для текущего места в программе.

Если после R указать без пробела имя регистра или флага, отладчик предложит изменить его значение.

В качестве имен регистров используются только две последних буквы их имени. Таким образом, допустимы только команды:

RAX RBX RCX RDX RSP RBP RSI RDI RIP

После указания одной из этих команд отладчик выведет полное значение регистра на экран и будет ожидать ввода.

Если ввести символ точки, ввод прекратится, и значение не изменится.

Если нажать «Enter», значение также не изменится, но отладчик выведет значение следующего по порядку регистра (от RBX до RIP) и опять будет ожидать ввода нового значения.

Если набрать новое значение и нажать «Enter», значение изменится и отладчик выведет значение следующего по порядку регистра (от RBX до RIP) и опять будет ожидать ввода нового значения.

Таким образом, можно посмотреть и задать полные значения всех регистров, включая R8-R15.

Командой типа RAX <значение> можно сразу присвоить регистру RAX значение, не переходя в интерактивный режим ввода. Однако для регистров R8-R15 единственный способ изменения и просмотра – интерактивный ввод. Для ускорения, просмотр и изменение R8-R15 можно проводить командой RDI, та как следующий по порядку регистр – R8.

В качестве значений для регистров могут быть и символические имена.

По команде RIP меняется текущее место в программе, и теперь ее выполнение может быть продолжено с заданного места.

Например, можно повторить выполнение команд в программе, если указать RIP <адрес>, где <адрес> - место, через которое уже прошло управление.

Просмотр и изменение флагов процессора производится командой R, за которой без пробела указывается буква флага, например, RC. Значения флажков меняются только интерактивно и только по одному. Они могут иметь значения только 0 или 1. Имена флагов:

Символ флага	**Назначение флага**
O	переполнение
D	направление
I	разрешение прерываний
T	пошаговая трассировка
S	отрицательный результат
Z	нулевой результат
A	дополнительный перенос
P	контроль четности
C	перенос

30.4.15. Команда Z. Просмотр регистров FPU

По команде Z на экран выводится текущее состояние рабочих и служебных регистров FPU. Если в регистры ST0-ST7 ничего не записывалось, они будут помечены как «пустые».

Например,

```
-z

  CW=0360  SW=0000  TW=0000  IP=00000000  OP=00000000 DBE3/FNINIT
   ПУСТОЙ РЕГИСТР    ПУСТОЙ РЕГИСТР    ПУСТОЙ РЕГИСТР    ПУСТОЙ РЕГИСТР
   ПУСТОЙ РЕГИСТР    ПУСТОЙ РЕГИСТР    ПУСТОЙ РЕГИСТР    ПУСТОЙ РЕГИСТР
```

Кроме регистров ST0-ST7 выводится управляющее слово CW и слово текущего состояния SW, а также адрес последнего обращения к FPU и код и имя последней выполненной команды FPU.

Имеется также несколько разновидностей команды Z:

ZS для показа регистров XMM:

```
-zs
X0= 0.0000000000E+000 0.0000000000E+000 X1= 0.0000000000E+000 0.0000000000E+000
X2= 0.0000000000E+000 0.0000000000E+000 X3= 0.0000000000E+000 0.0000000000E+000
X4= 0.0000000000E+000 0.0000000000E+000 X5= 0.0000000000E+000 0.0000000000E+000
X6= 0.0000000000E+000 0.0000000000E+000 X7= 0.0000000000E+000 0.0000000000E+000
```

ZM для показа регистров MMX в 16-ричном виде

```
-zm

  00000000-00000000  00000000-00000000  00000000-00000000  00000000-00000000
  00000000-00000000  00000000-00000000  00000000-00000000  00000000-00000000
```

ZA для показа регистров 3DNow в «плавающем» виде

```
-za

 0.00E+000 0.00E+000 0.00E+000 0.00E+000 0.00E+000 0.00E+000 0.00E+000 0.00E+000
 0.00E+000 0.00E+000 0.00E+000 0.00E+000 0.00E+000 0.00E+000 0.00E+000 0.00E+000
```

30.4.16. Команда H. Просмотр имен и шестнадцатеричная арифметика

Команда имеет вид:

H <имя> или
H <адрес> или
HH или
H <шестнадцатеричное число> < шестнадцатеричное число>

Если для программы сформирован файл имен (он имеет расширение .SYM), то команда H может работать с этим списком имен. H без параметров покажет весь список имен. Список выводится поэкранно и останавливается до нажатия любой клавиши. По клавише ESC выдача прекращается.

Если задать в качестве параметра имя (имя всегда должно начинаться с символа точка) отладчик ищет заданное имя в списке имен, доступных программе и в случае, если имя нашлось, выводит на экран его смещение, в соответствии с таблицей символов. Если для программы сформирован и файл .TRF, содержащий характеристики переменных в терминах PL/1, то кроме адреса и имени будут выведены и эти характеристики.

Например:

```
-h .deltadate

00415578 .DELTADATE     FLOAT(53)
```

Если задать в качестве параметра шестнадцатеричный адрес можно выяснить, какое имя располагается по заданному адресу:

```
-h 415578

00415578 .DELTADATE     FLOAT(53)
```

Если такого имени не нашлось, отладчик покажет ближайшее к заданному адресу имя. Если это имя подпрограммы, то по выданному адресу можно легко сориентироваться, в каком месте программы остановился отладчик.

Поскольку чаще всего необходимо определиться в текущем месте программы, то можно выполнить команду: H EIP или в более простой форме без параметров командой HH:

```
-h eip

Внутри 00401200 .SIG_PAR        PROC

-hh

Внутри 00401200 .SIG_PAR        PROC
```

Эта же команда H <число><число> используется и для простейшей шестнадцатеричной арифметики, чтобы простейшие действия с адресами проводить здесь же, не вызывая приложение «калькулятор».

Для шестнадцатеричной арифметики задается два шестнадцатеричных числа через пробел в качестве параметров.

В результате выводится строка результатов, которая содержит десятичное представление чисел (со значком #) и далее результаты сложения, вычитания, умножения, деления этих двух чисел. Результат деления состоит из двух чисел – частное и остаток:

```
-h 100 20

#256 #32 +00000120 -000000E0 *0000000000002000 /00000008:00000000
```

30.4.17. Команда U. Вывод команд в виде строк ассемблера

Команда имеет вид:

[-]U <начало><конец> или

[-]U <начало> +<число> или

U/ или
U*<адрес>

По команде U на экран выводится указанная область памяти в виде команд ассемблера, близкого к ассемблеру RASM-KT. Для этого в отладчике имеется дисассемблер, который занимает более половины объема всего отладчика.

Если <конец> или <число> не заданы, выводится 12 строк.

Если <начало> не задано, выводится с текущего места в программе.
Если указан знак плюс, то перед ним должен быть пробел, а после –нет.
Примеры:

U 401200
U .m1 +100
U .НАЧАЛО

В каждой строке выводится метка, если есть, текущий адрес команды, ее шестнадцатеричный код, собственно сама команда, символьное имя, если операнд совпал с адресом какой-либо переменной таблицы символов:

```
00401344 E8FC6C0000              CALL   00408045 .?SIOOP
00401349 E8DE720000              CALL   0040862C .?GNVOC
0040134E 7917                    JNS    00401367
00401350 F9                      STC
00401351 E8E59B0000              CALL   0040AF3B .?QCFOP
00401356 BBD84B4100              MOV    EBX,00414BD8
0040135B DD0424                  FLD64  [RSP]
0040135E DC0B                    FMUL64 [RBX]
00401360 58                      POP    RAX
00401361 DD1D51420100            FST64P [004155B8] .EPS
00401367 E8C0720000              CALL   0040862C .?GNVOC
0040136C 790D                    JNS    0040137B
```

Вывод шестнадцатеричного кода можно отменить (или установить) командой U/.

Вывод имен можно отменить, если использовать префикс "-".

Иногда требуется просмотр нескольких команд не «вниз» от текущего места, а вверх. Это можно сделать с помощью команды U*:

```
-u*401367

00401347 0000                    ADD    [RAX],AL
00401349 E8DE720000              CALL   0040862C .?GNVOC
0040134E 7917                    JNS    00401367
00401350 F9                      STC
00401351 E8E59B0000              CALL   0040AF3B .?QCFOP
00401356 BBD84B4100              MOV    EBX,00414BD8
0040135B DD0424                  FLD64  [RSP]
0040135E DC0B                    FMUL64 [RBX]
```

```
00401360 58                          POP    RAX
00401361 DD1D51420100                FST64P [004155B8] .EPS
00401367*E8C0720000                  CALL   0040862C .?GNVOC
0040136C 790D                        JNS    0040137B
```

При этом выводятся команды с адреса на 64 байта меньшего, а команда по заданному адресу будет помечена звездочкой. Однако может так случиться, что команды выше будут выведены неправильно, поскольку их начала неизвестны. В примере самая верхняя команда по адресу 401347 декодирована с середины и неправильно. В этом случае надо еще увеличить число байт «вверх»:

```
-u*401367-20

00401327 41008F07E8C56E              ADD    [R15]+6EC5E807,CL
0040132E 0000                        ADD    [RAX],AL
00401330 45BFC7025349                MOV    R15D,495302C7
00401336 BB30C64100                  MOV    EBX,0041C630
0040133B B994134000                  MOV    ECX,00401394
00401340 66BA9006                    B32:   MOV    DX,0690
00401344 E8FC6C0000                  CALL   00408045 .?SIOOP
00401349 E8DE720000                  CALL   0040862C .?GNVOC
0040134E 7917                        JNS    00401367
00401350 F9                          STC
```

Теперь команды по адресу 401327 декодированы неправильно, но зато команда по адресу 401344 (а не 401347) теперь декодирована правильно.

30.4.18. Команда E. Изменение содержимого памяти

Команда имеет вид: E <начало> или EW <начало>.

Команда позволяет изменить содержимое памяти, начиная с указанного адреса побайтно. После ввода команды, на экран выдается содержимое очередного байта и ожидается ввод. Если просто нажать «Enter», содержимое байта не изменится и произойдет переход к следующему. Окончание ввода – символ «точка»:

```
-e 401200

 B8
 88 89
 92 .
```

Память можно изменять не байтами, а словами, в этом случае, используется EW.

30.4.19. Команда F. Заполнение памяти

Команда имеет вид:

F <начало><конец> <значение> или
F <начало> +<число> <значение> или

FW <начало><конец> <значение> или
FW<начало> +<число> <значение>

Иногда требуется участок памяти заполнить одинаковым значением (например, нулями). В этом случае используется команда F или FW, если память должна быть заполнена словами. Следите, чтобы <значение> было числом не более FF для заполнения байтами и FFFF для заполнения словами.

Перед плюсом должен быть хотя бы один пробел, иначе он воспримется как выражение для начального адреса.

Примеры:

```
f 401200 401400 00
f 401200 +200 #22
fw 401200 401400 5566
```

30.4.20. Команда S. Поиск заданного значения в памяти

Команда имеет вид:
S <начало> <конец> <значение> или
S <начало> +<число> <значение>

По команде S отладчик ищет заданное значение в указанном участке памяти. Значение должно быть или байтом, цепочкой байт или символьной строкой. Перед плюсом должен быть хотя бы один пробел, иначе он воспримется как выражение для начального адреса.

Цепочку байт следует вводить через пробел, начиная с младшего, аналогично тому, как они выглядят на экране по команде D. Символьную строку следует начать с двойной кавычки. Если значение найдено, на экран выдается его адрес в памяти, если не найдено - ничего и не выдается.

Примеры:

```
s 400100 400200 #22
s 401100 402200  ac a0 ae a0 ac
s 400100 +80  "НАЧАЛО
```

30.4.21. Команда C. Сравнение участков памяти

Команда имеет вид:

C <начало1> <конец> <начало2> или
C <начало1> +<число> <начало2>

По команде C отладчик сравнит между собой два заданных участка памяти. Для не совпавших байт выдается адрес и оба значения байта. Если участки совпали - ничего не выдается.

Перед плюсом должен быть хотя бы один пробел, иначе он воспримется как выражение для начального адреса.

30.4.22. Команда M. Перенос участков памяти

Команда имеет вид:

M <начало1> <конец> <начало2> или
M <начало1> +<число> <начало2>

По команде M отладчик пересылает один участок памяти в другой.

Перед плюсом должен быть хотя бы один пробел, иначе он воспримется как выражение для начального адреса.

Пример:

```
m 401200 +100 402000
теперь сравнение дает полное совпадение
c 401200 +100 402000
```

30.4.23. Команда L перехвата всех ON-операторов в программе

В PL/1-программе могут быть установлены различные ON-операторы. При возникновении указанных в этих ON-операторах ситуаций, управление будет передаваться на них. Это затрудняет пошаговую отладку. Например, директивой P отладчика выполняли системную подпрограмму открытия файла. Но если указанный файл не найден и управление передалось на заданный ON-оператор, то установленная отладчиком контрольная точка после подпрограммы не сработает, программа «отцепится» от пошаговой трассировки и начнет выполняться без участия отладчика.

Это может ввести в заблуждение программиста, если он, например, искал, какая подпрограмма выдает ошибку, поскольку внешне именно при выполнении этой подпрограммы и выдалось сообщение на экран. На самом же деле, здесь программа только «отцепилась» от отладчика, пошла выполняться дальше и где-то далее произошла ошибка.

Чтобы избежать таких ситуаций, предусмотрена команда L без параметров, которая устанавливает контрольную точку внутри системной подпрограммы реакции на ситуации с именем «?ON». Через эту единственную точку начинают выполняться все ON-операторы и программист не теряет контроля над

программой со стороны отладчика. Отменить такой перехват можно командой –K, поскольку это обычная контрольная точка.

30.4.24. Команды I и O. Работа с портами ввода/вывода

Команды имеют вид:

I[W] <номер порта>
O[W] <номер порта> <значение>

Отладчик предоставляет возможность читать и писать значения портов ввода/вывода, но только если операционная система позволяет это (например если специальным образом изменить ядро Windows). Если значение порта байт – используются команды I и O, если слово - IW и OW. Например:

```
i  372
iw 372+100

o 372 00
ow 372+100 ffff
```

30.4.25. Команда V. Информация о процессоре

По команде V без параметров можно получить на экране информацию о процессоре, доступную по команде процессора x86 CPUID.

30.4.26. Команда A. Показ кириллицы Windows

Данная вспомогательная команда влияет только когда командой D выводится участок памяти как строки кода, и они же, как текст. Поскольку обычно в консольном окне используется «кириллица DOS», а в окнах Windows – «кириллица Windows», данная команда преобразует в выводе отладчика на экран все в «кириллицу DOS», например:

```
-d 401000 +#15
002B:00401000 8F CF 00 00 00 00 00 00-00 00 00 00 00 00 00 00 П╧..............
-a
Font Win
-d 401000 +#15
002B:00401000 8F CF 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ПП..............
```

После команды A и код 8F и код CF отображаются на экране как «П».

30.5. Создание макрокоманд отладчика

Чтобы упростить ввод одинаковых последовательностей команд существует возможность создания макрокоманд в отладчике.

Макрокоманда – это последовательность нескольких команд отладчика, объединенных под некоторым именем, которую можно вызвать, как обычную команду. Имена макрокоманд могут совпадать с именами обычных команд, поскольку вызов макрокоманд производится с помощью знака равенства.

При этом вся заданная последовательность выполнится в автоматическом режиме. Признаком задания нового макро является двоеточие, признаком исполнения – знак равенства.

Новая макрокоманда определяется следующим образом:

- вводится имя новой макрокоманды с символом двоеточие;
- затем набираются команды отладчика по одной на строке;
- пустая строка является концом макрокоманды.

Пример ввода макрокоманды с именем t:

```
:t

d 400100 400110 <Enter>
d 400200 400210 <Enter>
<Enter>
```

Вызов макрокоманды производится по ее имени, перед которым ставится знак равенства.

Таким образом, если теперь ввести в отладчике новую команду =t, в автоматическом режиме выполнится сначала команда d 400100 400110, а затем d 400200 400210.

31. АССЕМБЛЕР RASM-KT

31.1. Введение в ассемблер

Ассемблер перемещаемого кода (Relocating ASseMbler) RASM-KT предназначен для создания программ на машинном языке для конкретных поколений процессоров 8086/80186/80286/80386/80486/Pentium, включая Pentium 4 и AMD K6-2 а также AMD-64, SSE-2, SSE-4. Т.е. можно генерировать и «древние» коды и современные.

Это необычный ассемблер, насколько вообще может быть необычным язык, представляющий собой набор простых правил записи машинных команд.

Во-первых, это трехпроходный ассемблер, здесь нет ограничений на порядок описания и использования объектов как для двухпроходных ассемблеров (здесь переменные могут быть описаны и до и после их использования).

Во-вторых, RASM-KT когда-то был ориентирован на написание отдельных программ и поэтому может сразу создавать выполняемый образ программы (файл типа .COM), не создавая промежуточного объектного модуля, который потом надо преобразовывать в файлы типа .EXE или .COM.

В-третьих, данный ассемблер имеет механизм добавления новых команд, который внешне похож на макросредства обычных ассемблеров. Такой механизм позволяет, не меняя самого ассемблера, оперативно вводить (с соблюдением всех правил семейства x86) очередные доработки архитектуры процессоров.

И, наконец, этот ассемблер полностью русифицирован, т.е. можно использовать русский алфавит (кириллицу) в любом месте программы, в том числе в именах идентификаторов. Все сообщения при трансляции также выдаются на русском языке.

Данное описание является справочником именно по ассемблеру, а не по процессорам. Предполагается, что читатель знает архитектуру процессоров семейства x86, мнемонику их команд и т.п. Названия регистров и команд в RASM-KT, конечно же, соответствуют официальным названиям из технических описаний процессоров и точно такие же, как и у других ассемблеров. Исключение - названия команд для FPU с одним операндом в памяти. Здесь с помощью чисел 16, 32, 64 или 80 прямо в названии команды явно указывается длина операнда. Кроме этого, если команда не имеет операндов, то признак 32/64-разрядной формы – это удвоенная первая буква команды, например: XLAT использует EBX, а XXLAT – RBX.

Когда-то, в начале 80-х, этот ассемблер (под именем RASM-86) был разработан фирмой «Digital Research». Код его современной версии, включенный вместе с другими утилитами в файл PLINK64.EXE, сильно отличается от той первоначальной и по существу отражает всю историю развития процессоров x86.

Следует отметить, что благодаря удачной реализации, RASM-KT является одним из самых маленьких ассемблеров в мире.

31.2. Синтаксис ассемблера

31.2.1. Набор символов ассемблера

RASM-KT работает с заглавными и строчными буквами латинского и русского алфавита, цифрами, а также следующими знаками:

+ - * / = ( ) [ ] ; ' . ! , _ : @ $? пробел, табуляция, возврат каретки, конец строки.

Везде, за исключением символьных строк, верхний и нижний регистр воспринимается одинаково (это правило можно отменить директивой трансляции "$+").

В символьных строках допускаются любые символы.

В именах идентификаторов русские и латинские буквы, одинаковые по написанию, воспринимаются как один и тот же символ.

Строки исходного текста с ассемблерной программой могут быть любой длины, однако, если заказан листинг, на нем будет распечатано не более 79 байт каждой строки.

31.2.2. Атомы и разделители

Атом - это наименьший неделимый элемент языка (например, число или идентификатор). Разделители служат для выделения атомов на строках печатного текста и могут стоять в любом месте между атомами. Табуляция располагает атом с ближайшей колонки, кратной 8.

31.2.3. Ограничители ассемблера

Ограничители отмечают конец атома или обозначают отдельные части операторов языка. Если стоит ограничитель, разделитель для обозначения конца атома уже не нужен, но разделители делают текст программы более наглядным. Ниже приводится список всех возможных ограничителей и разделителей :

Символ	Наименование	Использование
20H	пробел	разделитель
09H	табуляция	разделитель
CR	возврат каретки	признак конца текущей строки
LF	конец строки	ставится сразу за CR, игнорируется в RASM-KT
;	точка с запятой	начало поля комментария
:	двоеточие	признак метки, используется и в операторах
.	точка	значение адреса в сегменте данных

$	доллар	в идентификаторах просто пропускается, значение счетчика в сегментах команд
/ * - +	арифметические знаки	используются в арифметических операторах
@	коммерческое а	используется в именах идентификаторов или для обозначения специальной метки
?	вопросительный знак	используется в именах идентификаторов
_	подчеркивание	используется в именах идентификаторов
!	восклицательный знак	используется для объединения нескольких конструкций языка в одной строке
'	апостроф	ограничитель символьных строк
,	запятая	ограничитель операндов машинной команды и директив

31.2.4. Константы языка ассемблера

Числовые константы могут быть в любой из четырех систем счисления, признак системы счисления ставится сразу за константой (без пробела) и может принимать следующие значения:

B - двоичная, O или Q - восьмеричная, D - десятичная, H - шестнадцатеричная. Если признак не указан, константа считается десятичной (это правило можно изменить с помощью директивы трансляции $#). Если шестнадцатеричная константа начинается с буквы, необходимо добавить ноль, чтобы транслятор мог отличить ее от идентификатора.

Примеры: 1234 1234D 1100b 10101011101b 1234H 0FFEH 3377O 0FFFFH 1234d 0fffh -1 0000000000

Символьные константы представляют цепочку символов ASCII в апострофах. Сам апостроф при этом записывается двумя апострофами подряд. Если эти константы являются операндами машинной команды, они могут иметь длину только от одного до четырех символа и воспринимаются просто как 8-32-х битовое число, причем символы идут "в обратном порядке", если считать с 8 младших разрядов.

Только в операторе DB символьная константа может быть большей длины (до 255 символов).

В символьных константах может также использоваться специальный знак "^", который подобно клавише Ctrl гасит в следующем за ним символе 3 старших

бита. Сам знак "^" записывается как "^^". Таким способом можно указывать в символьных строках управляющие коды, например, ^@ - это код 00, а ^M^J - это 0DH и 0AH.

Примеры: 'a', 'Ab''Cd', '', 'I don''t like IBM^M^J'

31.2.5. Идентификаторы ассемблера

Идентификаторы языка - это имена, начинающиеся с буквы или со знаков ?, _ и @ длиной не более 80 символов и состоящие из букв, цифр, специальных знаков ?, @, _, $. Знак "$" просто игнорируется, поэтому идентификаторы "X$1" и "X1" в RASM-KT эквивалентны.

Одиночный символ @ используется для обозначения специальных меток (см. ниже).

Существуют два вида идентификаторов: определенные пользователем и ключевые слова языка.

31.2.6. Идентификаторы, определенные пользователем

Все идентификаторы, определенные пользователем, делятся на метки и переменные (не считая переименованных констант).

Метка - это адрес байта начала инструкции (машинной команды). Метка имеет 2 атрибута:

а) сегмент - показывающий, в каком сегменте описана данная метка;

б) смещение - показывающий, на сколько байтов от начала сегмента сдвинут адрес, обозначаемый данной меткой;

Метка считается описанной в программе, если она указана перед инструкцией, от которой она должна быть отделена двоеточием.

Переменная - это адрес байта в памяти. Каждая переменная имеет три атрибута: сегмент, смещение и тип, показывающий, сколько байтов должно быть обработано при обращении к этой переменной (1, 2, 4, 6, 8, 10 или даже 16);

Тип переменной задается или с помощью директив DB, DW, DD, DE или DP (см ниже) или непосредственно обозначается в команде. Операнды команд сопроцессора и расширенных команд (типа MMX) своего специального типа не имеют (может быть указан любой), а количество обрабатываемых байт определяется явно мнемоникой команды.

31.2.7. Ключевые слова, встроенные в ассемблер

RASM-KT распознает 5 типов ключевых слов:

- инструкции (названия машинных команд);
- директивы языка (указания транслятору);
- операторы языка (встроенные функции);
- имена регистров;
- непосредственное обозначение типа.

31.2.8. Непосредственное обозначение типа

Для непосредственного обозначения типа используются специальные ключевые слова всегда вместе с ключевым словом PTR (PoinTeR):

B или BYTE	обозначает тип со значением 1 (байт);
W или WORD	обозначает тип со значением 2 (слово);
D или DWORD	обозначает тип со значением 4 (двойное слово);
Q или QWORD	обозначает тип со значением 8 (два двойных);
E	обозначает тип со значением 4 (расширенное слово);
P	обозначает тип со значением 6 (полный указатель);

разница между D и E не очень велика и часто можно использовать одно вместо другого. В принципе, D было введено для обозначения пары сегмент: смещение в 16-разрядных режимах адресации, а E это 32-разрядное смещение в 32-разрядном режиме адресации (т.е. E и P аналог W и D для 32-разрядного режима адресации).

Если в команде нужно явно указать тип операнда, то пишется, например:

```
MOV B PTR [BX],0        или
LDS SI, D PTR [BX]      или
LDS ESI, P PTR [EBX]
```

Следует отметить, что BYTE, WORD, DWORD, QWORD практически просто встроенные переименованные числа 1, 2, 4 и 8, которым, в отличии от обычных чисел, приписаны тип байт, тип слово и тип двойное слово. Если, например, написать команду MOV AX, WORD, то это все равно, что написать MOV AX,2.

В отличие от BYTE, WORD и DWORD, ключевые слова B, W, D и P имеют значения не 1, 2 и 4, а равные текущей дате и времени трансляции (соответствуют обращению к функциям Win API), поэтому, если например, в программе имеется директива типа: DW B,W; то в кодах оттранслированной программы появятся 4

байта, содержащие текущий день, месяц и год трансляции этой программы. Можно перевести эти значения и в текстовый вид:

```
Ч1 EQU ((B AND 0FFH)/10)+'0'          ! M1 EQU ((B SHR 8)/10)+'0'
Ч2 EQU ((B AND 0FFH) MOD 10)+'0'  ! M2 EQU ((B SHR 8) MOD 10)+'0'
Г1 EQU W/1000+'0'                          ! Г2 EQU (W MOD 1000)/100+'0'
Г3 EQU (W MOD 100)/10+'0'             ! Г4 EQU (W MOD 10)+'0'
```

а затем использовать в константах программы, например:

```
DB 'Версия ', Ч1,Ч2,'.',M1,M2,'.',Г1,Г2,Г3,Г4,' г.'
```

31.2.9. Переименования в ассемблере

Переменной может быть присвоено другое имя (или даже несколько имен) с помощью директивы EQU.

Пример:
```
my_variable db 0
another_variable equ my_variable
```

Метки также можно переименовывать директивой EQU.

Пример:
```
METKA2: MOV AX,BX
METKA1  EQU METKA2
LOOP METKA1
```

С помощью EQU можно также присваивать имя любому числу и затем использовать это имя в любом месте программы:

```
ЧИСЛО_5 EQU 5
MOV AL, ЧИСЛО_5
```

С помощью EQU можно также присваивать имя любому режиму адресации и затем использовать это имя в любом месте программы:

```
АДРЕС EQU [BX+SI]-100
MOV AL, АДРЕС+1
```

С помощью EQU можно присвоить значение только один раз в программе, исключение - ключевое слово "E", которому с помощью EQU можно присваивать в разных местах программы разные значения:

```
E EQU 1
MOV AL,E
...
E EQU 2
MOV AL, E
```

31.3. Операторы и выражения в ассемблере

Как и в любом ассемблере, в RASM-KT разрешается записывать действия, которые надо выполнить при трансляции. Эти действия представляют собой встроенные функции языка (операторы), используемые в разных комбинациях (выражения).

Операторы используются в операндах команд и директив, при сложных выражениях можно ставить круглые скобки, однако их вложенность не должна превышать 4.

31.3.1. Формальный синтаксис выражений в RASM-KT:

Операнд ::= Операнд_команды │ Операнд_директивы

Операнд_команды ::= Регистр │ Имя │ Число │ Составной_операнд ["[" Косвенный_адрес "]"]

Регистр ::=

```
  "AL"  | "AH"  | "BL"   | "BH"   | "CL"   | "CH"   | "DL"   | "DH"
| "BPL" | "SPL" | "SIL"  | "DIL"
| "R8L" | "R9L" | "R10L" | "R11L" | "R12L" | "R13L" | "R14L" | "R15L"
| "AX"  | "BX"  | "CX"   | "DX"   | "BP"   | "SP"   | "SI"   | "DI"
| "R8W" | "R9W" | "R10W" | "R11W" | "R12W" | "R13W" | "R14W" | "R15W"
| "EAX" | "EBX" | "ECX"  | "EDX"  | "EBP"  | "ESP"  | "ESI"  | "EDI"
| "R8D" | "R9D" | "R10D" | "R11D" | "R12D" | "R13D" | "R14D" | "R15D"
| "RAX" | "RBX" | "RCX"  | "RDX"  | "RBP"  | "RSP"  | "RSI"  | "RDI"
| "R8"  | "R9"  | "R10"  | "R11"  | "R12"  | "R13"  | "R14"  | "R15"
```

Косвенный_адрес ::= База "+" Индекс "*" Масштаб

База ::=

```
  "EAX" | "EBX" | "ECX"  | "EDX"  | "EBP"  | "ESP"  | "ESI"  | "EDI"
| "R8D" | "R9D" | "R10D" | "R11D" | "R12D" | "R13D" | "R14D" | "R15D"
| "RAX" | "RBX" | "RCX"  | "RDX"  | "RBP"  | "RSP"  | "RSI"  | "RDI"
| "R8"  | "R9"  | "R10"  | "R11"  | "R12"  | "R13"  | "R14"  | "R15"
```

Индекс ::=

```
  "EAX" | "EBX" | "ECX"  | "EDX"  | "EBP"  | "ESI"  | "EDI"
| "R8D" | "R9D" | "R10D" | "R11D" | "R12D" | "R13D" | "R14D" | "R15D"
| "RAX" | "RBX" | "RCX"  | "RDX"  | "RBP"  | "RSI"  | "RDI"
| "R8"  | "R9"  | "R10"  | "R11"  | "R12"  | "R13"  | "R14"  | "R15"
```

Масштаб ::= "1" │ "2" │ "4" │ "8"

Операнд_директивы ::= Регистр │ Имя │ Число │ Составной_операнд

Составной_операнд ::= Операция_И { Операция_ИЛИ Операция_И }

Операция_ИЛИ ::= "OR" | "XOR"

Операция_И ::= Операция_НЕ { "AND" Операция_НЕ }

Операция_НЕ ::= "NOT" Операция_НЕ | Отношение

Отношение ::= Бинарная_операция { Операция_отношения Бинарная_операция }

Операция_отношения ::= "EQ" | "LT" | "LE" | "GT" | "GE" | "NE"

Бинарная_операция ::= Мультипликативная_операция { Знак Мультипликативная_операция }

Знак ::= "+" | "-"

Мультипликативная_операция ::= Унарная_операция { Знак_мультипликативной Унарная_операция }

Знак_мультипликативной ::= "*" | "/" | "MOD" | "SHL" | "SHR"

Унарная_операция ::= Знак Унарная_операция | Выражение

Выражение ::= [Сегмент ":"] Адресное_выражение

Сегмент ::= "CS" | "DS" | "ES" | "SS" | "FS" | "GS"

Адресное_выражение ::= Оператор Скобочное_выражение | Скобочное_выражение ["PTR" Скобочное_выражение]

Оператор ::= "SEG" | "OFFSET" | "TYPE" | "LENGTH"

Скобочное_выражение ::= "(" Составной_операнд ")" | "[" Косвенный_адрес "]" | Представление_адреса

Представление_адреса ::= "$" | "." Число |

Символьная_константа_меньше_5 | Имя | Число

Примечание: *в Скобочном_выражении Косвенный_адрес может быть только в Операнде_команды.*

31.3.2. Логические операторы

Операндами логических операторов (a и b) могут быть только числа.

Синтаксис	Результат
a XOR b	Исключающее ИЛИ
a OR b	Логическое ИЛИ
a AND b	Логическое И
NOT a	Логическое НЕТ, инверсия: все нули заменяются на единицы и наоборот

31.3.3. Операторы отношения

Возможные операнды (a и b) - это числа, метки или переменные в одном и том же сегменте.

Синтаксис	Результат
a EQ b	все 1 если a=b иначе 0
a LT b	все 1 если a<b иначе 0
a LE b	все 1 если a<=b иначе 0
a GT b	все 1 если a>b иначе 0
a GE b	все 1 если a>=b иначе 0
a NE b	все 1 если a><b иначе 0

31.3.4. Арифметические операторы

Синтаксис	Результат	Возможные операнды
a + b	сумма a и b	a - число, переменная, метка, относительный адрес, абсолютный адрес, b - те же значения, но из того же сегмента, что и a.
a - b	вычитание b из a	a - число, переменная, метка, относительный адрес, абсолютный адрес, b - те же значения, но из того же сегмента, что и a.
a * b	умножение a на b	a, b - числа
a / b	деление a на b	a, b - числа
a MOD b	остаток от деления a/b	a, b - числа

a SHL b	сдвиг содержимого a влево на число разрядов, записанных в b	a, b - числа
a SHR b	сдвиг содержимого a вправо на число разрядов, записанных в b	a, b - числа

31.3.5. Оператор явного указания сегмента

<сегмент>: <адресное выражение>, где
<сегмент>=CS, DS, SS, ES, FS, GS
Примеры:
MOV AX, ES:[RBX]
MOV CS:BYTE PTR [RSI],0

31.3.6. Операторы создания адресов и обработки переменных

Синтаксис	Результат	Возможные операнды
SEG a	создает число, которое является значением сегмента a или устанавливает факт равенства (неравенства) текущих значений сегментных регистров для отмены генерации лишних префиксов	a - метка, переменная или выражения DS=CS, CS=DS, SS=DS, DS=SS
OFFSET a	создает число, которое является значением смещения a	a - метка, переменная
TYPE a	создает число, которое является типом a, т.е. 1 (для BYTE), 2 (для WORD или NEAR), 4 (для DWORD или FAR или EXTENDED), 6 (для POINTER) и 0 для ABS	a - метка, переменная
LENGTH a	создает число, которое показывает, сколько байт, слов, двойных слов или указателей связано с переменной a, т.е. сколько элементов в директивах DB/DW/DD/DE/DP.	a - переменная
a PTR b	создает виртуальную переменную с типом a и атрибутами b	a= BYTE, WORD, DWORD, W, D, E, P b - адресное выражение

.a	создает переменную со смещением a, сегмент считается сегментом данных	a - число
$	создает метку со значением, равным текущему счетчику. Сегмент этой метки - текущий сегмент	нет операндов

31.3.7. Примеры использования операторов

Логические операторы допускаются только с числами:

```
mask equ 0fch
signbit equ 80h
mov cl, mask and signbit
mov al, not mask
```

Операторы отношения выдают все 1, если условие выполнилось и все 0 в противном случае:

```
limit1 equ 10
limit2 equ 25
mov ax, limit1 lt limit2  ;    (все 1)
mov ax, limit1 gt limit2 ;   (все 0)
```

Арифметические операторы позволяют вычислять адреса внутри данного сегмента:

```
count equ 2
displ equ 5
flag equ 0ffh
mov eax, (offset x1-offset x2)*(offset x1+offset x2)
mov al, flag+1
mov cl, flag+displ
mov bl, displ-count
mov si, 256/3
mov bl, 64/4
buffersize equ 80
mov ax, buffersize * 2
mov cl, +35
mov al, 2--5
mov dl, -12
mov dl, (-1-3)/4
```

Операторы явного указания сегмента:

```
mov ax, ss:dword ptr [ebx]
mov cx, es:array
```

```
movs b ptr [edi], es:[esi]
mov es:b ptr .20h,al
```

Операторы обработки переменных:

```
word_buffer db 0, 0, 0
buffer dw 1, 2, 3, 4, 5
mov ax, length buffer
mov ax, type buffer
mov ax, length word_buffer * type word_buffer
```

Операторы создания переменных:

```
mov b ptr [rbx], 5
mov al, byte ptr [rbx]
inc word ptr .1234h
```

Операторы создания переменных в сегменте данных:

```
mov ax, .0
mov bx, .400h
```

Операторы использования текущего счетчика команд:

```
jmp $+3    ; переход на следующую команду
jmps $+2   ; переход на следующую команду
loop $     ; пауза
```

31.3.8. Приоритет выполнения операторов

Операторы выполняются в следующем порядке:

```
.  $
( ) [ ]
SEG, OFFSET, PTR, TYPE, LENGTH
явное указание сегмента <сегмент>:
унарные операции + -
* / MOD SHL SHR
сложение, вычитание + -
EQ, LT, LE, GT, GE, NE
NOT
AND
XOR, OR
```

31.3.9. Выражения

RASM-KT различает адресные, числовые и скобочные выражения.

Адресные выражения состоят из трех компонент: значения сегмента, типа и значения смещения. Адресные выражения можно вычислять, используя операторы.

Значением числового выражения является число, в эти выражения могут входить только числа.

Скобочные выражения позволяют использовать регистры базы и индекса при адресации.

В скобочные выражения ранее могли входить индексные регистры DI и SI, базовые регистры BX и BP или пара регистров со знаком +, начиная с 80386 в скобочное выражение может входить любой расширенный регистр или пара регистров, один из которых (кроме ESP/RSP) должен быть умножен на масштаб.

Примеры:

```
mov variable[ebx], 0
mov  ax, [ebx+edi]
mov  ax, [rsi]
mov  ax, 10h[rdi]
mov  ax, [esi]+10h
mov  ax, [esi+ebp]-10h+80/2
mov  eax,[rcx+rdi*8]
```

31.4. Конструкции ассемблера RASM-KT

Конструкции языка могут быть инструкциями (машинными командами) или директивами.

Инструкции переводятся в машинный код, директивы управляют работой транслятора RASM-KT.

Каждую конструкцию языка можно разместить на одной строке или расположить метку и комментарий на отдельных строках, а можно наоборот, объединять несколько конструкций на строке с помощью знака "!".

31.4.1. Инструкции RASM-KT

Общий вид инструкции:

[метка:] [префикс] <мнемоническое имя> [<операнд(ы)>] [;<комментарий>]

<метка> - распознается по двоеточию, метка может отсутствовать;

<префикс> - специальный байт (типа LOCK или REP) используется в некоторых командах;

<мнемоническое имя> - определяет код машинной команды;
<операнды> - разделяются запятыми, в инструкции может быть 0, 1 или 2 операнда (3 операнда в командах IMUL, SHLD, SHRD);
<комментарий> - начинается с ";" и продолжается до конца строки. Могут быть строки, где только комментарии.

31.4.2. Использование специальной метки @

Обычно в программе на ассемблере много меток, что затрудняет анализ текста программы. Причем много меток используется только один раз (часто в условном переходе для обхода группы инструкций). Для облегчения анализа структуры программы и выбора имен меток, можно использовать специальную метку из одного символа @, например:

```
    CMP  AX ,0
    JNZ  @
     MOV  BX ,2
@: CMP  AX ,1
...
```

Каждый раз, встречая метку "@:", RASM-KT заменяет ее на «@0000:», «@0001:» и т.д. А, встречая переход на метку "@", RASM-KT заменяет ее на ближайшую «вниз» метку. Таким образом, если в программе есть переход «вниз» и первая ниже метка и есть нужное место, можно всегда использовать метку с одним и тем же именем "@". Можно, конечно, использовать эту метку и для перехода из нескольких мест сразу и после обычных меток, однако это не рекомендуется, так как облегчение анализа текста в этом случае не получится, и мало этого, легко допустить ошибку, на которую RASM-KT не даст диагностики "МЕТКА УЖЕ БЫЛА".

31.5. Директивы RASM-KT

31.5.1. Основные понятия директивы

Директивы RASM-KT управляют процессом трансляции и влияют на размещение по сегментам, выполнение условий трансляции, определение размещения данных, распределение памяти, составление листинга, включение текстов из других файлов.

Общий вид директивы:
[<имя>] <директива> [<операнд(ы)>] [;<комментарий>]

<имя> - имена допустимы для следующих директив: CSEG, DSEG, ESEG, SSEG, FSEG, GSEG, GROUP, DB, DW, DD, DE, DP, RB, RW, RE, RD, RP,

RS, RQ, EQU; Для директив GROUP и EQU имена обязательны, двоеточие после имени ставить нельзя.
<директива> - ключевое слово директивы.

Директивы ассемблера разделяются на следующие группы:

управления сегментами: CSEG, SSEG, DSEG, ESEG, FSEG, GSEG, GROUP
директивы для редактора связей: NAME, PUBLIC, EXTRN, END, INCLUDE
директивы условий трансляции: IF, ELSE, ENDIF
директива определения имен: EQU
директивы размещения данных в памяти: DB, DW, DD, DE, DP, RS, RB, RW, RD, RE, RP, RQ
директивы управления листингом: PAGE, PAGESIZE, PAGEWIDTH, NOLIST, NOIFLIST, NAME
директивы управления исходным текстом и памятью: INCLUDE, ORG
директива выдачи сообщений на экран NAME

31.5.2. Логическое понятие сегмента памяти

Сегментом называется часть адресного пространства памяти, к которой можно обратиться в команде. Величина этого пространства 4 Гбайта, когда под адрес в команде отводятся 32 бита.

Физический адрес памяти получается путем сложения адреса, указанного в команде и адреса, записанного в специальном регистре, называемого сегментным регистром. Раньше каждый бит этого шестнадцатибитового сегментного регистра обозначает один параграф физической памяти (16 байт). Таким образом, физический адрес памяти определяется как:

«адрес в регистре»*16 + «адрес в команде»

Для современных процессоров в защищенном режиме адрес вычисляется иначе, однако все равно в вычислении адреса используется понятия смещение в команде и сегментный регистр.

Адрес, указанный в команде, называется смещением. Легко заметить, что память может быть разбита на множество сегментов и каждый адрес памяти может входить в несколько сегментов.

Кроме этого, процессор одновременно работает с шестью сегментными регистрами:

В сегментном регистре команд (CS) находится адрес, с которого расположены команды выполняемой программы.

В сегментном регистре данных (DS) находится адрес, с которого расположены данные для этой программы.

В сегментном регистре стека (SS) находится адрес, с которого организован стек для этой программы.

В сегментном регистре внешних данных (ES) находится адрес, с которого обычно располагаются данные из других программ, необходимые для данной программы.

В дополнительных сегментных регистрах данных (FS и GS) обычно также находится адрес, с которого располагаются данные из других программ, необходимые для данной программы.

RASM-KT позволяет разбить программу по сегментам или наоборот, объединить данные и команды в один сегмент. Можно делать сегменты любого размера и объединять их в группы.

31.5.3. Директива управления сегментами

Каждая команда или переменная в программе должна входить в какой-либо сегмент. Команды должны располагаться в сегменте команд, но вообще, директивами ассемблера их можно поместить в сегмент любого типа.

Создать сегмент и назвать его можно следующей директивой:

```
[<имя>] <сегмент> [<тип выравнивания>] [<тип объединения>] ['<класс>']
        [<режим>]
```

<сегмент> может быть
CSEG (сегмент команд)
DSEG (сегмент данных)
SSEG (сегмент стека)
ESEG (сегмент внешних данных)
FSEG (дополнительный сегмент данных)
GSEG (дополнительный сегмент данных)

<имя> - должно быть идентификатором по правилам RASM-KT. Если имя опущено, то в соответствии с классом сегмента, ему будет присвоено одно из стандартных имен: CODE, DATA, STACK, EXTRA. Для FSEG и GSES - DATA.

<тип выравнивания> - принимает значения: BYTE, WORD, PARA, PAGE. Он сообщает редактору связей, как расположить данный сегмент в памяти: начиная с текущего байта (BYTE), с границы слова (WORD), с границы параграфа (PARA), с границы страницы (PAGE) т.е. 4096 байт. Заметим, что атрибут PAGE не реализован в редакторе связей LINK-KT. Если тип не указан, сегмент команд начинается с текущего байта, а сегменты других типов - с границы слова.

<тип объединения> - принимает значения: PUBLIC, COMMON и LOCAL. Он сообщает редактору связей как объединять сегменты с данным именем в группу.

Если указан PUBLIC (смежный), или тип отсутствует, редактор расположит сегменты друг за другом с промежутками, заданными типом выравнивания. Каждый сегмент будет адресоваться относительно своего значения сегментного регистра. Если указан тип COMMON (общий), то все сегменты с этим именем будут адресоваться относительно общего значения сегментного регистра (соответствует COMMON-блоку в языках высокого уровня). Если указан тип LOCAL (отдельный), то этот сегмент при редактировании связей не будет объединяться с другими сегментами. Для сегмента этого типа возможно указание абсолютного расположения в памяти, например: D1 DSEG 0FF00H.

<класс> - по умолчанию всем сегментам приписываются 4 класса: CODE, DATA, STACK, EXTRA. Для FSEG и GSES - DATA. Редактор связей в соответствии с классом группирует сегменты в секции. Обратите внимание, что по умолчанию для DSEG, FSEG и GSEG задан один и тот же класса DATA, а значит и данные будут расположены вместе. Однако для данных, описанных в FSEG и GSEG, транслятор автоматически будет генерировать префиксы FS: и GS:.

<режим> - если задан режим RM (Real Mode), предполагается, что все данные в этом сегменте имеют 32-х разрядные адреса и транслятор будет генерировать специальный префикс 67H, при обращении к этим данным. Для команд можно задать режим JM (Jump Mode), этот режим позволяет располагать команды выше 1M. Режим PM (Protect Mode) определяет защищенный режим. По умолчанию предполагается обычный режим реального адреса.

Для режима 64-разрядной адресации обычно сегментные регистры всегда только CS и DS и указывается 8 байтные адреса режимом A8.

Директива определения сегмента может встречаться в программе несколько раз. До следующей директивы все инструкции или данные располагаются в одном сегменте. Если несколько раз встречается директива с одним и тем же именем, то тип выравнивания в ней можно не указывать, однако изменять значение этого типа, заданного в первой директиве, уже нельзя.

Если в начале программы нет директив определения сегментов, считается, что задана директива CODE CSEG PUBLIC BYTE 'CODE'

31.5.4. Директива GROUP

Директива имеет вид: <имя> GROUP <имя1>, <имя2>, <имя3>, ...

Данная директива определяет, как надо объединять сегменты <имя1>, <имя2>, <имя3> ... в блок, называемой группой. Если сегменты объединены в

группу, то можно задавать смещение в командах различных сегментов относительно одного значения сегментного регистра. Обычно это используется, если в программе несколько небольших сегментов, тогда невыгодно адресоваться внутри каждого сегмента относительно своего значения сегментного регистра, когда можно вычислить адрес при трансляции относительно только одного значения и не перезагружать сегментные регистры при работе. Ведь в этом случае нужно меньше команд и выполнение такой группы будет быстрее.

Один и тот же сегмент не может входить в разные группы.

31.5.5. Директива ORG

Директивой ORG <числовое выражение> можно задать значение текущего счетчика внутри данного сегмента.

<числовое выражение> - должно быть определено до того, как встретилась эта директива.

Если данный сегмент не объединяется с другими, директива ORG задает абсолютное смещение от начала сегмента. Однако, если этот сегмент объединяется редактором связей в группы с другими, то данная директива задает смещение не от начала сегмента, а от той части сегмента, которая была обработана до директивы.

31.5.6. Директива END

Директива имеет вид: END [<начальная метка>] и обозначает конец исходного текста. Все строки файла после END игнорируются. Если эта директива опущена, концом текста считается физический конец файла.

<начальная метка> - имеет два назначения:

она определяет данный модуль как главную программу, такая программа может быть только одна.

она определяет, какая программа должна выполняться первой после загрузки.

Если эта метка опущена, то программа выполняется с первой команды в первом сегменте (CSEG).

31.5.7. Директива PUBLIC

Директива вида PUBLIC <имя> [, <имя>], указывает транслятору, что объекты с указанными именами будут использованы в других программах.

На основании этой директивы транслятор создает специальные записи типа PUBDEF в файле .OBJ, куда запишет объекты, описанные как PUBLIC. При работе редактора связей LINK-KT в этих записях будут искаться объекты,

описанные в других модулях как EXTRN. Если при вызове транслятора был указан параметр $LO, то все переменные и метки, независимо от того описаны они как PUBLIC или нет, помещаются в файл .OBJ в специальную запись типа LOCSYM (т.е. становятся доступными для редактора связей и отладки).

Можно совмещать директиву с самой меткой, например: PUBLIC M:

31.5.8. Директива EXTRN

Директива имеет вид: EXTRN <описание объекта>[, <описание объекта>]. Данная директива сообщает редактору связей, что перечисленные объекты описаны в других программах, но будут использованы в этой программе. <Описание объекта> состоит из имени и типа, разделенных двоеточием.

Тип для переменных может быть: BYTE, WORD, DWORD, E, P;

Тип для меток может быть: NEAR, FAR;

Тип для чисел: ABS (число в EXTRN всегда имеет тип WORD);

Типы можно писать одной буквой, т.е. B, W, D, E, P, N, F и A.

Замечание: *типом сегмента внешней переменной (CSEG, SSEG, DSEG, ESEG, FSEG, GSEG) считается тип сегмента, где встретилась директива EXTRN, даже, если реально эта внешняя переменная находится в сегменте другого типа.*

Пример:

```
EXTRN FCB:BYTE,BUFFER:WORD,INIT:FAR,MAX:ABS,ДВ_СЛОВО:D
```

31.5.9. Директива EQU

Эта директива (уже описанная выше) позволяет присваивать имя выражению, стоящему справа от EQU. Формально директива имеет вид:

```
<имя> EQU <числовое выражение>
<имя> EQU <адресное выражение>
<имя> EQU <регистр>
<имя> EQU <мнемоническое имя команды>
```

Выражения должны быть определены к моменту действия директивы.

Примеры:

```
FIVE        EQU 2*2+1
NEXT        EQU BUFFER
СЧЕТЧИК  EQU CX
АДРЕС      EQU [BX+SI]+100-5
СЛОЖИТЬ EQU ADD
```

```
СЛОЖИТЬ AX, СЧЕТЧИК
MOV AX, АДРЕС
```

31.5.10. Директива DB

Директива [<имя>] DB <числовое выражение> [, <числовое выражение>] или [<имя>] DB <строка символов> [, <строка символов>] выделяет память в байтах для записи числовых и строчных констант.

Только в этой директиве строки могут быть длиннее двух байт (до 255 байт). Если в этой директиве указана переменная, то будет взят младший байт ее смещения.

Если после строки символов (после последней кавычки) стоит буква «F», то считается, что это задано число в формате FLOAT (53) или 8 байт в формате IEE-754, т.е. такую константу можно использовать в командам типа FLD64, например:

```
                    DSEG

313233343536        X1 DB '123456789E10'     ; ОБЫЧНАЯ СТРОКА
373839453130
10E9B7FF87D6        X2 DB '-1234567.9989E0'F; ЧИСЛО IEEE-754
32C1
```

Примеры:

```
TEXT DB 'CP/M SYSTEM',
AA DB 'a'+80H
X DB 1, 2, 3, 4, 5
MOV CX, LENGTH TEXT
```

31.5.11. Директива DW

Директива [<имя>] DW <числовое выражение> [, <числовое выражение>] или

[<имя>] DW <символьная константа> [, <символьная константа>] выделяет память в словах (2 байта) под указанные выражения. Данная директива не воспринимает символьные строки длиннее, чем два байта.

Примеры:

```
CNTR DW 0
JMPTAB DW OFFSET SUBR1, SUBR1, SUBR1
DW 1, 2, 3, 4, 5
```

31.5.12. Директива DE

Директива [<имя>] DE <числовое выражение> [, <числовое выражение>] выделяет память в словах (4 байта) под указанные выражения. Данная директива нужна для процессоров 386/486/Pentium для задания смещения 4, а не 2 байта.

Примеры:

```
CNTR DE 0
JMPTAB DE SUBR1, SUBR1, SUBR1
DE 1, 2, 3, 4, 5
```

31.5.13. Директива DD

Директива [<имя>] DD <адресное выражение> [, <адресное выражение>] выделяет 4 байта памяти.

Пример: АДРЕС DD X1, X2, X3+100

Смещение заданного выражения помещается в двух младших байтах, а сегмент в двух старших байтах. Таким образом, можно поместить полный адрес в двух смежных словах. Для процессоров, начиная с 80386 в качестве данных можно задавать числа, и тогда директива DD становится эквивалентной DE.

31.5.14. Директива DP

Директива [<имя>] DP <адресное выражение> [, <адресное выражение>]

выделяет 6 байт памяти под полный указатель переменной (4 байта смещения и 2 байта сегмента), это возможно только для процессоров 386/486/Pentium.

Пример: АДРЕС DP X1, X2, X3+100

31.5.15. Директивы RS, RB, RW, RE, RD, RP, RQ

Данные директивы аналогичны DB, DW, DE, DD, DP, но заполняют выделенную память не данными, а нулями (однако, если это последняя директива в программе, память ничем не заполняется!). Обращение к этим директивам имеет вид: [<имя>] <директива> <числовое выражение>

<числовое выражение> определяет размер выделяемой памяти соответственно в байтах, словах, двойных словах или в полных указателях.

Директива RS работает как RB, но не присваивает получившемуся элементу никакого типа.

Остальные директивы присваивают своим элементам тип BYTE, WORD, DWORD или тип POINTER (6 байт).

Примеры:

```
BUF     RB 80
        RS 4000H
WTAB    RW 122
DWTAB   RD 4
```

Если указать <числовое выражение> равным 0, память не выделяется. Таким образом, можно обращаться к одному адресу как к разным объектам, например, в данном случае к переменной можно обращаться как к байту (X1) и как к слову (X2):

```
X1  RB 0
X2  RW 0
    DB 0,0
```

31.5.16. Директивы условной трансляции

Можно указать RASM-KT какие куски исходного текста нужно пропустить, а какие наоборот, учитывать при трансляции. Директива имеет вид:

```
IF <числовое выражение>
<исходная строка 1>
<исходная строка 2>
...
<исходная строка N>
ELSE
<исходная строка N+1>
<исходная строка N+2>
...
<исходная строка M>
ENDIF
```

Транслятор вычисляет числовое выражение и если оно не 0, то транслируются строки 1-N, если 0, то транслируются строки N+1 - M. Если альтернативной ветви нет и условие равно 0, то пропускается весь блок до ENDIF. Если задать заведомо не выполняющееся условие, то можно записать длинный комментарий до слова ENDIF.

Следует отметить, что условие для этих директив можно задать из командной строки вызова транслятора (с помощью параметра $%).

31.5.17. Директива INCLUDE

Директивой INCLUDE <имя файла> можно включить в исходный текст программы текст из другого файла. Если в <имя файла> расширение опущено, то подразумевается .A86, если диск не указан, файл ищется на текущем диске в текущей папке. Нельзя использовать директиву INCLUDE в файле, который сам был вызван этой директивой.

С помощью директивы INCLUDE можно также указать, какие библиотеки необходимо вызывать по умолчанию при обработке данного модуля редактором связей LINK-KT. Признаком библиотеки, а не включения файла с исходным текстом является знак "%", например:

```
INCLUDE %SERVICE     ; ПРИМЕР ВЫЗОВА БИБЛИОТЕКИ
```

В этом примере при работе редактора связей LINK-KT будет использоваться библиотека SERVICE.L86. В имени библиотеки нельзя указывать диск или расширение. Сами директивы могут располагаться в тексте в любом месте (в том числе и в файлах вызываемых директивой INCLUDE), однако всего их может быть не более 16.

31.5.18. Директива NAME

Директива вида NAME '<имя>' используется для трех целей:

во-первых, с помощью этой директивы можно просто вывести на экран при трансляции произвольный текст (на третьем проходе), в этом случае текст в кавычках должен начинаться знаком "$". Сам знак $ на экран не выводится.

во-вторых, она может задать имя модуля, с которого будет начинаться файл .OBJ (запись THEADR). Имя не должно превышать 40 символов и не должно начинаться со знаков "$" и "#". Если данная директива не использовалась, в качестве имени будет взято имя файла с исходным текстом;

в-третьих, с помощью этой директивы можно задать печать заголовка на каждой странице листинга. Длина строки не должна превышать 30 символов и текст в кавычках должен начинаться знаком "#". Сам знак # в заголовок не выводится.

Пример:

```
IF ПРОЦЕССОР EQ 8086
NAME 'M8086'
NAME '$ВЕРСИЯ ДЛЯ ПРОЦЕССОРА 8086'
NAME '#ПРОГРАММА ДЛЯ ПРОЦЕССОРА 8086'
ELSE
NAME 'M80286'
NAME '$ВЕРСИЯ ДЛЯ ПРОЦЕССОРА 80286'
NAME '#ПРОГРАММА ДЛЯ ПРОЦЕССОРА 80286'
```

ENDIF

31.5.19. Директивы управления листингом

Эти директивы понемногу теряют практическое значение (вместе с бумажными листингами), однако в RASM-KT они сохранены. Заголовок на каждой странице листинга можно задать с помощью директивы NAME.

PAGE. Директива PAGE начинает новую страницу при распечатке листинга.

PAGESIZE. Директива PAGESIZE <числовое выражение> определяет число строк на каждой странице листинга. По умолчанию 0 строк. Если <числовое выражение> равно 0, листинг не будет разбит на страницы.

PAGEWIDTH. Директива PAGEWIDTH <числовое выражение> задает число символов в каждой строке листинга. По умолчанию, если вывод идет не на принтер - 79 символов, если вывод идет на принтер - 120 символов.

NOLIST. Директива NOLIST задает и отменяет печать следующих далее строк исходного текста. Первый раз печать отменяется, второй раз восстанавливается, третий раз опять отменяется и т.д. Таким образом, строки между двумя NOLIST не попадают в листинг.

NOIFLIST. Директива NOIFLIST задает и отменяет печать тех блоков исходного текста, которые не транслировались при выполнении директив условной трансляции. Первый раз печать отменяется, второй раз восстанавливается, третий раз опять отменяется и т.д. Таким образом, не транслированные строки между двумя NOIFLIST не попадают в листинг.

31.6. Специальные макросредства RASM-KT

31.6.1. Определение макросредств

В RASM-KT нет макрорасширений в обычном смысле. То есть нельзя описать цепочку команд, которую транслятор затем будет генерировать в местах вызова, заменяя формальные параметры фактическими.

Однако в RASM-KT можно описать элемент в виде комбинации бит, соответствующей какой-либо команде (или командам) и указать, как в эту

цепочку бит подставить фактические параметры. Транслятор помещает описание таких «макрокодов» в свою внутреннюю таблицу команд (а не в таблицу идентификаторов), и затем обрабатывает их как обычные команды.

Обращение к макрокоманде такое же, как и к обычной инструкции, например:

```
xchg bx,word3
mycode par1,par2
mul ax, word2
...
```

Здесь происходит обращение к макрокоманде MYCODE, которая была описана в программе с двумя формальными параметрами, заменяемых в точке вызова на фактические.

Используя эти средства можно расширить возможности и уже имеющихся команд, если задать имя стандартной команды как макрокоманду. Тело такой макрокоманды будет добавлено к уже имеющимся внутри транслятора стандартным телам. Например, можно расширить команду SHR таким образом, что строка SHR AX будет восприниматься как SHR AX,1. В обычных ассемблерах также можно создать подобную команду с помощью макросредств, например:

```
CodeMacro SHR1 REG
SHR REG,1
EndM
```

Здесь становится видным отличие RASM-KT, у которого это описание выглядит так:

```
CodeMacro SHR X:Eb
DB 0D0H
MODRM 5,X1
EndM
CodeMacro SHR X:Ew
DB 0D1H
MODRM 5,X1
EndM
```

Во-первых, в RASM-KT макрокоманда может иметь такое же имя, как и "встроенная" команда, в этом случае новая комбинация операндов добавляется к уже имеющимся (или заменяет такую же).

Во-вторых, команда формируется по правилам x86 (в данном случае с помощью MODRM-байта) и таким образом можно вводить новые команды, а в обычных ассемблерах пришлось бы формировать цепочку бит в директивах DB или использовать наложение на команды с похожим кодом.

31.6.2. Описание макрокоманды

Общий вид описания:

```
CodeMacro  <имя> [<список формальных параметров>]
<тело макрокоманды>
EndM
```

CodeMacro и EndM - ключевые слова.

<список формальных параметров> - описание от одного до семи параметров, включающее имя параметра, отделенное двоеточием, его спецификацию, тип и область изменения.
<тело макрокоманды> - последовательность директив.

Примеры:

```
CodeMacro AAA
DB 37h
EndM

CodeMacro DIV divisor:Eb
SEGFIX divisor
DB 6Fh
EndM

CodeMacro ESC opcode:Db(0,63),src:Eb
SEGFIX src
DBIT 5(1Bh),3(opcode(3))
EndM
```

31.6.3. Описание формальных параметров макро

Каждый параметр должен быть описан одной из следующих спецификаций:
A - сумматор AX или AL;
C - выражение типа метка;
D - непосредственный операнд;
E - адресное выражение, записанное в регистре или в памяти;
M - адресное выражение, может быть переменным и использовать индексные и базовые регистры в квадратных скобках;
R - один из общих регистров;
S - сегментный регистр;
X - прямое обращение к памяти (при обмене с сумматором);

После спецификации может быть указан тип параметра:
b - байт,
w -слово,
d - двойное слово,
sb - знаковый байт,
e - расширенный операнд (32 разряда),
se - расширенный знаковый байт (32 разряда).
q – 64-разрядный операнд

Если параметр число, то тип b определяет его возможные значения от -256 до 255, если тип w - от -32768 до 32767, а если тип d - от -2**32 до 2**32-1.

После спецификации и типа, может быть указана допустимая область изменения параметра в круглых скобках. Здесь могут быть указаны числа или регистры, или комбинация двух регистров, или двух чисел через запятую.

Примеры описания параметров:

```
CodeMacro IN dst:Aw,port:Rw(DX)
CodeMacro ROR dst:Ew,count:Rb(CL)
CodeMacro ESC opcode:Db(0,63),adds:Eb
```

31.6.4. Директива SEGFIX

Директива вида SEGFIX <имя>, где <имя> - имя формального параметра указывает транслятору, что если фактический параметр при обращении к макрокоманде находится не в сегменте, который подразумевается по умолчанию, например не в DSEG, а в ESEG, то необходимо генерировать специальный префиксный байт перед этой командой. Этот байт и будет являться началом команды, показывая, что в адресе берется сегментный регистр не по умолчанию. Если директивы SEGFIX нет, префиксный байт перед командой не генерируется.

31.6.5. Директива NOSEGFIX

Директива вида NOSEGFIX <регистр>, <имя>, где <регистр> - это один из сегментных регистров ES, DS, CS или SS, а <имя> - имя формального параметра, никакого кода не генерирует, а просто проверяет, что обращение к данному параметру производится с использованием указанного регистра и сообщает об ошибке, если это не так. Такая проверка нужна в цепочечных командах. Пример:

```
CodeMacro MOVS di_ptr:Ew,si_ptr:Ew
NOSEGFIX ES,di_ptr
SEGFIX si_ptr
DB 0A5H
EndM
```

Эта же директива используется и для расширения генерации кодов команд.

Директива NOSEGFIX <число> управляет генерацией префиксов размера 66H/67H для по следующим правилам:

0 – никогда нет префиксов 66H/67H
1 – всегда есть префикс 66H
2 – всегда есть префикс 67H
3 – всегда есть префиксы 66H/67H
4 – операнд всегда 16-разрядный (например, в команде CBW)
5 – никогда нет префикса 66H
6 – никогда нет префикса 67H
7 – всегда есть префикс 66H в режиме x86-64
8 – всегда 32 разрядный операнд по умолчанию (например, в команде LOOP)
9 – никогда нет префикса 66H в режиме x86-64

31.6.6. Директива MODRM

Эта директива определяет вид байта модификации адреса команды (MODRM-байта). Во многих командах этот байт следует за байтом кода команды (opcode-байтом) и определяет, какие регистры и как будут использоваться для получения адресов в команде.

MODRM-байт имеет 3 поля:
mod-поле (2 старших бита),
reg-поле (3 следующих бита),
register-поле (3 последних бита).

Для процессоров, начиная с 80386, кроме MODRM-байта, может быть еще SIB-байт, также входящий в данную директиву.

mod-поле и register-поле определяют 32 возможные комбинации использования регистров (8 регистров + 24 способа индексации).

reg-поле (в зависимости от opcode-байта) содержит или номер регистра или информацию, относящуюся к коду команды.

Кроме MODRM-байта, данная директива генерирует и смещение операнда (если оно необходимо).

Замечание: *полная информация о структуре команд содержится в фирменных руководствах по процессорам.*

Директива имеет вид:
MODRM <имя>, <имя>
MODRM <число>, <имя>
<имя> - имя формального параметра, а <число> - число от 0 до 7.

Примеры:
```
CodeMacro RCR dst:Ew, count:Rb(CL)
SEGFIX dst
DB 0D3h
MODRM 3,dst
EndM

CodeMacro OR dst:Rw,src:Ew
SEGFIX src
DB 0Bh
MODRM dst,src
EndM
```

31.6.7. Директивы RELB и RELW

Эти директивы нужны в командах, которые передают управление на метку, отстоящую от текущего счетчика команд (EIP) на заданное число байт. Для того, чтобы это расстояние вычислялось правильно, адрес, который обозначает метка, должен быть вычислен относительно конца кода команды.

Директивы RELB <имя> и RELW<имя> пропускают соответственно байт или слово от конца кода команды перед размещением операнда <имя> и вычисляют смещение.

Пример:
```
CodeMacro LOOP place:Cb
DB 0E2h
RELB place
EndM
```

31.6.8. Директивы DB, DW и DD внутри макро

Эти директивы аналогичны обычным директивам DB, DW и DD в RASM-KT и позволяют записать сам код команды, а в некоторых случаях ее адресную часть. Директивы могут иметь только один операнд. Для процессоров, начиная с 80386, директива DW генерирует 4 байта кодов.

Пример:

```
CodeMacro XOR dst:Ew,src:Db
SEGFIX dst
DB 81h
MODRM 6,dst
DW src
EndM
```

31.6.9. Директива RE

Директива RE <цифра1 цифра2> задает правила формирования REX-префиксов в командах x86-64 по следующим правилам:

Цифра1 = 9 - не учитывать первый операнд команды в REX-префиксе
Цифра1 = 0 - учитывать первый операнд команд в REX-префиксе
Цифра1 = 1 - учитывать второй операнд команд в REX-префиксе
Цифра2 = 9 - не учитывать второй операнд команды в REX-префиксе
Цифра2 = 0 - учитывать первый операнд команд в REX-префиксе
Цифра2 = 1 - учитывать второй операнд команд в REX-префиксе

31.6.10. Директива DBIT

Данная директива позволяет прямо сформировать цепочку бит, не используя DW, DB, DD или MODRM.

Директива имеет вид: DBIT <список полей>, где <список полей> - перечисление через запятую отдельных элементов битовой комбинации.

Каждое поле начинается с числа, указывающего, сколько бит занимает это поле (от 1 до 8), затем идет собственно код поля или имя параметра в круглых скобках, после параметра может быть указано также в круглых скобках смещение - количество бит, на которые надо сдвинуть этот код вправо.

Пример:

```
CodeMacro DEC dst:Rw
DBIT 5(9h),3(dst(0))
EndM
```

Сгенерированный в данном примере код будет 4Ah, если фактический параметр - это регистр AX.

31.6.11. Встроенные макрокоманды

Для удобства записи имеется ряд встроенных в ассемблер макрокоманд:

Всем командам сдвигов, не имеющим второго операнда, приписывается сдвиг на один: SHL AX эквивалентна SHL AX,1

Команда XLAT без операнда допустима;

В командах PUSH и POP могут быть перечислены несколько регистров через запятую, причем регистры в одной такой команде должны быть однотипны (или только обычные или только сегментные):

PUSH AX, BX, CX, DX заменяется на

PUSH AX ! PUSH BX ! PUSH CX ! PUSH DX

POP DS,ES заменяется на POP DS ! POP ES

В команде MOV для сегментного регистра допустимы константы и другие сегментные регистры:

MOV DS,0 заменяется на PUSH 0 ! POP DS

MOV DS, CS заменяется на PUSH CS ! POP DS

В команде MOV допустимы два операнда-слова памяти:

MOV X,CS:W PTR [BX+DI] заменяется на

PUSH CS:W PTR [BX+DI] ! POP X

В командах IN и OUT можно не указывать операнд DX:

IN AL заменяется на IN AL, DX

OUT AL заменяется на OUT DX, AL

31.7. Транслятор RASM-KT

31.7.1. Основные особенности работы

Транслятор обрабатывает файл с исходным текстом на языке ассемблера RASM-KT за три прохода и создает файл (.OBJ) с перемещаемым объектным кодом в формате объектных модулей фирмы INTEL (Intel Technical Specification 121748-001), доработанный для x86-64).

Транслятор даже может сразу создать выполняемый модуль (.COM) для среды MS DOS, если в исходном тексте нет внешних переменных и не требуется сложная обработка сегментов. При генерации выполняемого модуля, объектный модуль не создается.

Транслятор может создать также файл с таблицей всех переменных и меток (.SYM), в котором перечислены все описанные в программе объекты: переменные, константы (числа) и метки, а также их адреса. По умолчанию файл .SYM не создается.

Транслятор может создать также файл листинга (.LST), в котором находится листинг программы и все сообщения об ошибках. По умолчанию файл .LST не создается.

После окончания своей работы транслятор возвращает операционной системе код ответа, равный числу ошибок при трансляции.

31.7.2. Вызов транслятора RASM-KT

Вызов транслятора RASM-KT осуществляется директивой:

PLINK64 [<путь>] <имя файла>.A86^ [$<параметры>] или
PLINK64 +[<путь>] <имя файла>^ [$<параметры>]

<имя файла> - имя файла с исходным текстом, если расширение имени опущено, нужно впереди ставить знак плюс. Если не указан путь, файл ищется на текущем диске. Пример: PLINK64 d:\bior88.A86 $ oc pc sc

После вызова транслятора на экран выдается информационная картинка с названием ассемблера и номером его версии. Во время работы в верхней строке стандартного окна Windows отображается текущий номер прохода (1, 2 или 3).

По окончании работы транслятора, на экран выдается размер всех секций получившейся программы, число ошибок (если они были) и процент использования памяти, показывающий, какой процент от доступной транслятору памяти использовался под внутреннюю таблицу объектов при трансляции (если больше 10%).

31.7.3. Параметры вызова транслятора RASM-KT

После знака $ можно написать список параметров, которые укажут транслятору, куда выдавать полученные файлы. Параметры файлов имеют вид: td, где t - тип файла и d - буква диска.

RASM-KT различает 4 типа файлов:
O - файл с объектным кодом (.OBJ)
P - файл листинга (.LST)
S - файл таблицы идентификаторов (.SYM)
C - файл выполняемого модуля (.COM)

Буква диска может принимать значения от A до P или обозначатся специальными знаками: X - вывод на экран, Y - вывод на принтер, Z - отмена генерации файла.

Отсутствие параметров файлов эквивалентно строке: $ PZ SZ CZ. Для выполняемого файла или файла объектного кода нельзя указывать букву X или Y.

Пример: PLINK64 IO.A86 $ OZ означает, что исходный файл берется с текущего диска, файл IO.OBJ не генерируется.

Можно также с помощью особого параметра LO включить все локальные объекты в файл .OBJ и, таким образом, сделать их доступными для редактора связей и отладчика.

Для определения множества допустимых команд можно задать параметр - тип процессора:

086 (процессоры 8086 или 8088),
186 (процессор 80186),
286 (процессор 80286),
386 (процессор 80386),
486 (процессор 80486)
586 и 686 (процессор Pentium).

Если в тексте встречается команда, недопустимая для данного типа процессора, транслятор фиксирует ошибку № 28. По умолчанию считается, что трансляция идет для самого старшего процессора (т.е. все команды допустимы).

Пример: PLINK64 TEST.A86 $PX 086

Имеется специальный параметр "@", который выключает контроль типов операндов в арифметических и логических операторах, т.е. при задании этого параметра можно писать команды типа:

MOV AX,OFFSET X1+OFFSET X2

и сообщений об ошибках не будет, однако следует помнить, что ответственность за правильность действий в этом случае лежит на программисте.

Пример: PLINK64 TEST.A86 $PX @

Имеется специальный параметр "#" (RADIX), позволяющий задавать систему счисления для чисел, в которых она не указана явно. После символа "#" ставится тип системы счисления, т.е. H, B, D, Q или O. По умолчанию задан $#D. Примеры:

```
PLINK64 TEST.A86 $ #H
...
MOV AX,10H     ; - операнд 16
MOV AX,10      ; - операнд 16
```

```
PLINK64 TEST.A86 $ #B
...
MOV AX,10H     ; - операнд 16
MOV AX,10      ; - операнд 2
```

Имеется специальный параметр "%", позволяющий ввести значение в текст программы из командной строки. Значение задается десятичным числом без знака от 0 до 65535 и попадает в специальную зарезервированную переменную типа "число" и с именем ??. Если в командной строке параметр % не задавался, зарезервированная переменная ?? имеет значение 0.

Переменную ?? удобно использовать в директивах условной трансляции.

Пример:

```
PLINK64 TEST.A86 $ %128
...
MOV AX, ??          ; - В РЕГИСТР ПЕРЕДАЕТСЯ 128
XX EQU ??+100
MOV BX, XX+100  ; - В РЕГИСТР ПЕРЕДАЕТСЯ 328
IF ?? EQ 0
...
```

Имеется специальный параметр "&", позволяющий заменить все команды INT 3 в программе на команды NOP (код 90H), что иногда удобно при отладке.

Пример:

```
PLINK64 TEST.A86 $ &
...
INT 3  ; - ГЕНЕРИРУЕТСЯ КОД КОМАНДЫ NOP
```

Имеется специальный параметр "*", который отменяет генерацию файла .OBJ и создает файл .COM на текущем диске. Таким образом, вызов

PLINK64 D:TEST.A86 $* эквивалентен вызову PLINK64 D:TEST.A86 $OZ CD.

Имеется специальный параметр "+", который отменяет перевод латинских символов исходного текста в верхний регистр (в большие буквы). Это бывает необходимо при объединении с модулями, написанными на языке Си, так как в тех модулях имена не переводятся в верхний регистр. Не забудьте, что в этом случае все ключевые слова и команды в Вашей программе (и даже командная строка после "+") должны быть обязательно написаны в верхнем регистре. Например: PLINK64 D:TEST.A86 $cx+PX.

Есть специальный параметр "^" (или "~"), который имеет смысл, только если заказан вывод файла .SYM. При данном параметре в файл .SYM включаются только те переменные, переименованные числа и метки, которые описаны в

программе, но к которым нет явного обращения в этой программе. Таким образом, можно проанализировать, какие объекты в программе можно исключить. Пример: PLINK64 D:TEST.A86 $SD^.

Параметр "^", повторенный два раза подряд, выдает в файле .SYM не адреса, а количество раз, которые встречались переименованные числа, метки и переменные в выражениях. Пример: PLINK64 D:TEST.A86 $SD^^. Заметим, что в этом случае фактически трансляция не производится (выполняются только 2 прохода), а количество раз, равное нулю означает, что число, метка или переменная описаны, но явно не используются.

Есть специальный параметр "/", который вместо префиксов 67H в командах вставляет код CCH, делая возможным подключение эмуляторов расширенной памяти (типа EHIMEM.COM), т.е. перехватывать все старые режимы адресации.

Если строку параметров начать двумя знаками $ подряд, например:
PLINK64 TEST.A86 $$PX или
PLINK64 IO.A86 $$,
то на экран не будет выдаваться информационная картинка с названием ассемблера.

31.7.4. Ошибки, выдавемые при работе RASM-KT

Ошибки при работе с файлами

После выдачи этих сообщений работа транслятора прекращается.

НЕТ ФАЙЛА	RASM-KT не может найти файл с исходным текстом или файл директивы INCLUDE на указанном диске.
ДИСК ПОЛОН	При выводе результатов трансляции в файл не хватает места на диске или транслятор запущен в разных окнах
НЕЛЬЗЯ СОЗДАТЬ ФАЙЛ	Скорее всего, транслятор запущен одновременно в нескольких окнах Windows
НЕ ЧИТАЕТСЯ ДИСК	При чтении файла с исходным текстом или файла директивы INCLUDE не найден конец файла, возможно файлы испорчены.
ПЕРЕПОЛНЕНА ТАБЛИЦА	Не хватает места в памяти для таблицы объектов трансляции.
ОШИБКА ВЫЗОВА	Ошибка в командной строке вызова RASM-KT.
МАЛО ПАМЯТИ	Не хватает места для расположения в памяти порции исходного текста (и текста, вызванного через INCLUDE)

Диагностические ошибки при трансляции

Транслятор выдает все сообщения об ошибках на экран, а если заказан листинг, то и в листинг. Если в процессе трансляции были обнаружены ошибки, то после вывода на экран сообщения о первой из них работа приостановится и выдается запрос: ИДТИ ? (Esc-НЕТ)

Если нажать клавиши "Esc" или "0", трансляция прервется, если же нажать любую простую клавишу (например, пробел) работа продолжится и для следующей остановки нужно опять нажать любую клавишу. Если нажать управляющую клавишу (например, курсор вниз), работа продолжится и на следующей ошибке опять остановится.

<<ОШИБКА N: 0>>	НАЧАЛО СТРОКИ НЕИЗВЕСТНО ЧТО	Первый элемент в данной строке не идентификатор, не директива и не команда.
<<ОШИБКА N: 1>>	ЭТО НЕ КОМАНДА И НЕ ДИРЕКТИВА	Первый элемент в данной строке правильный идентификатор, а второй не является названием директивы или команды
<<ОШИБКА N: 2>>	НА МЕСТЕ КОМАНДЫ НЕИЗВЕСТНО ЧТО	Идентификатор в поле команды описан позже, чем к нему обратились или используется еще как метка или переменная.
<<ОШИБКА N: 3>>	ПЕРЕМЕННАЯ УЖЕ БЫЛА	Указанная переменная описана в программе несколько раз. Далее эта переменная считается неописанной.
<<ОШИБКА N: 4>>	МЕТКА УЖЕ БЫЛА	Указанная метка описана в программе несколько раз .Далее эта метка считается неописанной.
<<ОШИБКА N: 5>>	ТАКОЙ КОМАНДЫ НЕТ	Элемент в поле команды не является названием команды.
<<ОШИБКА N: 6>>	ЛИШНЕЕ ПОСЛЕ КОМАНДЫ ОТБРОШЕНО	После операндов инструкции идут еще элементы (но не комментарий), которые RASM-KT игнорирует.
<<ОШИБКА N: 7>>	НЕ ТОТ ТИП У ОПЕРАНДА ИЛИ ИХ ЧИСЛО НЕ ТО	В данной команде неверное число операндов или операнды с неправильным типом.
<<ОШИБКА N: 8>>	ЭТО ВООБЩЕ НЕ ОПЕРАНДЫ	Операнды данной команды или директивы неправильно записаны. Например, в выражении получилось деление на ноль.

<<ОШИБКА N: 9>>	ПРЕФИКС ЕСТЬ, А КОМАНДЫ НЕТ	В данной строке есть префиксная команда, а следующей за ней команды обработки цепочек нет. Пример: REP STOSB - правильно, один REP - ошибка
<<ОШИБКА N:10>>	НЕ ОПИСАННЫЙ ЭЛЕМЕНТ	Операнд в команде не описан или в директиве EQU правый операнд описан позже, чем на него ссылаются.
<<ОШИБКА N:11>>	НЕ ТОТ ОПЕРАНД ДИРЕКТИВЫ	Операнд директивы неправильно записан.
<<ОШИБКА N:12>>	ОЧЕНЬ ДЛИННЫЙ "IF"-ОН ОТБРОШЕН	Число элементов в конструкции IF превосходит максимально допустимое, IF-конструкция игнорируется.
<<ОШИБКА N:13>>	В "IF" ЧТО-ТО НЕ ТО	В конструкции IF элементы не числа или они еще не описаны. IF-конструкция игнорируется.
<<ОШИБКА N:14>>	УЖЕ "ENDIF", А "IF" ЕЩЕ НЕ БЫЛО	Встретилась директива ENDIF, но директивы IF не было.
<<ОШИБКА N:15>>	ЭЛЕМЕНТ ЕЩЕ НЕ ОПИСАН	Указанный идентификатор в директивах ORG, RS, EQU или IF еще не описан.
<<ОШИБКА N:16>>	ЭЛЕМЕНТ УЖЕ БЫЛ В "EQU"	Идентификатор, который стоял в директиве EQU, в программе описывается еще раз. Далее этот элемент считается неописанным.
<<ОШИБКА N:17>>	КОМАНДА ПОПАЛА В ДАННЫЕ	Команда размещается ассемблером не в сегменте команд (т.е. она не может быть выполнена при работе программы).
<<ОШИБКА N:19>>	"INCLUDE" ВНУТРИ "INCLUDE"	Директива INCLUDE в файле, который сам был вызван директивой INCLUDE.
<<ОШИБКА N:20>>	В ВЫРАЖЕНИИ ЧТО-ТО НЕ ТО	Выражение в ассемблере записано неправильно, например неправильный арифметический оператор.
<<ОШИБКА N:21>>	НЕТ ТИПА У ОПЕРАНДА	Для данного операнда команды нет информации о его типе, например: MOV [BX],10.

<<ОШИБКА N:22>>	МЕТКА ВНЕ ГРАНИЦ	Метка, указанная в команде расположена в другом сегменте (что для данной команды недопустимо) или отстоит от данной команды на расстоянии больше 128 байт (что тоже для данной команды недопустимо).
<<ОШИБКА N:23>>	НЕТ СЕГМЕНТА У ОПЕРАНДА	Встретились команды CALLF или JMPF, а значение регистра сегмента команд (CS), которое они используют еще не определено, например директивой CSEG <число>.
<<ОШИБКА N:24>>	В МАКРО ЧТО-ТО НЕ ТО	Ошибки при использовании МАКРО-директив.
<<ОШИБКА N:25>>	УЖЕ "ELSE", А "IF" ЕЩЕ НЕ БЫЛО	Встретилась директива ELSE, а директивы IF не было.
<<ОШИБКА N:26>>	ГДЕ "ENDIF" ?	Была директива IF, а соответствующий ENDIF не найден.
<<ОШИБКА N:28>>	КОМАНДА НЕ ДЛЯ 80x86	Встретились команды недопустимые для заданного типа процессора.

31.7.5. Приложение 1. Пример ассемблерной программы

Задача: найти путь к данным, требуемым для работы программы. Путь задан как параметр среды с определенным именем. Если такого параметра нет – нужно взять путь, с которым был указан запуск программы в командной строке. Если и при запуске программы путь не указан – нужно взять текущую папку.

```
;------------------------------------------------------------------------------
; ??? "???????"                    ПРИМЕР ВЗАИМОДЕЙСТВИЯ АССЕМБЛЕРА И WIN API
;
; ОТДЕЛ ??? СЕКТОР ?            ИСПОЛНИТЕЛЬ: ????????          ЯЗЫК:  RASM-KT
;------------------------------------------------------------------------------
; МОДУЛЬ GETPATH                ВЕРСИЯ 1.0             ДАТА СОЗДАНИЯ 10.07.21
; ИСПОЛЬЗУЕМЫЕ МАТЕРИАЛЫ: ОПИСАНИЕ WIN API
;------------------------------------------------------------------------------
; ВНЕСЕННЫЕ ИСПРАВЛЕНИЯ:  НЕТ
;------------------------------------------------------------------------------
; МОДУЛЬ ИЩЕТ ЗАДАННЫЙ ПУТЬ ИЛИ КАК ПАРАМЕТР СРЕДЫ ИЛИ КАК ПУТЬ ПРИ
; ЗАПУСКЕ EXE ИЛИ КАК ТЕКУЩИЙ ПУТЬ, ПРИ ЭТОМ ОРГАНИЗУЯ:
;
; а) ПОИСК ПУТИ ЗАПУЩЕННОГО EXE-ФАЙЛА
;
; ОПИСАНИЕ НА PL/1
; DCL GINTPATH ENTRY(CHAR(*) VAR);
; DCL ПУТЬ_EXE CHAR(*) VAR;
; ОБРАЩЕНИЕ CALL GINTPATH(ПУТЬ_EXE);

; б) ПОИСК ПУТИ КАК ПАРАМЕТРА СРЕДЫ ПО ЗАДАННОМУ ИМЕНИ

; ОПИСАНИЕ НА PL/1
; DCL GETPATH ENTRY(CHAR(*) VAR,(CHAR(*) VAR);
; DCL (ИМЯ_ПАР,ЗНАЧЕНИЕ_ПАР) CHAR(*) VAR;
; ОБРАЩЕНИЕ CALL GETPATH(ИМЯ_ПАР,*ЗНАЧЕНИЕ_ПАР);

;------------------------------------------------------------------------------

      CSEG A8                ;РАБОТА В 64-РАЗРЯДНОМ РЕЖИМЕ

;----------------- ДОСТАТЬ ПУТЬ, С КОТОРЫМ ВЫЗВАН НАШ EXE -------------------

PUBLIC GINTPATH:             ;НАЗВАНИЕ ПОДПРОГРАММЫ

;---- ДОСТАЕМ КОМАНДНУЮ СТРОКУ ЗАПУСКА ПРОГРАММЫ ---

      MOV   RSI,?COMMAND_LINE ;АДРЕС КОМАНДНОЙ СТРОКИ
      LLODSB                 ;ДОСТАЛИ ДЛИНУ СТРОКИ
      MOVZX ECX,AL           ;ДЛИНА ПУТИ

;---- ПРОВЕРКА НА КАВЫЧКУ ПРИ ЗАПУСКЕ ИЗ ЯРЛЫКА ----

      CMP   B PTR [RSI],'"'  ;УКАЗАНА КАВЫЧКА ?
      JNZ   @                ;НЕТ КАВЫЧКИ
      INC   RSI              ;ПРОПУСТИЛИ КАВЫЧКУ
      SUB   CL,2             ;УЧЛИ ОТКРЫВАЮЩУЮ И ЗАКРЫВАЮЩУЮ

;---- ДОСТАЛИ АДРЕС ОТВЕТА ----
```

```
@:    MOV   RDI,[RBX]+0       ;ПЕРВЫЙ АДРЕС ИЗ СПИСКА ПАРАМЕТРОВ
      MOV   RDX,RDI           ;ЭТО АДРЕС ОТВЕТА, ЗАПОМНИЛИ ЕГО
      INC   RDI               ;ПРОПУСТИЛИ БАЙТ ДЛИНЫ

;---- ОТДЕЛЯЕМ ПУТЬ ОТ ИМЕНИ ФАЙЛА ----

M0001:MOV   AL,[RSI+RCX]-1    ;ОЧЕРЕДНОЙ СИМВОЛ ОТ КОНЦА СТРОКИ
      CMP   AL,'\'            ;ЭТО ПУТЬ?
      JZ    @                 ;ДА, НАЧАЛСЯ ПУТЬ
      CMP   AL,':'            ;ЭТО ДИСК?
      JZ    @                 ;ДА, ДИСК ВХОДИТ В ПУТЬ
      LLOOP M0001             ;ПРОДОЛЖАЕМ АНАЛИЗИРОВАТЬ СТРОКУ
@:    JECXZ @                 ;ВСЯ СТРОКА
      CMP   AL,'\'            ;ЭТО НАЧАЛО ПУТИ?
      JNZ   @                 ;ПУТИ ВООБЩЕ НЕ БЫЛО
      DEC   ECX               ;УЧИТЫВАЕМ САМ СЛЭШ

;---- ПЕРЕПИСЫВАЕМ ОТВЕТ ----

@:    MOV   B PTR [RDX],CL    ;ДЛИНА НАЙДЕННОЙ СТРОКИ
      OR    CL,CL             ;СТРОКА ПУСТАЯ?
      JZ    @                 ;ДА, ПУТИ НЕ БЫЛО ЗАДАНО
      REP   MMOVSB            ;НЕПУСТУЮ СТРОКУ ПЕРЕПИСЫВАЕМ
      RET                     ;ВЫХОДИМ С ОТВЕТОМ

;---- ЕСЛИ ВЫЗВАЛИ БЕЗ ПУТИ – БЕРЕМ ТЕКУЩИЙ ПУТЬ ----

@:    PUSH  RDX               ;ЗАПОМНИЛИ АДРЕС ОТВЕТА
      SUB   RSP,20H           ;ГОТОВИМ СТЕК ДЛЯ Win API
      MOV   RDX,RDI           ;2 ПАРАМЕТР – АДРЕС СТРОКИ ОТВЕТА
      MOV   ECX,80H           ;1 ПАРАМЕТР – ДЛИНА СТРОКИ ОТВЕТА
      CALL  @GETCURRENTDIRECTORYA ;ЗАПРОСИЛИ ТЕКУЩУЮ ПАПКУ ЧЕРЕЗ WinAPI
      ADD   RSP,20H           ;ВЕРНУЛИ СТЕК
      POP   RDX               ;ВОССТАНОВИЛИ АДРЕС ДЛИНЫ СТРОКИ
      MOV   B PTR [RDX],AL    ;ПИШЕМ В НАШ ОТВЕТ ДЛИНУ СТРОКИ
      RET                     ;ВЫХОДИМ С ОТВЕТОМ-СТРОКОЙ

;---------------- ДОСТАТЬ ПУТЬ КАК ЗНАЧЕНИЕ СРЕДЫ ПО ИМЕНИ -------------------

PUBLIC GETPATH:               ;НАЗВАНИЕ ПОДПРОГРАММЫ

      MOV   RSI,[RBX]+0       ;АДРЕС ПЕРВОГО ПАРАМЕТРА
      LLODSB                  ;ДОСТАЛИ ДЛИНУ ИМЕНИ
      MOVZX ECX,AL            ;ЗАПОМНИЛИ ДЛИНУ
      MOV   EDI,OFFSET ИМЯ    ;ВНУТРЕННЯ ПЕРЕМЕННАЯ
      REP   MMOVSB            ;ПЕРЕПИСАЛИ СЕБЕ ИМЯ ПАРАМЕТРА
      MOV   AL,0
      SSTOSB                  ;ПОСЛЕ ИМЕНИ ДОПИСЫВАЕМ НОЛЬ

      PUSH  RBX               ;ЗАПОМНИЛИ АДРЕСА ПАРАМЕТРОВ
      SUB   RSP,20H           ;ГОТОВИМ СТЕК ДЛЯ Win API
      MOV   R8D,LENGTH ЗНАЧЕНИЕ ;ДЛИНА ОТВЕТА ДЛЯ Win API
      MOV   EDX,OFFSET ЗНАЧЕНИЕ ;АДРЕС ОТВЕТА ДЛЯ Win API
      MOV   ECX,OFFSET ИМЯ    ;АДРЕС ВНУТРЕННЕЙ ПЕРЕМЕННОЙ
      CALL  @GETENVIRONMENTVARIABLEA ;ОБРАЩАЕМСЯ К Win API
      ADD   RSP,20H           ;ВОССТАНОВИЛИ СТЕК

      POP   RBX               ;ВОССТАНОВИЛИ АДРЕСА ПАРАМЕТРОВ
      ADD   RBX,8             ;АДРЕС ВТОРОГО ПАРАМЕТРА
```

```
      OR    EAX,EAX           ;ЗНАЧЕНИЯ С ТАКИМ ИМЕНЕМ НЕТ?
      JJZ   GINTPATH          ;ТОГДА КАК ОТВЕТ БЕРЕМ ТЕКУЩИЙ ПУТЬ EXE
      MOV   ECX,EAX           ;ДЛИНА СТРОКИ ОТВЕТА
      MOV   ESI,OFFSET ЗНАЧЕНИЕ ;АДРЕС СТРОКИ ОТВЕТА
      MOV   RDI,[RBX]         ;АДРЕС НАШЕЙ СТРОКИ-ОТВЕТА (2 ПАРАМЕТР)
      SSTOSB                  ;ДЛИНА ОТВЕТА
      REP   MMOVSB            ;ПЕРЕПИСЫВАЕМ ОТВЕТ
      RET                     ;ВЫХОДИМ С ОТВЕТОМ

      DSEG                    ;СЕГМЕНТ ДАННЫХ

ИМЯ       RB 20               ;ВНУТРЕННЯЯ ПЕРЕМЕННАЯ ДЛЯ ИМЕНИ
ЗНАЧЕНИЕ  RB 100              ;ВНУТРЕННЯЯ ПЕРЕМЕННАЯ ДЛЯ ЗНАЧЕНИЯ

EXTRN ?COMMAND_LINE:Q         ;ВНЕШНЯЯ ВСТРОЕННАЯ ПЕРЕМЕННАЯ

;---- ДАННЫЕ ДЛЯ ОБРАЩЕНИЯ К WIN API ----
DGROUP GROUP GETENVIRONMENTVARIABLEA,GETCURRENTDIRECTORYA

GETENVIRONMENTVARIABLEA   DSEG COMMON  ;ЗАПОЛНЯЕТ WINDOWS ПРИ ЗАПУСКЕ
@GETENVIRONMENTVARIABLEA  RQ 1

GETCURRENTDIRECTORYA      DSEG COMMON  ;ЗАПОЛНЯЕТ WINDOWS ПРИ ЗАПУСКЕ
@GETCURRENTDIRECTORYA     RQ 1
```

31.7.6. Приложение 2. Мнемокоды команд ассемблера RASM-KT

В данном приложении приведены все допустимые в RASM-KT мнемокоды инструкций и примеры их операндов (включая встроенные расширения). Предполагается, что процессор работает в 64-х разрядном режиме.

```
                    ;------------- ADC -------------
66411308            ADC   CX,W PTR [R8]
41130C24            ADC   ECX,D PTR [R12]
41130C24            ADC   ECX,D PTR [R12]
44121B              ADC   R11L,B PTR [RBX]
66411108            ADC   W PTR [R8],CX
41110C24            ADC   D PTR [R12],ECX
4E112CE3            ADC   Q PTR [RBX+R12*8],R13
44101B              ADC   B PTR [RBX],R11L
66418310FF          ADC   W PTR [R8],0FFFFH
41831424FF          ADC   D PTR [R12],0FFFFFFFFH
4A8314E3FF          ADC   Q PTR [RBX+R12*8],0FFFFFFFFH
66153412            ADC   AX,1234H
6683D012            ADC   AX,12H
83D012              ADC   EAX,12H
4883D012            ADC   RAX,12H
1578563412          ADC   EAX,12345678H
481578563412        ADC   RAX,12345678H
1412                ADC   AL,12H
664181103412        ADC   W PTR [R8],1234H
6641831012          ADC   W PTR [R8],12H
4183142412          ADC   D PTR [R12],12H
4A8314E312          ADC   Q PTR [RBX+R12*8],12H
4181142478563412    ADC   D PTR [R12],12345678H
4A8114E378563412    ADC   Q PTR [RBX+R12*8],12345678H
801312              ADC   B PTR [RBX],12H

                    ;------------- ADD -------------
66410308            ADD   CX,W PTR [R8]
41030C24            ADD   ECX,D PTR [R12]
4E032CE3            ADD   R13,Q PTR [RBX+R12*8]
44021B              ADD   R11L,B PTR [RBX]
66410108            ADD   W PTR [R8],CX
41010C24            ADD   D PTR [R12],ECX
4E012CE3            ADD   Q PTR [RBX+R12*8],R13
44001B              ADD   B PTR [RBX],R11L
66418300FF          ADD   W PTR [R8],0FFFFH
41830424FF          ADD   D PTR [R12],0FFFFFFFFH
4A8304E3FF          ADD   Q PTR [RBX+R12*8],0FFFFFFFFH
66053412            ADD   AX,1234H
6683C012            ADD   AX,12H
83C012              ADD   EAX,12H
4883C012            ADD   RAX,12H
0578563412          ADD   EAX,12345678H
480578563412        ADD   RAX,12345678H
0412                ADD   AL,12H
664181003412        ADD   W PTR [R8],1234H
6641830012          ADD   W PTR [R8],12H
4183042412          ADD   D PTR [R12],12H
4A8304E312          ADD   Q PTR [RBX+R12*8],12H
418104247856        ADD   D PTR [R12],12345678H
4A8104E378563412    ADD   Q PTR [RBX+R12*8],12345678H
800312              ADD   B PTR [RBX],12H

                    ;------------- AND -------------
66412308            AND   CX,W PTR [R8]
41230C24            AND   ECX,D PTR [R12]
4E232CE3            AND   R13,Q PTR [RBX+R12*8]
44221B              AND   R11L,B PTR [RBX]
662108              AND   W PTR [R8],CX
210C24              AND   D PTR [R12],ECX
49212CE3            AND   Q PTR [RBX+R12*8],R13
44201B              AND   B PTR [RBX],R11L
66418320FF          AND   W PTR [R8],0FFFFH
41832424FF          AND   D PTR [R12],0FFFFFFFFH
```

```
4A8324E3FF          AND   Q PTR [RBX+R12*8],0FFFFFFFFH
66253412            AND   AX,1234H
6683E012            AND   AX,12H
83E012              AND   EAX,12H
4883E012            AND   RAX,12H
2578563412          AND   EAX,12345678H
482578563412        AND   RAX,12345678H
2412                AND   AL,12H
664181203412        AND   W PTR [R8],1234H
6641832012          AND   W PTR [R8],12H
4183242412          AND   D PTR [R12],12H
4A8324E312          AND   Q PTR [RBX+R12*8],12H
4181242478563412    AND   D PTR [R12],12345678H
4A8124E378563412    AND   Q PTR [RBX+R12*8],12345678H
802312              AND   B PTR [RBX],12H

                    ;------------- BSF -------------
66410FBC08          BSF   CX,W PTR [R8]
410FBC0C24          BSF   ECX,D PTR [R12]
4E0FBC2CE3          BSF   R13,Q PTR [RBX+R12*8]

                    ;------------- BSR -------------
66410FBD08          BSR   CX,W PTR [R8]
410FBD0C24          BSR   ECX,D PTR [R12]
4E0FBD2CE3          BSR   R13,Q PTR [RBX+R12*8]

                    ;------------ BSWAP ------------
0FC9                BSWAP   ECX
4D0FCD              BSWAP   R13

                    ;-------------- BT -------------
66410FA308          BT   W PTR [R8],CX
410FA30C24          BT   D PTR [R12],ECX
4E0FA32CE3          BT   Q PTR [RBX+R12*8],R13
66410FBA2002        BT   W PTR [R8],2
410FBA242402        BT   D PTR [R12],2
4A0FBA24E33E        BT   Q PTR [RBX+R12*8],3EH

                    ;------------- BTC -------------
66410FBB08          BTC   W PTR [R8],CX
410FBB0C24          BTC   D PTR [R12],ECX
4E0FBB2CE3          BTC   Q PTR [RBX+R12*8],R13
66410FBA3802        BTC   W PTR [R8],2
410FBA3C2402        BTC   D PTR [R12],2
4A0FBA3CE33E        BTC   Q PTR [RBX+R12*8],3EH

                    ;------------- BTR -------------
66410FB308          BTR   W PTR [R8],CX
410FB30C24          BTR   D PTR [R12],ECX
4E0FB32CE3          BTR   Q PTR [RBX+R12*8],R13
66410FBA3002        BTR   W PTR [R8],2
410FBA342402        BTR   D PTR [R12],2
4A0FBA34E33E        BTR   Q PTR [RBX+R12*8],3EH

                    ;------------- BTS -------------
66410FAB08          BTS   W PTR [R8],CX
410FAB0C24          BTS   D PTR [R12],ECX
4E0FAB2CE3          BTS   Q PTR [RBX+R12*8],R13
66410FBA2802        BTS   W PTR [R8],2
410FBA2C2402        BTS   D PTR [R12],2
4A0FBA2CE33E        BTS   Q PTR [RBX+R12*8],3EH

                    ;------------- CALL ------------
E8E9FDFFFF          CALL   M1
E8E4FDFFFF          CALL   M1
FFD1                CALL   ECX
4DFFD5              CALL   R13
41FF1424            CALL   D PTR [R12]
4AFF14E3            CALL   Q PTR [RBX+R12*8]

                    ;------------- CBW -------------
6698                CBW

                    ;------------ CCMPSB -----------
```

```
A6                  CCMPSB

                    ;------------ CCMPSD -----------
A7                  CCMPSD

                    ;------------ CCMPSW -----------
66A7                CCMPSW

                    ;------------ CCMPSQ -----------
6648A7              CCMPSQ

                    ;------------- CDQ -------------
99                  CDQ

                    ;------------- CDQE ------------
4899                CDQE

                    ;------------- CLC -------------
F8                  CLC

                    ;------------- CLD -------------
FC                  CLD

                    ;------------- CLI -------------
FA                  CLI

                    ;------------- CLTS ------------
0F06                CLTS

                    ;------------- CMC -------------
F5                  CMC

                    ;------------ CMOVA ------------
66410F4708          CMOVA   CX,W PTR [R8]
410F470C24          CMOVA   ECX,D PTR [R12]
4E0F472CE3          CMOVA   R13,Q PTR [RBX+R12*8]

                    ;------------ CMOVAE -----------
66410F4308          CMOVAE   CX,W PTR [R8]
410F430C24          CMOVAE   ECX,D PTR [R12]
4E0F432CE3          CMOVAE   R13,Q PTR [RBX+R12*8]

                    ;------------ CMOVB ------------
66410F4208          CMOVB   CX,W PTR [R8]
410F420C24          CMOVB   ECX,D PTR [R12]
4E0F422CE3          CMOVB   R13,Q PTR [RBX+R12*8]

                    ;------------ CMOVBE -----------
66410F4608          CMOVBE   CX,W PTR [R8]
410F460C24          CMOVBE   ECX,D PTR [R12]
4E0F462CE3          CMOVBE   R13,Q PTR [RBX+R12*8]

                    ;------------ CMOVC ------------
66410F4208          CMOVC   CX,W PTR [R8]

                    ;------------ CMOVE ------------
66410F4408          CMOVE   CX,W PTR [R8]
410F440C24          CMOVE   ECX,D PTR [R12]
4E0F442CE3          CMOVE   R13,Q PTR [RBX+R12*8]

                    ;------------ CMOVG ------------
66410F4F08          CMOVG   CX,W PTR [R8]
410F4F0C24          CMOVG   ECX,D PTR [R12]
4E0F4F2CE3          CMOVG   R13,Q PTR [RBX+R12*8]

                    ;------------ CMOVGE -----------
66410F4D08          CMOVGE   CX,W PTR [R8]
410F4D0C24          CMOVGE   ECX,D PTR [R12]
4E0F4D2CE3          CMOVGE   R13,Q PTR [RBX+R12*8]

                    ;------------ CMOVL ------------
66410F4C08          CMOVL   CX,W PTR [R8]
410F4C0C24          CMOVL   ECX,D PTR [R12]
4E0F4C2CE3          CMOVL   R13,Q PTR [RBX+R12*8]
```

```
                ;------------ CMOVLE -----------
66410F4E08      CMOVLE   CX,W PTR [R8]
410F4E0C24      CMOVLE   ECX,D PTR [R12]
4E0F4E2CE3      CMOVLE   R13,Q PTR [RBX+R12*8]

                ;------------ CMOVNA -----------
66410F4608      CMOVNA   CX,W PTR [R8]

                ;----------- CMOVNAE -----------
66410F4208      CMOVNAE   CX,W PTR [R8]

                ;------------ CMOVNB -----------
66410F4308      CMOVNB   CX,W PTR [R8]

                ;----------- CMOVNBE -----------
66410F4708      CMOVNBE   CX,W PTR [R8]

                ;------------ CMOVNC -----------
66410F4308      CMOVNC   CX,W PTR [R8]

                ;------------ CMOVNE -----------
66410F4508      CMOVNE   CX,W PTR [R8]
410F450C24      CMOVNE   ECX,D PTR [R12]
4E0F452CE3      CMOVNE   R13,Q PTR [RBX+R12*8]

                ;------------ CMOVNG -----------
66410F4E08      CMOVNG   CX,W PTR [R8]

                ;----------- CMOVNGE -----------
66410F4C08      CMOVNGE   CX,W PTR [R8]

                ;------------ CMOVNL -----------
66410F4D08      CMOVNL   CX,W PTR [R8]

                ;----------- CMOVNLE -----------
66410F4F08      CMOVNLE   CX,W PTR [R8]

                ;------------ CMOVNO -----------
66410F4108      CMOVNO   CX,W PTR [R8]
410F410C24      CMOVNO   ECX,D PTR [R12]
4E0F412CE3      CMOVNO   R13,Q PTR [RBX+R12*8]

                ;------------ CMOVNP -----------
66410F4B08      CMOVNP   CX,W PTR [R8]
410F4B0C24      CMOVNP   ECX,D PTR [R12]
4E0F4B2CE3      CMOVNP   R13,Q PTR [RBX+R12*8]

                ;------------ CMOVNS -----------
66410F4908      CMOVNS   CX,W PTR [R8]
410F490C24      CMOVNS   ECX,D PTR [R12]
4E0F492CE3      CMOVNS   R13,Q PTR [RBX+R12*8]

                ;------------ CMOVNZ -----------
66410F4508      CMOVNZ   CX,W PTR [R8]

                ;------------ CMOVO ------------
66410F4008      CMOVO   CX,W PTR [R8]
410F400C24      CMOVO   ECX,D PTR [R12]
4E0F402CE3      CMOVO   R13,Q PTR [RBX+R12*8]

                ;------------ CMOVP ------------
66410F4A08      CMOVP   CX,W PTR [R8]
410F4A0C24      CMOVP   ECX,D PTR [R12]
4E0F4A2CE3      CMOVP   R13,Q PTR [RBX+R12*8]

                ;------------ CMOVPE -----------
66410F4A08      CMOVPE   CX,W PTR [R8]

                ;------------ CMOVPO -----------
66410F4B08      CMOVPO   CX,W PTR [R8]

                ;------------ CMOVS ------------
66410F4808      CMOVS   CX,W PTR [R8]
```

```
410F480C24          CMOVS   ECX,D PTR [R12]
4E0F482CE3          CMOVS   R13,Q PTR [RBX+R12*8]

                    ;------------ CMOVZ ------------
66410F4408          CMOVZ   CX,W PTR [R8]

                    ;------------- CMP -------------
66413B08            CMP   CX,W PTR [R8]
413B0C24            CMP   ECX,D PTR [R12]
4E3B2CE3            CMP   R13,Q PTR [RBX+R12*8]
443A1B              CMP   R11L,B PTR [RBX]
66413908            CMP   W PTR [R8],CX
41390C24            CMP   D PTR [R12],ECX
4E392CE3            CMP   Q PTR [RBX+R12*8],R13
44381B              CMP   B PTR [RBX],R11L
66418338FF          CMP   W PTR [R8],0FFFFH
41833C24FF          CMP   D PTR [R12],0FFFFFFFFH
4A833CE3FF          CMP   Q PTR [RBX+R12*8],0FFFFFFFFH
663D3412            CMP   AX,1234H
6683F812            CMP   AX,12H
83F812              CMP   EAX,12H
4883F812            CMP   RAX,12H
3D78563412          CMP   EAX,12345678H
483D78563412        CMP   RAX,12345678H
3C12                CMP   AL,12H
664181383412        CMP   W PTR [R8],1234H
6641833812          CMP   W PTR [R8],12H
41833C2412          CMP   D PTR [R12],12H
4A833CE312          CMP   Q PTR [RBX+R12*8],12H
41813C2478563412    CMP   D PTR [R12],12345678H
4A813CE378563412    CMP   Q PTR [RBX+R12*8],12345678H
803B12              CMP   B PTR [RBX],12H

                    ;------------- CMPS ------------
662643A7            CMPS   ES:W PTR [R11+R12*2],ES:W PTR [R11+R12*2]
A7                  CMPS   D PTR [RBX+RBP*2],D PTR [RBX+RBP*2]
48A7                CMPS   Q PTR [RBX+RSI*8],Q PTR [RBX+RSI*8]
26A6                CMPS   ES:B PTR [RBX+RSI*2]+1,ES:B PTR [RBX+RSI*2]+1

                    ;------------ CMPSB ------------
67A6                CMPSB

                    ;------------ CMPSD ------------
67A7                CMPSD

                    ;------------ CMPSW ------------
6667A7              CMPSW

                    ;----------- CMPXCHG -----------
440FB01B            CMPXCHG   R11L,B PTR [RBX]
66410FB108          CMPXCHG   CX,W PTR [R8]
410FB10C24          CMPXCHG   ECX,D PTR [R12]
4E0FB12CE3          CMPXCHG   R13,Q PTR [RBX+R12*8]
440FB01B            CMPXCHG   B PTR [RBX],R11L
66410FB108          CMPXCHG   W PTR [R8],CX
410FB10C24          CMPXCHG   D PTR [R12],ECX
4E0FB12CE3          CMPXCHG   Q PTR [RBX+R12*8],R13

                    ;---------- CMPXCHG8B ----------
0FC7046B            CMPXCHG8B   D PTR [RBX+RBP*2]

                    ;---------- CMPXCHG16B ---------
480FC704F3          CMPXCHG16B   Q PTR [RBX+RSI*8]

                    ;------------ CPUID ------------
0FA2                CPUID

                    ;------------- CWD -------------
6699                CWD

                    ;------------- CWDE ------------
98                  CWDE

                    ;------------- DAA -------------
```

```
                    ;------------- DAS ------------

                    ;------------- DEC ------------
66FFC9              DEC   CX
FFC9                DEC   ECX
6641FF08            DEC   W PTR [R8]
41FF0C24            DEC   D PTR [R12]
4AFF0CE3            DEC   Q PTR [RBX+R12*8]
FE0B                DEC   B PTR [RBX]

                    ;------------- DIV ------------
6641F730            DIV   W PTR [R8]
41F73424            DIV   D PTR [R12]
4AF734E3            DIV   Q PTR [RBX+R12*8]
F633                DIV   B PTR [RBX]

                    ;------------- EMMS -----------
0F77                EMMS

                    ;------------ ENTER -----------
C8341212            ENTER   1234H,12H
C8120012            ENTER   12H,12H

                    ;------------ F2XM1 -----------
D9F0                F2XM1

                    ;------------- FABS -----------
D9E1                FABS

                    ;------------- FADD -----------
DCC3                FADD   ST3,ST
D8C3                FADD   ST,ST3
DEC1                FADD

                    ;------------ FADD32 ----------
42D8447B05          FADD32   [RBX+R15*2]+5

                    ;------------ FADD64 ----------
42DC447B05          FADD64   [RBX+R15*2]+5

                    ;------------ FADDP -----------
DEC3                FADDP   ST3,ST

                    ;------------- FBLD -----------
42DF647B05          FBLD   [RBX+R15*2]+5

                    ;------------ FBSTP -----------
42DF747B05          FBSTP   [RBX+R15*2]+5

                    ;------------- FCHS -----------
D9E0                FCHS

                    ;------------ FCMOVB ----------
DAC3                FCMOVB   ST,ST3

                    ;------------ FCMOVE ----------
DACB                FCMOVE   ST,ST3

                    ;----------- FCMOVBE ----------
DAD3                FCMOVBE   ST,ST3

                    ;------------ FCMOVU ----------
DADB                FCMOVU   ST,ST3

                    ;----------- FCMOVNB ----------
DBC3                FCMOVNB   ST,ST3

                    ;----------- FCMOVNE ----------
DBCB                FCMOVNE   ST,ST3

                    ;----------- FCMOVNBE ---------
DBD3                FCMOVNBE   ST,ST3
```

```
                    ;----------- FCMOVNU -----------
DBDB                FCMOVNU   ST,ST3

                    ;------------ FCLEX ------------
9BDBE2              FCLEX

                    ;------------- FCOM ------------
D8D3                FCOM   ST3
D8D1                FCOM

                    ;------------ FCOM32 -----------
42D8547B05          FCOM32   [RBX+R15*2]+5

                    ;----------- FCOM32P -----------
42D85C7B05          FCOM32P   [RBX+R15*2]+5

                    ;------------ FCOM64 -----------
42DC547B05          FCOM64   [RBX+R15*2]+5

                    ;----------- FCOM64P -----------
42DC5C7B05          FCOM64P   [RBX+R15*2]+5

                    ;------------ FCOMI ------------
DBF3                FCOMI   ST,ST3

                    ;------------ FCOMIP -----------
DFF3                FCOMIP   ST,ST3

                    ;------------ FCOMP ------------
D8DB                FCOMP   ST3
D8D9                FCOMP

                    ;------------ FCOMPP -----------
DED9                FCOMPP

                    ;------------- FCOS ------------
D9FF                FCOS

                    ;----------- FDECSTP -----------
D9F6                FDECSTP

                    ;------------ FDISI ------------
9BDBE1              FDISI

                    ;------------- FDIV ------------
DCFB                FDIV   ST3,ST
D8F3                FDIV   ST,ST3
DEF9                FDIV

                    ;------------ FDIV32 -----------
42D8747B05          FDIV32   [RBX+R15*2]+5

                    ;------------ FDIV64 -----------
42DC747B05          FDIV64   [RBX+R15*2]+5

                    ;------------ FDIVP ------------
DEFB                FDIVP   ST3,ST

                    ;------------ FDIVR ------------
DCF3                FDIVR   ST3,ST
D8FB                FDIVR   ST,ST3
DEF1                FDIVR

                    ;----------- FDIVR32 -----------
42D87C7B05          FDIVR32   [RBX+R15*2]+5

                    ;----------- FDIVR64 -----------
42DC7C7B05          FDIVR64   [RBX+R15*2]+5

                    ;------------ FDIVRP -----------
DEF3                FDIVRP   ST3,ST

                    ;------------- FDUP ------------
D9C0                FDUP
```

```
                ;------------- FENI ------------
9BDBE0          FENI

                ;------------ FFREE ------------
DDC3            FFREE   ST3

                ;----------- FIADD16 -----------
42DE447B05      FIADD16   [RBX+R15*2]+5

                ;----------- FIADD32 -----------
42DA447B05      FIADD32   [RBX+R15*2]+5

                ;----------- FICOM16 -----------
42DE547B05      FICOM16   [RBX+R15*2]+5

                ;----------- FICOM16P ----------
42DE5C7B05      FICOM16P   [RBX+R15*2]+5

                ;----------- FICOM32 -----------
42DA547B05      FICOM32   [RBX+R15*2]+5

                ;----------- FICOM32P ----------
42DA5C7B05      FICOM32P   [RBX+R15*2]+5

                ;----------- FIDIV16 -----------
42DE747B05      FIDIV16   [RBX+R15*2]+5

                ;----------- FIDIV32 -----------
42DA747B05      FIDIV32   [RBX+R15*2]+5

                ;----------- FIDIVR16 ----------
42DE7C7B05      FIDIVR16   [RBX+R15*2]+5

                ;----------- FIDIVR32 ----------
42DA7C7B05      FIDIVR32   [RBX+R15*2]+5

                ;------------ FILD16 -----------
42DF447B05      FILD16   [RBX+R15*2]+5

                ;------------ FILD32 -----------
42DB447B05      FILD32   [RBX+R15*2]+5

                ;------------ FILD64 -----------
42DF6C7B05      FILD64   [RBX+R15*2]+5

                ;----------- FIMUL16 -----------
42DE4C7B05      FIMUL16   [RBX+R15*2]+5

                ;----------- FIMUL32 -----------
42DA4C7B05      FIMUL32   [RBX+R15*2]+5

                ;----------- FINCSTP -----------
D9F7            FINCSTP

                ;------------ FINIT ------------
9BDBE3          FINIT

                ;------------ FIST16 -----------
42DF547B05      FIST16   [RBX+R15*2]+5

                ;----------- FIST16P -----------
42DF5C7B05      FIST16P   [RBX+R15*2]+5

                ;------------ FIST32 -----------
42DB547B05      FIST32   [RBX+R15*2]+5

                ;----------- FIST32P -----------
42DB5C7B05      FIST32P   [RBX+R15*2]+5

                ;----------- FIST64P -----------
42DF7C7B05      FIST64P   [RBX+R15*2]+5

                ;----------- FISUB16 -----------
```

```
42DE647B05          FISUB16   [RBX+R15*2]+5

                    ;----------- FISUB32 -----------
42DA647B05          FISUB32   [RBX+R15*2]+5

                    ;----------- FISUBR16 ----------
42DE6C7B05          FISUBR16   [RBX+R15*2]+5

                    ;----------- FISUBR32 ----------
42DA6C7B05          FISUBR32   [RBX+R15*2]+5

                    ;------------ FLD -------------
D9C3                FLD   ST3

                    ;------------ FLD1 ------------
D9E8                FLD1

                    ;------------ FLD32 -----------
42D9447B05          FLD32   [RBX+R15*2]+5

                    ;------------ FLD64 -----------
42DD447B05          FLD64   [RBX+R15*2]+5

                    ;------------ FLD80 -----------
42DB6C7B05          FLD80   [RBX+R15*2]+5

                    ;------------ FLDCW -----------
2643D92C63          FLDCW   ES:W PTR [R11+R12*2]

                    ;------------ FLDENV ----------
42D9647B05          FLDENV   [RBX+R15*2]+5

                    ;------------ FLDL2E ----------
D9EA                FLDL2E

                    ;------------ FLDL2T ----------
D9E9                FLDL2T

                    ;------------ FLDLG2 ----------
D9EC                FLDLG2

                    ;------------ FLDLN2 ----------
D9ED                FLDLN2

                    ;------------ FLDPI -----------
D9EB                FLDPI

                    ;------------ FLDZ ------------
D9EE                FLDZ

                    ;------------ FMUL ------------
DCCB                FMUL   ST3,ST
D8CB                FMUL   ST,ST3
DEC9                FMUL

                    ;------------ FMUL32 ----------
42D84C7B05          FMUL32   [RBX+R15*2]+5

                    ;------------ FMUL64 ----------
42DC4C7B05          FMUL64   [RBX+R15*2]+5

                    ;------------ FMULP -----------
DECB                FMULP   ST3,ST

                    ;------------ FNCLEX ----------
DBE2                FNCLEX

                    ;------------ FNDISI ----------
DBE1                FNDISI

                    ;------------ FNENI -----------
DBE0                FNENI

                    ;------------ FNINIT ----------
```

```
DBE3                FNINIT

                    ;------------- FNOP ------------
D9D0                FNOP

                    ;------------ FNSAVE -----------
42DD747B05          FNSAVE   [RBX+R15*2]+5

                    ;------------ FNSTCW -----------
2643D93C63          FNSTCW   ES:W PTR [R11+R12*2]

                    ;----------- FNSTENV -----------
42D9747B05          FNSTENV   [RBX+R15*2]+5

                    ;------------ FNSTSW -----------
DFE0                FNSTSW   AX
2643DD3C63          FNSTSW   ES:W PTR [R11+R12*2]

                    ;------------ FPATAN -----------
D9F3                FPATAN

                    ;------------- FPOP ------------
DDD8                FPOP

                    ;------------ FPREM ------------
D9F8                FPREM

                    ;------------ FPREM1 -----------
D9F5                FPREM1

                    ;------------ FPTAN ------------
D9F2                FPTAN

                    ;----------- FRNDINT -----------
D9FC                FRNDINT

                    ;------------ FRSTOR -----------
42DD647B05          FRSTOR   [RBX+R15*2]+5

                    ;------------ FSAVE ------------
9B6642DD747B05      FSAVE   [RBX+R15*2]+5


                    ;------------ FSCALE -----------
D9FD                FSCALE

                    ;------------- FSIN ------------
D9FE                FSIN

                    ;----------- FSINCOS -----------
D9FB                FSINCOS

                    ;------------ FSQRT ------------
D9FA                FSQRT

                    ;------------- FST -------------
DDD3                FST   ST3

                    ;------------ FST32 ------------
42D9547B05          FST32   [RBX+R15*2]+5

                    ;------------ FST32P -----------
42D95C7B05          FST32P   [RBX+R15*2]+5

                    ;------------ FST64 ------------
42DD547B05          FST64   [RBX+R15*2]+5

                    ;------------ FST64P -----------
42DD5C7B05          FST64P   [RBX+R15*2]+5

                    ;------------ FST80P -----------
42DB7C7B05          FST80P   [RBX+R15*2]+5

                    ;------------ FSTCW ------------
```

```
9B662643D93C        FSTCW   ES:W PTR [R11+R12*2]


                    ;------------ FSTENV -----------
9B6642D9747B05      FSTENV   [RBX+R15*2]+5

                    ;------------ FSTP ------------
DDDB                FSTP   ST3

                    ;------------ FSTSW -----------
9BDFE0              FSTSW   AX
9B662643DD3C63      FSTSW   ES:W PTR [R11+R12*2]

                    ;------------ FSUB ------------
DCEB                FSUB   ST3,ST
D8E3                FSUB   ST,ST3
DEE9                FSUB

                    ;------------ FSUB32 ----------
42D8647B05          FSUB32   [RBX+R15*2]+5

                    ;------------ FSUB64 ----------
42DC647B05          FSUB64   [RBX+R15*2]+5

                    ;------------ FSUBP -----------
DEEB                FSUBP   ST3,ST

                    ;------------ FSUBR -----------
DCE3                FSUBR   ST3,ST
D8EB                FSUBR   ST,ST3
DEE1                FSUBR

                    ;----------- FSUBR32 ----------
42D86C7B05          FSUBR32   [RBX+R15*2]+5

                    ;----------- FSUBR64 ----------
42DC6C7B05          FSUBR64   [RBX+R15*2]+5

                    ;------------ FSUBRP ----------
DEE3                FSUBRP   ST3,ST

                    ;------------ FTST ------------
D9E4                FTST

                    ;------------ FUCOM -----------
DDE3                FUCOM   ST3
DDE1                FUCOM

                    ;------------ FUCOMI ----------
DBEB                FUCOMI   ST,ST3

                    ;----------- FUCOMIP ----------
DFEB                FUCOMIP   ST,ST3

                    ;------------ FUCOMP ----------
DDEB                FUCOMP   ST3
DDE9                FUCOMP

                    ;----------- FUCOMPP ----------
DAE9                FUCOMPP

                    ;------------ FWAIT -----------
9B                  FWAIT

                    ;------------ FXAM ------------
D9E5                FXAM

                    ;------------ FXCH ------------
D9CB                FXCH   ST3
D9C9                FXCH

                    ;----------- FXTRACT ----------
D9F4                FXTRACT
```

```
                    ;------------ FYL2X ------------
D9F1                FYL2X

                    ;----------- FYL2XP1 -----------
D9F9                FYL2XP1

                    ;------------- HLT -------------
F4                  HLT

                    ;------------- IDIV ------------
6641F738            IDIV   W PTR [R8]
41F73C24            IDIV   D PTR [R12]
4AF73CE3            IDIV   Q PTR [RBX+R12*8]
F63B                IDIV   B PTR [RBX]

                    ;------------ IINSB ------------
6C                  IINSB

                    ;------------ IINSD ------------
6D                  IINSD

                    ;------------ IINSW ------------
666D                IINSW

                    ;------------- IMUL ------------
6641F728            IMUL   W PTR [R8]
41F72C24            IMUL   D PTR [R12]
4AF72CE3            IMUL   Q PTR [RBX+R12*8]
F62B                IMUL   B PTR [RBX]
6669C93412          IMUL   CX,1234H
66410FAF08          IMUL   CX,W PTR [R8]
410FAF0C24          IMUL   ECX,D PTR [R12]
4E0FAF2CE3          IMUL   R13,Q PTR [RBX+R12*8]
6BC912              IMUL   ECX,12H
4D6BED12            IMUL   R13,12H
69C978563412        IMUL   ECX,12345678H
4D69ED785634        IMUL   R13,12345678H
666BC912            IMUL   CX,12H
6669C93412          IMUL   CX,CX,1234H
664169083412        IMUL   CX,W PTR [R8],1234H
666BC9FF            IMUL   CX,CX,0FFFFH
66416B08FF          IMUL   CX,W PTR [R8],0FFFFH
666BC912            IMUL   CX,CX,12H
66416B0812          IMUL   CX,W PTR [R8],12H
6BC9FF              IMUL   ECX,ECX,0FFFFFFFFH
4D6BEDFF            IMUL   R13,R13,0FFFFFFFFH
416B0C24FF          IMUL   ECX,D PTR [R12],0FFFFFFFFH
4E6B2CE3FF          IMUL   R13,Q PTR [RBX+R12*8],0FFFFFFFFH
69C978563412        IMUL   ECX,ECX,12345678H
4D69ED785634        IMUL   R13,R13,12345678H
41690C2478563412    IMUL   ECX,D PTR [R12],12345678H
4E692CE378563412    IMUL   R13,Q PTR [RBX+R12*8],12345678H

                    ;------------- IN --------------
66ED                IN   AX,DX
ED                  IN   EAX,DX
EC                  IN   AL,DX
66E512              IN   AX,12H
E512                IN   EAX,12H
48E512              IN   RAX,12H
EC                  IN   AL
66ED                IN   AX
ED                  IN   EAX
48ED                IN   RAX
E412                IN   AL,12H

                    ;------------- INC -------------
66FFC1              INC   CX
FFC1                INC   ECX
6641FF00            INC   W PTR [R8]
41FF0424            INC   D PTR [R12]
4AFF04E3            INC   Q PTR [RBX+R12*8]
FE03                INC   B PTR [RBX]
```

```
                    ;------------- INSB ------------
676C                INSB

                    ;------------- INSD ------------
676D                INSD

                    ;------------- INSW ------------
66676D              INSW

                    ;------------- INT -------------
CC                  INT   3
CD12                INT   12H

                    ;------------- INTO ------------

                    ;------------- INVD ------------
0F08                INVD

                    ;------------ INVLPG -----------
660F017C7B05        INVLPG   [RBX+R15*2]+5

                    ;------------- IRET ------------
CF                  IRET

                    ;------------ IRETD ------------
CF                  IRETD

                    ;-------------- JA -------------
                    M2:
77FE                JA   M2

                    ;------------- JAE -------------
73FC                JAE   M2

                    ;-------------- JB -------------
72FA                JB   M2

                    ;------------- JBE -------------
76F8                JBE   M2

                    ;------------- JCXZ ------------
67E3F5              JCXZ   M2

                    ;-------------- JE -------------
74F3                JE   M2

                    ;------------ JECXZ ------------
E3F1                JECXZ   M2

                    ;-------------- JG -------------
7FEF                JG   M2

                    ;------------- JGE -------------
7DED                JGE   M2

                    ;------------- JJA -------------
0F87B5F8FFFF        JJA   M1

                    ;------------- JJAE ------------
0F83AFF8FFFF        JJAE   M1

                    ;------------- JJB -------------
0F82A9F8FFFF        JJB   M1

                    ;------------- JJBE ------------
0F86A3F8FFFF        JJBE   M1

                    ;------------- JJE -------------
0F849DF8FFFF        JJE   M1

                    ;------------- JJG -------------
0F8F97F8FFFF        JJG   M1

                    ;------------- JJGE ------------
```

```
0F8D91F8FFFF        JJGE   M1

                    ;------------- JJL -------------
0F8C8BF8FFFF        JJL   M1

                    ;------------- JJLE ------------
0F8E85F8FFFF        JJLE   M1

                    ;------------- JJNA ------------
0F867FF8FFFF        JJNA   M1

                    ;------------- JJNB ------------
0F8379F8FFFF        JJNB   M1

                    ;------------- JJNC ------------
0F8373F8FFFF        JJNC   M1

                    ;------------- JJNE ------------
0F856DF8FFFF        JJNE   M1

                    ;------------- JJNG ------------
0F8E67F8FFFF        JJNG   M1

                    ;------------- JJNL ------------
0F8D61F8FFFF        JJNL   M1

                    ;------------- JJNO ------------
0F815BF8FFFF        JJNO   M1

                    ;------------- JJNP ------------
0F8B55F8FFFF        JJNP   M1

                    ;------------- JJNS ------------
0F894FF8FFFF        JJNS   M1

                    ;------------- JJNZ ------------
0F8549F8FFFF        JJNZ   M1

                    ;------------- JJO -------------
0F8043F8FFFF        JJO   M1

                    ;------------- JJP -------------
0F8A3DF8FFFF        JJP   M1

                    ;------------- JJPO ------------
0F8B37F8FFFF        JJPO   M1

                    ;------------- JJS -------------
0F8831F8FFFF        JJS   M1

                    ;-------------- JL -------------
                    M3:
7CFE                JL   M3

                    ;------------- JLE -------------
7EFC                JLE   M3

                    ;------------- JMP -------------
E928F8FFFF          JMP   M1
FFE1                JMP   ECX
41FF2424            JMP   D PTR [R12]
4AFF24E3            JMP   Q PTR [RBX+R12*8]

                    ;------------- JMPS ------------
                    M4:
EBFE                JMPS   M4

                    ;------------- JNE -------------
75FC                JNE   M4

                    ;------------- JNO -------------
71FA                JNO   M4

                    ;------------- JNP -------------
```

```
7BF8                JNP   M4

                    ;------------- JNS -------------
79F6                JNS   M4

                    ;-------------- JO -------------
70F4                JO   M4

                    ;-------------- JP -------------
7AF2                JP   M4

                    ;-------------- JS -------------
78F0                JS   M4

                    ;------------- LAHF ------------
9F                  LAHF

                    ;------------- LAR -------------
660F0208            LAR   CX,W PTR [R8]

                    ;------------- LEA -------------
66428D4C7B05        LEA   CX,[RBX+R15*2]+5
428D4C7B05          LEA   ECX,[RBX+R15*2]+5
4E8D6C7B05          LEA   R13,[RBX+R15*2]+5

                    ;------------ LEAVE ------------
C9                  LEAVE

                    ;------------- LES -------------

                    ;------------- LFS -------------
660FB40B            LFS   CX,D PTR [R11]
0FB44801            LFS   ECX,[R8]+1

                    ;------------- LGDT ------------
0F01547B05          LGDT   [RBX+R15*2]+5

                    ;------------- LGS -------------
660FB50B            LGS   CX,D PTR [R11]
0FB54801            LGS   ECX,[R8]+1

                    ;------------- LIDT ------------
0F015C7B05          LIDT   [RBX+R15*2]+5

                    ;------------- LLDT ------------
0F0010              LLDT   W PTR [R8]

                    ;------------ LLODSB -----------
AC                  LLODSB

                    ;------------ LLODSD -----------
AD                  LLODSD

                    ;------------ LLODSQ -----------
48AD                LLODSQ

                    ;------------ LLODSW -----------
66AD                LLODSW

                    ;------------ LLOOP ------------
                    M5:
E2FE                LLOOP   M5

                    ;------------ LLOOPE -----------
E1FC                LLOOPE   M5

                    ;----------- LLOOPNE -----------
E0FA                LLOOPNE   M5

                    ;----------- LLOOPNZ -----------
E0F8                LLOOPNZ   M5

                    ;------------ LLOOPZ -----------
E1F6                LLOOPZ   M5
```

```
                    ;------------- LMSW ------------
660F0130            LMSW   W PTR [R8]

                    ;------------- LOCK ------------
F0A4                LOCK   MMOVSB

                    ;------------- LODS ------------
6626AD              LODS   ES:W PTR [R11+R12*2]
AD                  LODS   D PTR [RBX+RBP*2]
26AC                LODS   ES:B PTR [RBX+RSI*2]+1

                    ;------------ LODSB ------------
67AC                LODSB

                    ;------------ LODSD ------------
67AD                LODSD

                    ;------------ LODSQ ------------
6748AD              LODSQ

                    ;------------ LODSW ------------
6667AD              LODSW

                    ;------------- LOOP ------------
67E2DD              LOOP   M5

                    ;------------ LOOPE ------------
67E1DA              LOOPE   M5

                    ;------------ LOOPNE -----------
67E0D7              LOOPNE   M5

                    ;------------ LOOPNZ -----------
67E0D4              LOOPNZ   M5

                    ;------------ LOOPZ ------------
67E1D1              LOOPZ   M5

                    ;------------- LSL -------------
660F0308            LSL   CX,W PTR [R8]
0F030C24            LSL   ECX,D PTR [R12]

                    ;------------- LSS -------------
660FB20B            LSS   CX,D PTR [R11]
0FB24801            LSS   ECX,[R8]+1

                    ;------------- LTR -------------
0F0018              LTR   W PTR [R8]

                    ;------------ MMOVSB -----------
A4                  MMOVSB

                    ;------------ MMOVSD -----------
A5                  MMOVSD

                    ;------------ MMOVSQ -----------
48A5                MMOVSQ

                    ;------------ MMOVSW -----------
66A5                MMOVSW

                    ;------------- MOV -------------
66A300000000000000000 MOV   W PTR X,AX
A30000000000000000   MOV   D PTR X,EAX
48A300000000000000000 MOV   Q PTR X,RAX
A20000000000000000   MOV   B PTR X,AL
66A100000000000000000 MOV   AX,W PTR X
A10000000000000000   MOV   EAX,D PTR X
48A100000000000000000 MOV   RAX,Q PTR X
A00000000000000000   MOV   AL,B PTR X
0FA00FA1             MOV   FS,FS
68341200000FA1       MOV   FS,1234H
6A120FA1             MOV   FS,12H
```

```
8EE1                MOV   FS,CX
8E20                MOV   FS,W PTR [R8]
8CE1                MOV   CX,FS
418C20              MOV   W PTR [R8],FS
8CD1                MOV   CX,SS
418C10              MOV   W PTR [R8],SS
66418B08            MOV   CX,W PTR [R8]
418B0C24            MOV   ECX,D PTR [R12]
4E8B2CE3            MOV   R13,Q PTR [RBX+R12*8]
448A1B              MOV   R11L,B PTR [RBX]
66418908            MOV   W PTR [R8],CX
41890C24            MOV   D PTR [R12],ECX
4E892CE3            MOV   Q PTR [RBX+R12*8],R13
44881B              MOV   B PTR [RBX],R11L
66B93412            MOV   CX,1234H
B978563412          MOV   ECX,12345678H
4DBD7856341200000000 MOV  R13,12345678H
66B91200            MOV   CX,12H
45B312              MOV   R11L,12H
6641C7003412        MOV   W PTR [R8],1234H
41C7042478563412    MOV   D PTR [R12],12345678H
4AC704E378563412    MOV   Q PTR [RBX+R12*8],12345678H
6641C7001200        MOV   W PTR [R8],12H
C60312              MOV   B PTR [RBX],12H
0F23D1              MOV   DR2,ECX
0F21D1              MOV   ECX,DR2
0F26D1              MOV   TR2,ECX
0F24D1              MOV   ECX,TR2
0F22D1              MOV   CR2,ECX
0F20D1              MOV   ECX,CR2

                    ;------------- MOVS ------------
A5                  MOVS   D PTR [RSI],ES:D PTR [RDI]
6626A5              MOVS   W PTR [RSI],ES:W PTR [RDI]
26A4                MOVS   B PTR [RSI],ES:B PTR [RDI]

                    ;------------ MOVSB ------------
67A4                MOVSB

                    ;------------ MOVSD ------------
67A5                MOVSD
F2460F10647B05      MOVSD   XMM12,[RBX+R15*2]+5
F2460F11647B05      MOVSD   [RBX+R15*2]+5,XMM12
F2450F10E4          MOVSD   XMM12,XMM12

                    ;------------ MOVSW ------------
6667A5              MOVSW

                    ;------------ MOVSQ ------------
6748A5              MOVSQ

                    ;------------ MOVSX ------------
660FBE0B            MOVSX   CX,B PTR [RBX]
0FBE0B              MOVSX   ECX,B PTR [RBX]
4C0FBE2B            MOVSX   R13,B PTR [RBX]
410FBF08            MOVSX   ECX,W PTR [R8]
4D0FBF28            MOVSX   R13,W PTR [R8]
4D632C24            MOVSX   R13,D PTR [R12]

                    ;------------ MOVZX ------------
660FB60B            MOVZX   CX,B PTR [RBX]
0FB60B              MOVZX   ECX,B PTR [RBX]
4C0FB62B            MOVZX   R13,B PTR [RBX]
410FB708            MOVZX   ECX,W PTR [R8]
4D0FB728            MOVZX   R13,W PTR [R8]

                    ;------------- MUL -------------
6641F720            MUL   W PTR [R8]
41F72424            MUL   D PTR [R12]
4AF724E3            MUL   Q PTR [RBX+R12*8]
F623                MUL   B PTR [RBX]

                    ;------------- NEG -------------
6641F718            NEG   W PTR [R8]
```

```
41F71C24            NEG   D PTR [R12]
4AF71CE3            NEG   Q PTR [RBX+R12*8]
F61B                NEG   B PTR [RBX]

                    ;------------- NOP -------------
90                  NOP

                    ;------------- NOT -------------
6641F710            NOT   W PTR [R8]
41F71424            NOT   D PTR [R12]
4AF714E3            NOT   Q PTR [RBX+R12*8]
F613                NOT   B PTR [RBX]

                    ;------------ OOUTSB -----------
6E                  OOUTSB

                    ;------------ OOUTSD -----------
6F                  OOUTSD

                    ;------------ OOUTSW -----------
666F                OOUTSW

                    ;-------------- OR -------------
66410B08            OR   CX,W PTR [R8]
410B0C24            OR   ECX,D PTR [R12]
4E0B2CE3            OR   R13,Q PTR [RBX+R12*8]
440A1B              OR   R11L,B PTR [RBX]
66410908            OR   W PTR [R8],CX
41090C24            OR   D PTR [R12],ECX
4E092CE3            OR   Q PTR [RBX+R12*8 ,R13
44081B              OR   B PTR [RBX],R11L
66418308FF          OR   W PTR [R8],0FFFFH
41830C24FF          OR   D PTR [R12],0FFFFFFFFH
4A830CE3FF          OR   Q PTR [RBX+R12*8],0FFFFFFFFH
660D3412            OR   AX,1234H
6683C812            OR   AX,12H
83C812              OR   EAX,12H
0D78563412          OR   EAX,12345678H
480D78563412        OR   RAX,12345678H
0C12                OR   AL,12H
6641830812          OR   W PTR [R8],12H
41830C2412          OR   D PTR [R12],12H
4A830CE312          OR   Q PTR [RBX+R12*8],12H
664181083412        OR   W PTR[R8],1234H
41810C2478563412    OR   D PTR [R12],12345678H
4A810CE378563412    OR   Q PTR [RBX+R12*8],12345678H
800B12              OR   B PTR [RBX],12H

                    ;------------- OUT -------------
66EF                OUT   DX,AX
EF                  OUT   DX,EAX
EE                  OUT   DX,AL
66E712              OUT   12H,AX
E712                OUT   12H,EAX
48E712              OUT   12H,RAX
EE                  OUT   AL
66EF                OUT   AX
EF                  OUT   EAX
E612                OUT   12H,AL

                    ;------------ OUTSB ------------
676E                OUTSB

                    ;------------ OUTSD ------------
676F                OUTSD

                    ;------------ OUTSW ------------
66676F              OUTSW

                    ;------------- POP -------------
660FA1              POP   FS
66418F00            POP   W PTR [R8]
4A8F04E3            POP   Q PTR [RBX+R12*8]
```

```
                    ;------------- POPF ------------
669D                POPF

                    ;------------ POPFD ------------
9D                  POPFD

                    ;------------- PPOP ------------
0FA1                PPOP   FS

                    ;------------ PPUSH ------------
0FA0                PPUSH   FS
6AFF                PPUSH   0FFFFFFFFH
6AFF                PPUSH   0FFFFH
6A12                PPUSH   12H
6834120000          PPUSH   1234H

                    ;------------- PUSH ------------
660FA0              PUSH   FS
6641FF30            PUSH   W PTR [R8]
4AFF34E3            PUSH   Q PTR [RBX+R12*8]
6AFF                PUSH   0FFFFFFFFH
666AFF              PUSH   0FFFFH
666A12              PUSH   12H
66683412            PUSH   1234H
6878563412          PUSH   12345678H

                    ;------------ PUSHF ------------
669C                PUSHF

                    ;------------ PUSHFD -----------
9C                  PUSHFD

                    ;------------- RCL -------------
6641D310            RCL   W PTR [R8],CL
41D31424            RCL   D PTR [R12],CL
4AD314E3            RCL   Q PTR [RBX+R12*8],CL
D213                RCL   B PTR [RBX],CL
6641D110            RCL   W PTR [R8],1
41D11424            RCL   D PTR [R12],1
4AD114E3            RCL   Q PTR [RBX+R12*8],1
D013                RCL   B PTR [RBX],1
6641C11005          RCL   W PTR [R8],5
41C114241F          RCL   D PTR [R12],1FH
4AC114E340          RCL   Q PTR [RBX+R12*8],40H
D013                RCL   B PTR [RBX]
6641D110            RCL   W PTR [R8]
41D11424            RCL   D PTR [R12]
4AD114E3            RCL   Q PTR [RBX+R12*8]
C01308              RCL   B PTR [RBX],8

                    ;------------- RCR -------------
6641D318            RCR   W PTR [R8],CL
41D31C24            RCR   D PTR [R12],CL
4AD31CE3            RCR   Q PTR [RBX+R12*8],CL
D21B                RCR   B PTR [RBX],CL
6641D118            RCR   W PTR [R8],1
41D11C24            RCR   D PTR [R12],1
4AD11CE3            RCR   Q PTR [RBX+R12*8],1
D01B                RCR   B PTR [RBX],1
6641C11805          RCR   W PTR [R8],5
41C11C241F          RCR   D PTR [R12],1FH
4AC11CE340          RCR   Q PTR [RBX+R12*8],40H
D01B                RCR   B PTR [RBX]
6641D118            RCR   W PTR [R8]
41D11C24            RCR   D PTR [R12]
4AD11CE3            RCR   Q PTR [RBX+R12*8]
C01B08              RCR   B PTR [RBX],8

                    ;------------ RDMSR ------------
0F32                RDMSR

                    ;------------ RDPMC ------------
0F33                RDPMC
```

```
                ;------------ RDTSC ------------
0F31            RDTSC

                ;------------- REP -------------
F3A4            REP   MMOVSB

                ;------------- REPE ------------
F3A4            REPE   MMOVSB

                ;------------ REPNE ------------
F2A4            REPNE   MMOVSB

                ;------------ REPNZ ------------
F2A4            REPNZ   MMOVSB

                ;------------- RET -------------
C3              RET
C23412          RET   1234H
C21200          RET   12H

                ;------------- RETF ------------
CB              RETF
CA3412          RETF   1234H
CA1200          RETF   12H

                ;------------- ROL -------------
6641D300        ROL   W PTR [R8],CL
41D30424        ROL   D PTR [R12],CL
4AD304E3        ROL   Q PTR [RBX+R12*8],CL
D203            ROL   B PTR [RBX],CL
6641D100        ROL   W PTR [R8],1
41D10424        ROL   D PTR [R12],1
4AD104E3        ROL   Q PTR [RBX+R12*8],1
D003            ROL   B PTR [RBX],1
6641C10002      ROL   W PTR [R8],2
41C1042410      ROL   D PTR [R12],10H
4AC104E33F      ROL   Q PTR [RBX+R12*8],3FH
D003            ROL   B PTR [RBX]
6641D100        ROL   W PTR [R8]
41D10424        ROL   D PTR [R12]
4AD104E3        ROL   Q PTR [RBX+R12*8]
C00306          ROL   B PTR [RBX],6

                ;------------- ROR -------------
6641D308        ROR   W PTR [R8],CL
41D30C24        ROR   D PTR [R12],CL
4AD30CE3        ROR   Q PTR [RBX+R12*8],CL
D20B            ROR   B PTR [RBX],CL
6641D108        ROR   W PTR [R8],1
41D10C24        ROR   D PTR [R12],1
4AD10CE3        ROR   Q PTR [RBX+R12*8],1
D00B            ROR   B PTR [RBX],1
6641C10802      ROR   W PTR [R8],2
41C10C2410      ROR   D PTR [R12],10H
4AC10CE33F      ROR   Q PTR [RBX+R12*8],3FH
D00B            ROR   B PTR [RBX]
6641D108        ROR   W PTR [R8]
41D10C24        ROR   D PTR [R12]
4AD10CE3        ROR   Q PTR [RBX+R12*8]
C00B06          ROR   B PTR [RBX],6

                ;------------- RSM -------------
0FAA            RSM

                ;------------- SAHF ------------
9E              SAHF

                ;------------- SAL -------------
6641D320        SAL   W PTR [R8],CL
41D32424        SAL   D PTR [R12],CL
4AD324E3        SAL   Q PTR [RBX+R12*8],CL
D223            SAL   B PTR [RBX],CL
6641D120        SAL   W PTR [R8],1
41D12424        SAL   D PTR [R12],1
```

```
4AD124E3            SAL   Q PTR [RBX+R12*8],1
D023                SAL   B PTR [RBX],1
6641C12002          SAL   W PTR [R8],2
41C1242410          SAL   D PTR [R12],10H
4AC124E33F          SAL   Q PTR [RBX+R12*8],3FH
D023                SAL   B PTR [RBX]
6641D120            SAL   W PTR [R8]
41D12424            SAL   D PTR [R12]
4AD124E3            SAL   Q PTR [RBX+R12*8]
C02306              SAL   B PTR [RBX],6

                    ;------------- SAR -------------
6641D338            SAR   W PTR [R8],CL
41D33C24            SAR   D PTR [R12],CL
4AD33CE3            SAR   Q PTR [RBX+R12*8],CL
D23B                SAR   B PTR [RBX],CL
6641D138            SAR   W PTR [R8],1
41D13C24            SAR   D PTR [R12],1
4AD13CE3            SAR   Q PTR [RBX+R12*8],1
D03B                SAR   B PTR [RBX],1
6641C13802          SAR   W PTR [R8],2
41C13C2410          SAR   D PTR [R12],10H
4AC13CE33F          SAR   Q PTR [RBX+R12*8],3FH
D03B                SAR   B PTR [RBX]
6641D138            SAR   W PTR [R8]
41D13C24            SAR   D PTR [R12]
4AD13CE3            SAR   Q PTR [RBX+R12*8]
C03B06              SAR   B PTR [RBX],6

                    ;------------- SBB -------------
66411B08            SBB   CX,W PTR [R8]
411B0C24            SBB   ECX,D PTR [R12]
4E1B2CE3            SBB   R13,Q PTR [RBX+R12*8]
441A1B              SBB   R11L,B PTR [RBX]
66411908            SBB   W PTR [R8],CX
41190C24            SBB   D PTR [R12],ECX
4E192CE3            SBB   Q PTR [RBX+R12*8],R13
44181B              SBB   B PTR [RBX],R11L
66418318FF          SBB   W PTR [R8],0FFFFH
41831C24FF          SBB   D PTR [R12],0FFFFFFFFH
4A831CE3FF          SBB   Q PTR [RBX+R12*8],0FFFFFFFFH
661D3412            SBB   AX,1234H
1D78563412          SBB   EAX,12345678H
481D78563412        SBB   RAX,12345678H
6683D812            SBB   AX,12H
83D812              SBB   EAX,12H
4883D812            SBB   RAX,12H
1C12                SBB   AL,12H
664181183412        SBB   W PTR [R8],1234H
6641831812          SBB   W PTR [R8],12H
41831C2412          SBB   D PTR [R12],12H
4A831CE312          SBB   Q PTR [RBX+R12*8],12H
41811C2478563412    SBB   D PTR [R12],12345678H
4A811CE378563412    SBB   Q PTR [RBX+R12*8],12345678H
801B12              SBB   B PTR [RBX],12H

                    ;------------ SCASB ------------
67AE                SCASB

                    ;------------ SCASD ------------
67AF                SCASD

                    ;------------ SCASW ------------
6667AF              SCASW

                    ;------------- SETA ------------
0F9703              SETA   B PTR [RBX]

                    ;------------ SETAE ------------
0F9303              SETAE   B PTR [RBX]

                    ;------------- SETB ------------
0F9203              SETB   B PTR [RBX]
```

```
                ;------------ SETBE ------------
0F9603          SETBE   B PTR [RBX]

                ;------------ SETC ------------
0F9203          SETC   B PTR [RBX]

                ;------------ SETE ------------
0F9403          SETE   B PTR [RBX]

                ;------------ SETG ------------
0F9F03          SETG   B PTR [RBX]

                ;------------ SETGE ------------
0F9D03          SETGE   B PTR [RBX]

                ;------------ SETL ------------
0F9C03          SETL   B PTR [RBX]

                ;------------ SETLE ------------
0F9E03          SETLE   B PTR [RBX]

                ;------------ SETNA ------------
0F9603          SETNA   B PTR [RBX]

                ;------------ SETNC ------------
0F9303          SETNC   B PTR [RBX]

                ;------------ SETNE ------------
0F9503          SETNE   B PTR [RBX]

                ;------------ SETNG ------------
0F9E03          SETNG   B PTR [RBX]

                ;------------ SETNGE ------------
0F9C03          SETNGE   B PTR [RBX]

                ;------------ SETNL ------------
0F9D03          SETNL   B PTR [RBX]

                ;------------ SETNLE ------------
0F9F03          SETNLE   B PTR [RBX]

                ;------------ SETNO ------------
0F9103          SETNO   B PTR [RBX]

                ;------------ SETNP ------------
0F9B03          SETNP   B PTR [RBX]

                ;------------ SETNS ------------
0F9903          SETNS   B PTR [RBX]

                ;------------ SETNZ ------------
0F9503          SETNZ   B PTR [RBX]

                ;------------ SETO ------------
0F9003          SETO   B PTR [RBX]

                ;------------ SETP ------------
0F9A03          SETP   B PTR [RBX]

                ;------------ SETPE ------------
0F9A03          SETPE   B PTR [RBX]

                ;------------ SETPO ------------
0F9B03          SETPO   B PTR [RBX]

                ;------------ SETS ------------
0F9803          SETS   B PTR [RBX]

                ;------------ SETZ ------------
0F9403          SETZ   B PTR [RBX]

                ;------------ SGDT ------------
420F01447B05    SGDT   [RBX+R15*2]+5
```

```
                    ;------------- SHLD ------------
66410FA40812        SHLD   W PTR [R8],CX,12H
66410FA508          SHLD   W PTR [R8],CX,CL
410FA40C2412        SHLD   D PTR [R12],ECX,12H
4E0FA42CE312        SHLD   Q PTR [RBX+R12*8],R13,12H
410FA50C24          SHLD   D PTR [R12],ECX,CL
4E0FA52CE3          SHLD   Q PTR [RBX+R12*8],R13,CL

                    ;------------- SHR -------------
6641D328            SHR   W PTR [R8],CL
41D32C24            SHR   D PTR [R12],CL
4AD32CE3            SHR   Q PTR [RBX+R12*8],CL
D22B                SHR   B PTR [RBX],CL
6641D128            SHR   W PTR [R8],1
41D12C24            SHR   D PTR [R12],1
4AD12CE3            SHR   Q PTR [RBX+R12*8],1
D02B                SHR   B PTR [RBX],1
C02B02              SHR   B PTR[RBX],2
D02B                SHR   B PTR [RBX]
6641D128            SHR   W PTR [R8]
41D12C24            SHR   D PTR [R12]
4AD12CE3            SHR   Q PTR [RBX+R12*8]
6641C12802          SHR   W PTR [R8],2
41C12C2410          SHR   D PTR [R12],10H
4AC12CE33F          SHR   Q PTR [RBX+R12*8],3FH

                    ;------------- SHRD ------------
66410FAC0812        SHRD   W PTR [R8],CX,12H
66410FAD08          SHRD   W PTR [R8],CX,CL
410FAC0C2412        SHRD   D PTR [R12],ECX,12H
4E0FAC2CE312        SHRD   Q PTR [RBX+R12*8],R13,12H
410FAD0C24          SHRD   D PTR [R12],ECX,CL
4E0FAD2CE3          SHRD   Q PTR [RBX+R12*8],R13,CL

                    ;------------- SIDT ------------
0F014C7B05          SIDT   [RBX+R15*2]+5

                    ;------------- SLDT ------------
660F0000            SLDT   W PTR [R8]

                    ;------------- SMSW ------------
660F0120            SMSW   W PTR [R8]

                    ;------------ SSCASB -----------
AE                  SSCASB

                    ;------------ SSCASD -----------
AF                  SSCASD

                    ;------------ SSCASW -----------
66AF                SSCASW

                    ;------------ SSTOSB -----------
AA                  SSTOSB

                    ;------------ SSTOSD -----------
AB                  SSTOSD

                    ;------------ SSTOSQ -----------
48AB                SSTOSQ

                    ;------------ SSTOSW -----------
66AB                SSTOSW

                    ;------------- STC -------------
F9                  STC

                    ;------------- STD -------------
FD                  STD

                    ;------------- STI -------------
FB                  STI
```

```
                    ;------------ STOSB ------------
67AA                STOSB

                    ;------------ STOSD ------------
67AB                STOSD

                    ;------------ STOSQ ------------
6748AB              STOSQ

                    ;------------ STOSW ------------
6667AB              STOSW

                    ;------------- STR -------------
660F0008            STR   W PTR [R8]

                    ;------------- SUB -------------
66412B08            SUB   CX,W PTR [R8]
412B0C24            SUB   ECX,D PTR [R12]
4E2B2CE3            SUB   R13,Q PTR [RBX+R12*8]
442A1B              SUB   R11L,B PTR [RBX]
66412908            SUB   W PTR [R8],CX
41290C24            SUB   D PTR [R12],ECX
4E292CE3            SUB   Q PTR [RBX+R12*8],R13
44281B              SUB   B PTR [RBX],R11L
66418328FF          SUB   W PTR [R8],0FFFFH
41832C24FF          SUB   D PTR [R12],0FFFFFFFFH
4A832CE3FF          SUB   Q PTR [RBX+R12*8],0FFFFFFFFH
662D3412            SUB   AX,1234H
2D78563412          SUB   EAX,12345678H
482D78563412        SUB   RAX,12345678H
6683E812            SUB   AX,12H
83E812              SUB   EAX,12H
2C12                SUB   AL,12H
664181283412        SUB   W PTR [R8],1234H
41812C2478563412    SUB   D PTR [R12],12345678H
4A812CE378563412    SUB   Q PTR [RBX+R12*8],12345678H
6641832812          SUB   W PTR [R8],12H
41832C2412          SUB   D PTR [R12],12H
4A832CE312          SUB   Q PTR [RBX+R12*8],12H
802B12              SUB   B PTR [RBX],12H

                    ;------------- TEST ------------
66418508            TEST   CX,W PTR [R8]
41850C24            TEST   ECX,D PTR [R12]
4E852CE3            TEST   R13,Q PTR [RBX+R12*8]
44841B              TEST   R11L,B PTR [RBX]
66418508            TEST   W PTR [R8],CX
41850C24            TEST   D PTR [R12],ECX
4E852CE3            TEST   Q PTR [RBX+R12*8],R13
44841B              TEST   B PTR [RBX],R11L
66A93412            TEST   AX,1234H
A978563412          TEST   EAX,12345678H
48A978563412        TEST   RAX,12345678H
66A91200            TEST   AX,12H
A912000000          TEST   EAX,12H
48A912000000        TEST   RAX,12H
A812                TEST   AL,12H
6641F7003412        TEST   W PTR [R8],1234H
41F7042478563412    TEST   D PTR [R12],12345678H
4AF704E378563412    TEST   Q PTR [RBX+R12*8],12345678H
6641F7001200        TEST   W PTR [R8],12H
41F7042412000000    TEST   D PTR [R12],12H
4AF704E312000000    TEST   Q PTR [RBX+R12*8],12H
F60312              TEST   B PTR [RBX],12H

                    ;------------- UD2 -------------
0F0B                UD2

                    ;------------- VERR ------------
660F0020            VERR   W PTR [R8]

                    ;------------- VERW ------------
660F0028            VERW   W PTR [R8]
```

```
                    ;------------- WAIT ------------
9B                  WAIT

                    ;------------ WBINVD -----------
0F09                WBINVD

                    ;------------ WRMSR ------------
0F30                WRMSR

                    ;------------- XADD ------------
440FC01B            XADD   B PTR [RBX],R11L
66410FC108          XADD   W PTR [R8],CX
410FC10C24          XADD   D PTR [R12],ECX
4E0FC12CE3          XADD   Q PTR [RBX+R12*8],R13

                    ;------------- XCHG ------------
6691                XCHG   AX,CX
91                  XCHG   EAX,ECX
4C95                XCHG   RAX,R13
6691                XCHG   CX,AX
91                  XCHG   ECX,EAX
4D95                XCHG   R13,RAX
66418708            XCHG   CX,W PTR [R8]
41870C24            XCHG   ECX,D PTR[R12]
4E872CE3            XCHG   R13,Q PTR [RBX+R12*8]
44861B              XCHG   R11L,B PTR [RBX]
66418708            XCHG   W PTR [R8],CX
41870C24            XCHG   D PTR [R12],ECX
4E872CE3            XCHG   Q PTR [RBX+R12*8],R13
44861B              XCHG   B PTR [RBX],R11L

                    ;------------- XLAT ------------
67D7                XLAT
41D7                XLAT   B PTR [R9]

                    ;------------ XXLAT ------------
D7                  XXLAT

                    ;------------- XOR -------------
66413308            XOR   CX,W PTR [R8]
41330C24            XOR   ECX,D PTR [R12]
4E332CE3            XOR   R13,Q PTR [RBX+R12*8]
44321B              XOR   R11L,B PTR [RBX]
66413108            XOR   W PTR [R8],CX
41310C24            XOR   D PTR [R12],ECX
4E312CE3            XOR   Q PTR [RBX+R12*8],R13
44301B              XOR   B PTR [RBX],R11L
66418330FF          XOR   W PTR [R8],0FFFFH
41833424FF          XOR   D PTR [R12],0FFFFFFFFH
4A8334E3FF          XOR   Q PTR [RBX+R12*8],0FFFFFFFFH
66353412            XOR   AX,1234H
6683F012            XOR   AX,12H
83F012              XOR   EAX,12H
4883F012            XOR   RAX,12H
3578563412          XOR   EAX,12345678H
483578563412        XOR   RAX,12345678H
3412                XOR   AL,12H
664181303412        XOR   W PTR [R8],1234H
6641833012          XOR   W PTR [R8],12H
4183342412          XOR   D PTR [R12],12H
4A8334E312          XOR   Q PTR [RBX+R12*8],12H
4181342478563412    XOR   D PTR [R12],12345678H
4A8134E378563412    XOR   Q PTR [RBX+R12*8],12345678H
803312              XOR   B PTR [RBX],12H

                    ;------------- MOVD ------------
0F6ED1              MOVD   MM2,ECX
66440F6EE1          MOVD   XMM12,ECX
410F6E13            MOVD   MM2,D PTR [R11]
6666450F6E23        MOVD   XMM12,D PTR [R11]
0F7ED1              MOVD   ECX,MM2
66440F7EE1          MOVD   ECX,XMM12
410F7E13            MOVD   D PTR [R11],MM2
6666450F7E23        MOVD   D PTR [R11],XMM12
```

```
                    ;------------- MOVQ ------------
420F6F547B05        MOVQ   MM2,[RBX+R15*2]+5
420F7F547B05        MOVQ   [RBX+R15*2]+5,MM2
0F6FD2              MOVQ   MM2,MM2
F3460F7E647B05      MOVQ   XMM12,[RBX+R15*2]+5
66460FD6647B05      MOVQ   [RBX+R15*2]+5,XMM12
F3450F7EE4          MOVQ   XMM12,XMM12

                    ;----------- PACKSSWB ----------
420F63547B05        PACKSSWB   MM2,[RBX+R15*2]+5
0F63D2              PACKSSWB   MM2,MM2
66460F63647B05      PACKSSWB   XMM12,[RBX+R15*2]+5
66450F63E4          PACKSSWB   XMM12,XMM12

                    ;----------- PACKSSDW ----------
420F6B547B05        PACKSSDW   MM2,[RBX+R15*2]+5
0F6BD2              PACKSSDW   MM2,MM2
66460F6B647B05      PACKSSDW   XMM12,[RBX+R15*2]+5
66450F6BE4          PACKSSDW   XMM12,XMM12

                    ;----------- PACKUSWB ----------
420F67547B05        PACKUSWB   MM2,[RBX+R15*2]+5
0F67D2              PACKUSWB   MM2,MM2
66460F67647B05      PACKUSWB   XMM12,[RBX+R15*2]+5
66450F67E4          PACKUSWB   XMM12,XMM12

                    ;------------ PADDB ------------
420FFC547B05        PADDB   MM2,[RBX+R15*2]+5
0FFCD2              PADDB   MM2,MM2
66460FFC647B05      PADDB   XMM12,[RBX+R15*2]+5
66450FFCE4          PADDB   XMM12,XMM12

                    ;------------ PADDW ------------
420FFD547B05        PADDW   MM2,[RBX+R15*2]+5
0FFDD2              PADDW   MM2,MM2
66460FFD647B05      PADDW   XMM12,[RBX+R15*2]+5
66450FFDE4          PADDW   XMM12,XMM12

                    ;------------ PADDD ------------
420FFE547B05        PADDD   MM2,[RBX+R15*2]+5
0FFED2              PADDD   MM2,MM2
66460FFE647B05      PADDD   XMM12,[RBX+R15*2]+5
66450FFEE4          PADDD   XMM12,XMM12

                    ;------------ PADDSB -----------
420FEC547B05        PADDSB   MM2,[RBX+R15*2]+5
0FECD2              PADDSB   MM2,MM2
66460FEC647B05      PADDSB   XMM12,[RBX+R15*2]+5
66450FECE4          PADDSB   XMM12,XMM12

                    ;------------ PADDSW -----------
420FED547B05        PADDSW   MM2,[RBX+R15*2]+5
0FEDD2              PADDSW   MM2,MM2
66460FED647B05      PADDSW   XMM12,[RBX+R15*2]+5
66450FEDE4          PADDSW   XMM12,XMM12

                    ;----------- PADDUSB -----------
420FDC547B05        PADDUSB   MM2,[RBX+R15*2]+5
0FDCD2              PADDUSB   MM2,MM2
66460FDC647B05      PADDUSB   XMM12,[RBX+R15*2]+5
66450FDCE4          PADDUSB   XMM12,XMM12

                    ;----------- PADDUSW -----------
420FDD547B05        PADDUSW   MM2,[RBX+R15*2]+5
0FDDD2              PADDUSW   MM2,MM2
66460FDD647B05      PADDUSW   XMM12,[RBX+R15*2]+5
66450FDDE4          PADDUSW   XMM12,XMM12

                    ;------------- PAND ------------
420FDB547B05        PAND   MM2,[RBX+R15*2]+5
0FDBD2              PAND   MM2,MM2
66460FDB647B05      PAND   XMM12,[RBX+R15*2]+5
66450FDBE4          PAND   XMM12,XMM12
```

```
                        ;------------ PANDN ------------
420FDF547B05            PANDN   MM2,[RBX+R15*2]+5
0FDFD2                  PANDN   MM2,MM2
66460FDF647B05          PANDN   XMM12,[RBX+R15*2]+5
66450FDFE4              PANDN   XMM12,XMM12

                        ;----------- PCMPEQB -----------
420F74547B05            PCMPEQB   MM2,[RBX+R15*2]+5
0F74D2                  PCMPEQB   MM2,MM2
66460F74647B05          PCMPEQB   XMM12,[RBX+R15*2]+5
66450F74E4              PCMPEQB   XMM12,XMM12

                        ;----------- PCMPEQW -----------
420F75547B05            PCMPEQW   MM2,[RBX+R15*2]+5
0F75D2                  PCMPEQW   MM2,MM2
66460F75647B05          PCMPEQW   XMM12,[RBX+R15*2]+5
66450F75E4              PCMPEQW   XMM12,XMM12

                        ;----------- PCMPEQD -----------
420F76547B05            PCMPEQD   MM2,[RBX+R15*2]+5
0F76D2                  PCMPEQD   MM2,MM2
66460F76647B05          PCMPEQD   XMM12,[RBX+R15*2]+5
66450F76E4              PCMPEQD   XMM12,XMM12

                        ;----------- PCMPGTB -----------
420F64547B05            PCMPGTB   MM2,[RBX+R15*2]+5
0F64D2                  PCMPGTB   MM2,MM2
66460F64647B05          PCMPGTB   XMM12,[RBX+R15*2]+5
66450F64E4              PCMPGTB   XMM12,XMM12

                        ;----------- PCMPGTW -----------
420F65547B05            PCMPGTW   MM2,[RBX+R15*2]+5
0F65D2                  PCMPGTW   MM2,MM2
66460F65647B05          PCMPGTW   XMM12,[RBX+R15*2]+5
66450F65E4              PCMPGTW   XMM12,XMM12

                        ;----------- PCMPGTD -----------
420F66547B05            PCMPGTD   MM2,[RBX+R15*2]+5
0F66D2                  PCMPGTD   MM2,MM2
66460F66647B05          PCMPGTD   XMM12,[RBX+R15*2]+5
66450F66E4              PCMPGTD   XMM12,XMM12

                        ;----------- PMADDWD -----------
420FF5547B05            PMADDWD   MM2,[RBX+R15*2]+5
0FF5D2                  PMADDWD   MM2,MM2
66460FF5647B05          PMADDWD   XMM12,[RBX+R15*2]+5
66450FF5E4              PMADDWD   XMM12,XMM12

                        ;------------ PMULHW -----------
420FE5547B05            PMULHW   MM2,[RBX+R15*2]+5
0FE5D2                  PMULHW   MM2,MM2
66460FE5647B            PMULHW   XMM12,[RBX+R15*2]+5
66450FE5E4              PMULHW   XMM12,XMM12

                        ;------------ PMULLW -----------
420FD5547B05            PMULLW   MM2,[RBX+R15*2]+5
0FD5D2                  PMULLW   MM2,MM2
66460FD5647B05          PMULLW   XMM12,[RBX+R15*2]+5
66450FD5E4              PMULLW   XMM12,XMM12

                        ;------------- POR -------------
420FEB547B05            POR   MM2,[RBX+R15*2]+5
0FEBD2                  POR   MM2,MM2
66460FEB647B05          POR   XMM12,[RBX+R15*2]+5
66450FEBE4              POR   XMM12,XMM12

                        ;------------ PSLLW ------------
420FF1547B05            PSLLW   MM2,[RBX+R15*2]+5
0FF1D2                  PSLLW   MM2,MM2
66460FF1647B05          PSLLW   XMM12,[RBX+R15*2]+5
66450FF1E4              PSLLW   XMM12,XMM12
0F71F212                PSLLW   MM2,12H
66440F71F412            PSLLW   XMM12,12H
```

```
                        ;------------ PSLLD ------------
420FF2547B05            PSLLD   MM2,[RBX+R15*2]+5
0FF2D2                  PSLLD   MM2,MM2
66460FF2647B05          PSLLD   XMM12,[RBX+R15*2]+5
66450FF2E4              PSLLD   XMM12,XMM12
0F72F212                PSLLD   MM2,12H
66440F72F412            PSLLD   XMM12,12H

                        ;------------ PSLLQ ------------
420FF3547B05            PSLLQ   MM2,[RBX+R15*2]+5
0FF3D2                  PSLLQ   MM2,MM2
66460FF3647B05          PSLLQ   XMM12,[RBX+R15*2]+5
66450FF3E4              PSLLQ   XMM12,XMM12
0F73F212                PSLLQ   MM2,12H
66440F73F412            PSLLQ   XMM12,12H

                        ;------------ PSRAW ------------
420FE1547B05            PSRAW   MM2,[RBX+R15*2]+5
0FE1D2                  PSRAW   MM2,MM2
66460FE1647B05          PSRAW   XMM12,[RBX+R15*2]+5
66450FE1E4              PSRAW   XMM12,XMM12
0F71E212                PSRAW   MM2,12H
66440F71E412            PSRAW   XMM12,12H

                        ;------------ PSRAD ------------
420FE2547B05            PSRAD   MM2,[RBX+R15*2]+5
0FE2D2                  PSRAD   MM2,MM2
66460FE2647B05          PSRAD   XMM12,[RBX+R15*2]+5
66450FE2E4              PSRAD   XMM12,XMM12
0F72E212                PSRAD   MM2,12H
66440F72E412            PSRAD   XMM12,12H

                        ;------------ PSRLW ------------
420FD1547B05            PSRLW   MM2,[RBX+R15*2]+5
0FD1D2                  PSRLW   MM2,MM2
66460FD1647B05          PSRLW   XMM12,[RBX+R15*2]+5
66450FD1E4              PSRLW   XMM12,XMM12
0F71D212                PSRLW   MM2,12H
66440F71D412            PSRLW   XMM12,12H

                        ;------------ PSRLD ------------
420FD2547B05            PSRLD   MM2,[RBX+R15*2]+5
0FD2D2                  PSRLD   MM2,MM2
66460FD2647B05          PSRLD   XMM12,[RBX+R15*2]+5
66450FD2E4              PSRLD   XMM12,XMM12
0F72D212                PSRLD   MM2,12H
66440F72D412            PSRLD   XMM12,12H

                        ;------------ PSRLQ ------------
420FD3547B05            PSRLQ   MM2,[RBX+R15*2]+5
0FD3D2                  PSRLQ   MM2,MM2
66460FD3647B05          PSRLQ   XMM12,[RBX+R15*2]+5
66450FD3E4              PSRLQ   XMM12,XMM12
0F73D212                PSRLQ   MM2,12H
66440F73D412            PSRLQ   XMM12,12H

                        ;------------ PSUBB ------------
420FF8547B05            PSUBB   MM2,[RBX+R15*2]+5
0FF8D2                  PSUBB   MM2,MM2
66460FF8647B05          PSUBB   XMM12,[RBX+R15*2]+5
66450FF8E4              PSUBB   XMM12,XMM12

                        ;------------ PSUBW ------------
420FF9547B05            PSUBW   MM2,[RBX+R15*2]+5
0FF9D2                  PSUBW   MM2,MM2
66460FF9647B05          PSUBW   XMM12,[RBX+R15*2]+5
66450FF9E4              PSUBW   XMM12,XMM12

                        ;------------ PSUBD ------------
420FFA547B05            PSUBD   MM2,[RBX+R15*2]+5
0FFAD2                  PSUBD   MM2,MM2
66460FFA647B05          PSUBD   XMM12,[RBX+R15*2]+5
66450FFAE4              PSUBD   XMM12,XMM12
```

```
                    ;------------ PSUBSB -----------
420FE8547B05        PSUBSB   MM2,[RBX+R15*2]+5
0FE8D2              PSUBSB   MM2,MM2
66460FE8647B05      PSUBSB   XMM12,[RBX+R15*2]+5
66450FE8E4          PSUBSB   XMM12,XMM12

                    ;------------ PSUBSW -----------
420FE9547B05        PSUBSW   MM2,[RBX+R15*2]+5
0FE9D2              PSUBSW   MM2,MM2
66460FE9647B05      PSUBSW   XMM12,[RBX+R15*2]+5
66450FE9E4          PSUBSW   XMM12,XMM12

                    ;----------- PSUBUSB -----------
420FD8547B05        PSUBUSB   MM2,[RBX+R15*2]+5
0FD8D2              PSUBUSB   MM2,MM2
66460FD8647B05      PSUBUSB   XMM12,[RBX+R15*2]+5
66450FD8E4          PSUBUSB   XMM12,XMM12

                    ;----------- PSUBUSW -----------
420FD9547B05        PSUBUSW   MM2,[RBX+R15*2]+5
0FD9D2              PSUBUSW   MM2,MM2
66460FD9647B05      PSUBUSW   XMM12,[RBX+R15*2]+5
66450FD9E4          PSUBUSW   XMM12,XMM12

                    ;---------- PUNPCKHBW ----------
420F68547B05        PUNPCKHBW   MM2,[RBX+R15*2]+5
0F68D2              PUNPCKHBW   MM2,MM2
66460F68647B05      PUNPCKHBW   XMM12,[RBX+R15*2]+5
66450F68E4          PUNPCKHBW   XMM12,XMM12

                    ;---------- PUNPCKHWD ----------
420F69547B05        PUNPCKHWD   MM2,[RBX+R15*2]+5
0F69D2              PUNPCKHWD   MM2,MM2
66460F69647B05      PUNPCKHWD   XMM12,[RBX+R15*2]+5
66450F69E4          PUNPCKHWD   XMM12,XMM12

                    ;---------- PUNPCKHDQ ----------
420F6A547B05        PUNPCKHDQ   MM2,[RBX+R15*2]+5
0F6AD2              PUNPCKHDQ   MM2,MM2
66460F6A647B05      PUNPCKHDQ   XMM12,[RBX+R15*2]+5
66450F6AE4          PUNPCKHDQ   XMM12,XMM12

                    ;---------- PUNPCKLBW ----------
420F60547B05        PUNPCKLBW   MM2,[RBX+R15*2]+5
0F60D2              PUNPCKLBW   MM2,MM2
66460F60647B05      PUNPCKLBW   XMM12,[RBX+R15*2]+5
66450F60E4          PUNPCKLBW   XMM12,XMM12

                    ;---------- PUNPCKLWD ----------
420F61547B05        PUNPCKLWD   MM2,[RBX+R15*2]+5
0F61D2              PUNPCKLWD   MM2,MM2
66460F61647B05      PUNPCKLWD   XMM12,[RBX+R15*2]+5
66450F61E4          PUNPCKLWD   XMM12,XMM12

                    ;---------- PUNPCKLDQ ----------
420F62547B05        PUNPCKLDQ   MM2,[RBX+R15*2]+5
0F62D2              PUNPCKLDQ   MM2,MM2
66460F62647B05      PUNPCKLDQ   XMM12,[RBX+R15*2]+5
66450F62E4          PUNPCKLDQ   XMM12,XMM12

                    ;------------- PXOR ------------
420FEF547B05        PXOR   MM2,[RBX+R15*2]+5
0FEFD2              PXOR   MM2,MM2
66460FEF647B05      PXOR   XMM12,[RBX+R15*2]+5
66450FEFE4          PXOR   XMM12,XMM12

                    ;------------ FEMMS ------------
0F0E                FEMMS

                    ;----------- PAVGUSB -----------
420F0F547B05BF      PAVGUSB   MM2,[RBX+R15*2]+5
0F0FD2BF            PAVGUSB   MM2,MM2
```

```
                        ;------------ PF2ID ------------
420F0F547B051D          PF2ID   MM2,[RBX+R15*2]+5
0F0FD21D                PF2ID   MM2,MM2

                        ;------------ PFACC ------------
420F0F547B05AE          PFACC   MM2,[RBX+R15*2]+5
0F0FD2AE                PFACC   MM2,MM2

                        ;------------ PFADD ------------
420F0F547B059E          PFADD   MM2,[RBX+R15*2]+5
0F0FD29E                PFADD   MM2,MM2

                        ;------------ PFCMPEQ ------------
420F0F547B05B0          PFCMPEQ   MM2,[RBX+R15*2]+5
0F0FD2B0                PFCMPEQ   MM2,MM2

                        ;------------ PFCMPGE ------------
420F0F547B0590          PFCMPGE   MM2,[RBX+R15*2]+5
0F0FD290                PFCMPGE   MM2,MM2

                        ;------------ PFCMPGT ------------
420F0F547B05A0          PFCMPGT   MM2,[RBX+R15*2]+5
0F0FD2A0                PFCMPGT   MM2,MM2

                        ;------------ PFMAX ------------
420F0F547B05A4          PFMAX   MM2,[RBX+R15*2]+5
0F0FD2A4                PFMAX   MM2,MM2

                        ;------------ PFMIN ------------
420F0F547B0594          PFMIN   MM2,[RBX+R15*2]+5
0F0FD294                PFMIN   MM2,MM2

                        ;------------ PFMUL ------------
420F0F547B05B4          PFMUL   MM2,[RBX+R15*2]+5
0F0FD2B4                PFMUL   MM2,MM2

                        ;------------ PFRCP ------------
420F0F547B0596          PFRCP   MM2,[RBX+R15*2]+5
0F0FD296                PFRCP   MM2,MM2

                        ;------------ PFRCPIT1 ----------
420F0F547B05A6          PFRCPIT1   MM2,[RBX+R15*2]+5
0F0FD2A6                PFRCPIT1   MM2,MM2

                        ;------------ PFRCPIT2 ----------
420F0F547B05B6          PFRCPIT2   MM2,[RBX+R15*2]+5
0F0FD2B6                PFRCPIT2   MM2,MM2

                        ;------------ PFRSQIT1 ----------
420F0F547B05A7          PFRSQIT1   MM2,[RBX+R15*2]+5
0F0FD2A7                PFRSQIT1   MM2,MM2

                        ;------------ PFRSQRT ------------
420F0F547B0597          PFRSQRT   MM2,[RBX+R15*2]+5
0F0FD297                PFRSQRT   MM2,MM2

                        ;------------ PFSUB ------------
420F0F547B059A          PFSUB   MM2,[RBX+R15*2]+5
0F0FD29A                PFSUB   MM2,MM2

                        ;------------ PFSUBR ------------
420F0F547B05AA          PFSUBR   MM2,[RBX+R15*2]+5
0F0FD2AA                PFSUBR   MM2,MM2

                        ;------------ PI2FD ------------
420F0F547B050D          PI2FD   MM2,[RBX+R15*2]+5
0F0FD20D                PI2FD   MM2,MM2

                        ;------------ PMULHRW ------------
420F0F547B05B7          PMULHRW   MM2,[RBX+R15*2]+5
0F0FD2B7                PMULHRW   MM2,MM2

                        ;------------ PREFETCH ----------
420F0D447B05            PREFETCH   [RBX+R15*2]+5
```

```
                    ;---------- PREFETCHW ----------
420F0D4C7B05        PREFETCHW   [RBX+R15*2]+5

                    ;------------ ADDPS ------------
460F58647B05        ADDPS   XMM12,[RBX+R15*2]+5
450F58E4            ADDPS   XMM12,XMM12

                    ;------------ ADDSS ------------
F3460F58647B05      ADDSS   XMM12,[RBX+R15*2]+5
F3450F58E4          ADDSS   XMM12,XMM12

                    ;------------ ANDNPS -----------
460F55647B05        ANDNPS   XMM12,[RBX+R15*2]+5
450F55E4            ANDNPS   XMM12,XMM12

                    ;------------ ANDPS ------------
460F54647B05        ANDPS   XMM12,[RBX+R15*2]+5
450F54E4            ANDPS   XMM12,XMM12

                    ;----------- CMPEQPS -----------
460FC2647B0500      CMPEQPS   XMM12,[RBX+R15*2]+5
450FC2E400          CMPEQPS   XMM12,XMM12

                    ;----------- CMPLTPS -----------
460FC2647B0501      CMPLTPS   XMM12,[RBX+R15*2]+5
450FC2E401          CMPLTPS   XMM12,XMM12

                    ;----------- CMPLEPS -----------
460FC2647B0502      CMPLEPS   XMM12,[RBX+R15*2]+5
450FC2E402          CMPLEPS   XMM12,XMM12

                    ;---------- CMPUNORDPS ---------
460FC2647B0503      CMPUNORDPS   XMM12,[RBX+R15*2]+5
450FC2E403          CMPUNORDPS   XMM12,XMM12

                    ;----------- CMPNEQPS ----------
460FC2647B0504      CMPNEQPS   XMM12,[RBX+R15*2]+5
450FC2E404          CMPNEQPS   XMM12,XMM12

                    ;----------- CMPNLTPS ----------
460FC2647B0505      CMPNLTPS   XMM12,[RBX+R15*2]+5
450FC2E405          CMPNLTPS   XMM12,XMM12

                    ;----------- CMPNLEPS ----------
460FC2647B0506      CMPNLEPS   XMM12,[RBX+R15*2]+5
450FC2E406          CMPNLEPS   XMM12,XMM12

                    ;----------- CMPORDPS ----------
460FC2647B0507      CMPORDPS   XMM12,[RBX+R15*2]+5
450FC2E407          CMPORDPS   XMM12,XMM12

                    ;----------- CMPEQSS -----------
F3460FC2647B0500    CMPEQSS   XMM12,[RBX+R15*2]+5
F3450FC2E400        CMPEQSS   XMM12,XMM12

                    ;----------- CMPLTSS -----------
F3460FC2647B0501    CMPLTSS   XMM12,[RBX+R15*2]+5
F3450FC2E401        CMPLTSS   XMM12,XMM12

                    ;----------- CMPLESS -----------
F3460FC2647B0502    CMPLESS   XMM12,[RBX+R15*2]+5
F3450FC2E402        CMPLESS   XMM12,XMM12

                    ;---------- CMPUNORDSS ---------
F3460FC2647B0503    CMPUNORDSS   XMM12,[RBX+R15*2]+5
F3450FC2E403        CMPUNORDSS   XMM12,XMM12

                    ;----------- CMPNEQSS ----------
F3460FC2647B0504    CMPNEQSS   XMM12,[RBX+R15*2]+5
F3450FC2E404        CMPNEQSS   XMM12,XMM12

                    ;----------- CMPNLTSS ----------
F3460FC2647B0505    CMPNLTSS   XMM12,[RBX+R15*2]+5
```

```
F3450FC2E405        CMPNLTSS   XMM12,XMM12

                    ;----------- CMPNLESS ----------
F3460FC2647B0506    CMPNLESS   XMM12,[RBX+R15*2]+5
F3450FC2E406        CMPNLESS   XMM12,XMM12

                    ;----------- CMPORDSS ----------
F3460FC2647B0507    CMPORDSS   XMM12,[RBX+R15*2]+5
F3450FC2E407        CMPORDSS   XMM12,XMM12

                    ;------------ COMISS -----------
460F2F647B05        COMISS   XMM12,[RBX+R15*2]+5
450F2FE4            COMISS   XMM12,XMM12

                    ;----------- CVTPI2PS ----------
460F2A647B05        CVTPI2PS   XMM12,[RBX+R15*2]+5
440F2AE2            CVTPI2PS   XMM12,MM2

                    ;----------- CVTPS2PI ----------
420F2D547B05        CVTPS2PI   MM2,[RBX+R15*2]+5
410F2DD4            CVTPS2PI   MM2,XMM12

                    ;----------- CVTSI2SS ----------
F3460F2A647B05      CVTSI2SS   XMM12,[RBX+R15*2]+5
F3440F2AE1          CVTSI2SS   XMM12,ECX

                    ;----------- CVTSS2SI ----------
F3420F2D4C7B05      CVTSS2SI   ECX,[RBX+R15*2]+5
F3410F2DCC          CVTSS2SI   ECX,XMM12

                    ;---------- CVTTPS2PI ----------
420F2C547B05        CVTTPS2PI   MM2,[RBX+R15*2]+5
410F2CD4            CVTTPS2PI   MM2,XMM12

                    ;---------- CVTTSS2SI ----------
F3420F2C4C7B05      CVTTSS2SI   ECX,[RBX+R15*2]+5
F3410F2CCC          CVTTSS2SI   ECX,XMM12

                    ;------------ DIVPS ------------
460F5E647B05        DIVPS   XMM12,[RBX+R15*2]+5
450F5EE4            DIVPS   XMM12,XMM12

                    ;------------ DIVSS ------------
F3460F5E647B05      DIVSS   XMM12,[RBX+R15*2]+5
F3450F5EE4          DIVSS   XMM12,XMM12

                    ;----------- FXRSTOR -----------
420FAE4C7B05        FXRSTOR   [RBX+R15*2]+5

                    ;------------ FXSAVE -----------
420FAE447B05        FXSAVE   [RBX+R15*2]+5

                    ;----------- LDMXCSR -----------
420FAE547B05        LDMXCSR   [RBX+R15*2]+5

                    ;----------- MASKMOVQ ----------
0FF7D2              MASKMOVQ   MM2,MM2

                    ;------------ MAXPS ------------
460F5F647B05        MAXPS   XMM12,[RBX+R15*2]+5
450F5FE4            MAXPS   XMM12,XMM12

                    ;------------ MAXSS ------------
F3460F5F647B05      MAXSS   XMM12,[RBX+R15*2]+5
F3450F5FE4          MAXSS   XMM12,XMM12

                    ;------------ MINPS ------------
460F5D647B05        MINPS   XMM12,[RBX+R15*2]+5
450F5DE4            MINPS   XMM12,XMM12

                    ;------------ MINSS ------------
F3460F5D647B05      MINSS   XMM12,[RBX+R15*2]+5
F3450F5DE4          MINSS   XMM12,XMM12
```

```
                    ;------------ MOVAPS -----------
460F28647B05        MOVAPS   XMM12,[RBX+R15*2]+5
450F28E4            MOVAPS   XMM12,XMM12
460F29647B05        MOVAPS   [RBX+R15*2]+5,XMM12

                    ;----------- MOVHLPS -----------
450F12E4            MOVHLPS   XMM12,XMM12

                    ;------------ MOVHPS -----------
460F16647B05        MOVHPS   XMM12,[RBX+R15*2]+5
460F17647B05        MOVHPS   [RBX+R15*2]+5,XMM12

                    ;----------- MOVLHPS -----------
450F16E4            MOVLHPS   XMM12,XMM12

                    ;------------ MOVLPS -----------
460F12647B05        MOVLPS   XMM12,[RBX+R15*2]+5
460F13647B05        MOVLPS   [RBX+R15*2]+5,XMM12

                    ;----------- MOVMSKPS ----------
410F50CC            MOVMSKPS   ECX,XMM12

                    ;----------- MOVNTPS -----------
460F2B647B05        MOVNTPS   [RBX+R15*2]+5,XMM12

                    ;------------ MOVNTQ -----------
420FE7547B05        MOVNTQ   [RBX+R15*2]+5,MM2

                    ;------------ MOVSS ------------
F3460F10647B05      MOVSS   XMM12,[RBX+R15*2]+5
F3460F11647B05      MOVSS   [RBX+R15*2]+5,XMM12
F3450F10E4          MOVSS   XMM12,XMM12

                    ;------------ MOVUPS -----------
460F10647B05        MOVUPS   XMM12,[RBX+R15*2]+5
460F11647B05        MOVUPS   [RBX+R15*2]+5,XMM12
450F10E4            MOVUPS   XMM12,XMM12

                    ;------------ MULPS ------------
460F59647B05        MULPS   XMM12,[RBX+R15*2]+5
450F59E4            MULPS   XMM12,XMM12

                    ;------------ MULSS ------------
F3460F59647B05      MULSS   XMM12,[RBX+R15*2]+5
F3450F59E4          MULSS   XMM12,XMM12

                    ;------------ ORPS -------------
460F56647B05        ORPS   XMM12,[RBX+R15*2]+5
450F56E4            ORPS   XMM12,XMM12

                    ;------------ PAVGB ------------
420FE0547B05        PAVGB   MM2,[RBX+R15*2]+5
0FE0D2              PAVGB   MM2,MM2
66460FE0647B05      PAVGB   XMM12,[RBX+R15*2]+5
66450FE0E4          PAVGB   XMM12,XMM12

                    ;------------ PAVGW ------------
420FE3547B05        PAVGW   MM2,[RBX+R15*2]+5
0FE3D2              PAVGW   MM2,MM2
66460FE3647B05      PAVGW   XMM12,[RBX+R15*2]+5
66450FE3E4          PAVGW   XMM12,XMM12

                    ;------------ PEXTRW -----------
0FC5CA12            PEXTRW   ECX,MM2,12H
66410FC5CC12        PEXTRW   ECX,XMM12,12H

                    ;------------ PINSRW -----------
0FC4D112            PINSRW   MM2,ECX,12H
420FC4547B0512      PINSRW   MM2,[RBX+R15*2]+5,12H

66440FC4E112        PINSRW   XMM12,ECX,12H
66460FC4647B0512    PINSRW   XMM12,[RBX+R15*2]+5,12H
```

```
                    ;------------ PMAXSW -----------
420FEE547B05        PMAXSW   MM2,[RBX+R15*2]+5
0FEED2              PMAXSW   MM2,MM2
66460FEE647B05      PMAXSW   XMM12,[RBX+R15*2]+5
66450FEEE4          PMAXSW   XMM12,XMM12

                    ;------------ PMAXUB -----------
420FDE547B05        PMAXUB   MM2,[RBX+R15*2]+5
0FDED2              PMAXUB   MM2,MM2
66460FDE647B05      PMAXUB   XMM12,[RBX+R15*2]+5
66450FDEE4          PMAXUB   XMM12,XMM12

                    ;------------ PMINSW -----------
420FEA547B05        PMINSW   MM2,[RBX+R15*2]+5
0FEAD2              PMINSW   MM2,MM2
66460FEA647B05      PMINSW   XMM12,[RBX+R15*2]+5
66450FEAE4          PMINSW   XMM12,XMM12

                    ;------------ PMINUB -----------
420FDA547B05        PMINUB   MM2,[RBX+R15*2]+5
0FDAD2              PMINUB   MM2,MM2
66460FDA647B05      PMINUB   XMM12,[RBX+R15*2]+5
66450FDAE4          PMINUB   XMM12,XMM12

                    ;----------- PMOVMSKB ----------
0FD7CA              PMOVMSKB   ECX,MM2
66410FD7CC          PMOVMSKB   ECX,XMM12

                    ;----------- PMULHUW -----------
420FE4547B05        PMULHUW   MM2,[RBX+R15*2]+5
0FE4D2              PMULHUW   MM2,MM2
66460FE4647B05      PMULHUW   XMM12,[RBX+R15*2]+5
66450FE4E4          PMULHUW   XMM12,XMM12

                    ;---------- PREFETCHT0 ---------
420F184C7B05        PREFETCHT0   [RBX+R15*2]+5

                    ;---------- PREFETCHT1 ---------
420F18547B05        PREFETCHT1   [RBX+R15*2]+5

                    ;---------- PREFETCHT2 ---------
420F185C7B05        PREFETCHT2   [RBX+R15*2]+5

                    ;--------- PREFETCHNTA ---------
420F18447B05        PREFETCHNTA   [RBX+R15*2]+5

                    ;------------ PSADBW -----------
420FF6547B05        PSADBW   MM2,[RBX+R15*2]+5
0FF6D2              PSADBW   MM2,MM2
66460FF6647B05      PSADBW   XMM12,[RBX+R15*2]+5
66450FF6E4          PSADBW   XMM12,XMM12

                    ;------------ PSHUFW -----------
0F70D212            PSHUFW   MM2,MM2,12H
420F70547B0512      PSHUFW   MM2,[RBX+R15*2]+5,12H

                    ;------------ RCPPS ------------
460F53647B05        RCPPS   XMM12,[RBX+R15*2]+5
450F53E4            RCPPS   XMM12,XMM12

                    ;------------ RCPSS ------------
F3460F53647B05      RCPSS   XMM12,[RBX+R15*2]+5
F3450F53E4          RCPSS   XMM12,XMM12

                    ;----------- RSQRTPS -----------
460F52647B05        RSQRTPS   XMM12,[RBX+R15*2]+5
450F52E4            RSQRTPS   XMM12,XMM12

                    ;----------- RSQRTSS -----------
F3460F52647B05      RSQRTSS   XMM12,[RBX+R15*2]+5
F3450F52E4          RSQRTSS   XMM12,XMM12

                    ;------------ SFENCE -----------
0FAEF8              SFENCE
```

```
                    ;------------ SHUFPS -----------
450FC6E412          SHUFPS   XMM12,XMM12,12H
460FC6647B0512      SHUFPS   XMM12,[RBX+R15*2]+5,12H

                    ;------------ SQRTPS -----------
460F51647B05        SQRTPS   XMM12,[RBX+R15*2]+5
450F51E4            SQRTPS   XMM12,XMM12

                    ;------------ SQRTSS -----------
F3460F51647B05      SQRTSS   XMM12,[RBX+R15*2]+5
F3450F51E4          SQRTSS   XMM12,XMM12

                    ;----------- STMXCSR -----------
420FAE5C7B05        STMXCSR   [RBX+R15*2]+5

                    ;------------ SUBPS ------------
460F5C647B05        SUBPS   XMM12,[RBX+R15*2]+5
450F5CE4            SUBPS   XMM12,XMM12

                    ;------------ SUBSS ------------
F3460F5C647B05      SUBSS   XMM12,[RBX+R15*2]+5
F3450F5CE4          SUBSS   XMM12,XMM12

                    ;----------- SYSENTER ----------
0F34                SYSENTER

                    ;----------- SYSEXIT -----------
0F35                SYSEXIT

                    ;----------- UCOMISS -----------
460F2E647B05        UCOMISS   XMM12,[RBX+R15*2]+5
450F2EE4            UCOMISS   XMM12,XMM12

                    ;----------- UNPCKHPS ----------
460F15647B05        UNPCKHPS   XMM12,[RBX+R15*2]+5
450F15E4            UNPCKHPS   XMM12,XMM12

                    ;----------- UNPCKLPS ----------
460F14647B05        UNPCKLPS   XMM12,[RBX+R15*2]+5
450F14E4            UNPCKLPS   XMM12,XMM12

                    ;------------ XORPS ------------
460F57647B05        XORPS   XMM12,[RBX+R15*2]+5
450F57E4            XORPS   XMM12,XMM12

                    ;------------ ADDPD ------------
66460F58647B05      ADDPD   XMM12,[RBX+R15*2]+5
66450F58E4          ADDPD   XMM12,XMM12

                    ;------------ ADDSD ------------
F2460F58647B05      ADDSD   XMM12,[RBX+R15*2]+5
F2450F58E4          ADDSD   XMM12,XMM12

                    ;------------ ANDNPD -----------
66460F55647B05      ANDNPD   XMM12,[RBX+R15*2]+5
66450F55E4          ANDNPD   XMM12,XMM12

                    ;------------ ANDPD ------------
66460F54647B05      ANDPD   XMM12,[RBX+R15*2]+5
66450F54E4          ANDPD   XMM12,XMM12

                    ;----------- CLFLUSH -----------
420FAE7C7B05        CLFLUSH   [RBX+R15*2]+5

                    ;----------- CMPEQPD -----------
66460FC2647B0500    CMPEQPD   XMM12,[RBX+R15*2]+5
66450FC2E400        CMPEQPD   XMM12,XMM12

                    ;----------- CMPEQSD -----------
F2460FC2647B0500    CMPEQSD   XMM12,[RBX+R15*2]+5
F2450FC2E400        CMPEQSD   XMM12,XMM12

                    ;----------- CMPLEPD -----------
```

```
66460FC2647B0502    CMPLEPD   XMM12,[RBX+R15*2]+5
66450FC2E402        CMPLEPD   XMM12,XMM12

                    ;----------- CMPLESD -----------
F2460FC2647B0502    CMPLESD   XMM12,[RBX+R15*2]+5
F2450FC2E402        CMPLESD   XMM12,XMM12

                    ;----------- CMPLTPD -----------
66460FC2647B0501    CMPLTPD   XMM12,[RBX+R15*2]+5
66450FC2E401        CMPLTPD   XMM12,XMM12

                    ;----------- CMPLTSD -----------
F2460FC2647B0501    CMPLTSD   XMM12,[RBX+R15*2]+5
F2450FC2E401        CMPLTSD   XMM12,XMM12

                    ;----------- CMPNEQPD ----------
66460FC2647B0504    CMPNEQPD   XMM12,[RBX+R15*2]+5
66450FC2E404        CMPNEQPD   XMM12,XMM12

                    ;----------- CMPNEQSD ----------
F2460FC2647B0504    CMPNEQSD   XMM12,[RBX+R15*2]+5
F2450FC2E404        CMPNEQSD   XMM12,XMM12

                    ;----------- CMPNLEPD ----------
66460FC2647B0506    CMPNLEPD   XMM12,[RBX+R15*2]+5
66450FC2E406        CMPNLEPD   XMM12,XMM12

                    ;----------- CMPNLESD ----------
F2460FC2647B0506    CMPNLESD   XMM12,[RBX+R15*2]+5
F2450FC2E406        CMPNLESD   XMM12,XMM12

                    ;----------- CMPNLTPD ----------
66460FC2647B0505    CMPNLTPD   XMM12,[RBX+R15*2]+5
66450FC2E405        CMPNLTPD   XMM12,XMM12

                    ;----------- CMPNLTSD ----------
F2460FC2647B0505    CMPNLTSD   XMM12,[RBX+R15*2]+5
F2450FC2E405        CMPNLTSD   XMM12,XMM12

                    ;----------- CMPORDPD ----------
66460FC2647B0507    CMPORDPD   XMM12,[RBX+R15*2]+5
66450FC2E407        CMPORDPD   XMM12,XMM12

                    ;----------- CMPORDSD ----------
F2460FC2647B0507    CMPORDSD   XMM12,[RBX+R15*2]+5
F2450FC2E407        CMPORDSD   XMM12,XMM12

                    ;---------- CMPUNORDPD ---------
66460FC2647B0503    CMPUNORDPD   XMM12,[RBX+R15*2]+5
66450FC2E403        CMPUNORDPD   XMM12,XMM12

                    ;---------- CMPUNORDSD ---------
F2460FC2647B0503    CMPUNORDSD   XMM12,[RBX+R15*2]+5
F2450FC2E403        CMPUNORDSD   XMM12,XMM12

                    ;------------ COMISD -----------
66460F2F647B05      COMISD   XMM12,[RBX+R15*2]+5
66450F2FE4          COMISD   XMM12,XMM12

                    ;----------- CVTDQ2PD ----------
F3460FE6647B05      CVTDQ2PD   XMM12,[RBX+R15*2]+5
F3450FE6E4          CVTDQ2PD   XMM12,XMM12

                    ;----------- CVTDQ2PS ----------
460F5B647B05        CVTDQ2PS   XMM12,[RBX+R15*2]+5
450F5BE4            CVTDQ2PS   XMM12,XMM12

                    ;----------- CVTPD2DQ ----------
F2460FE6647B05      CVTPD2DQ   XMM12,[RBX+R15*2]+5
F2450FE6E4          CVTPD2DQ   XMM12,XMM12

                    ;----------- CVTPD2PI ----------
66420F2D547B05      CVTPD2PI   MM2,[RBX+R15*2]+5
66410F2DD4          CVTPD2PI   MM2,XMM12
```

```
                    ;----------- CVTPD2PS ----------
66460F5A647B05       CVTPD2PS   XMM12,[RBX+R15*2]+5
66450F5AE4           CVTPD2PS   XMM12,XMM12

                    ;----------- CVTPI2PD ----------
66460F2A647B05       CVTPI2PD   XMM12,[RBX+R15*2]+5
66440F2AE2           CVTPI2PD   XMM12,MM2

                    ;----------- CVTPS2DQ ----------
66460F58647B05       CVTPS2DQ   XMM12,[RBX+R15*2]+5
66450F58E4           CVTPS2DQ   XMM12,XMM12

                    ;----------- CVTPS2PD ----------
460F5A647B05         CVTPS2PD   XMM12,[RBX+R15*2]+5
450F5AE4             CVTPS2PD   XMM12,XMM12

                    ;----------- CVTSD2SI ----------
F2420F2D4C7B05       CVTSD2SI   ECX,[RBX+R15*2]+5
F2410F2DCC           CVTSD2SI   ECX,XMM12

                    ;----------- CVTSD2SS ----------
F2420F5A747B05       CVTSD2SS   XMM6,[RBX+R15*2]+5
F2450F5AE4           CVTSD2SS   XMM12,XMM12

                    ;----------- CVTSI2SD ----------
F2460F2A647B05       CVTSI2SD   XMM12,[RBX+R15*2]+5
F2440F2AE1           CVTSI2SD   XMM12,ECX

                    ;----------- CVTSS2SD ----------
F3460F5A647B05       CVTSS2SD   XMM12,[RBX+R15*2]+5
F3450F5AE4           CVTSS2SD   XMM12,XMM12

                    ;----------- CVTTPD2DQ ----------
66460FE6647B05       CVTTPD2DQ   XMM12,[RBX+R15*2]+5
66450FE6E4           CVTTPD2DQ   XMM12,XMM12

                    ;----------- CVTTPD2PI ----------
66420F2C547B05       CVTTPD2PI   MM2,[RBX+R15*2]+5
66410F2CD4           CVTTPD2PI   MM2,XMM12

                    ;----------- CVTTPS2DQ ----------
F3460F5B647B05       CVTTPS2DQ   XMM12,[RBX+R15*2]+5
F3450F5BE4           CVTTPS2DQ   XMM12,XMM12

                    ;----------- CVTTSD2SI ----------
F2420F2C4C7B05       CVTTSD2SI   ECX,[RBX+R15*2]+5
F2410F2CCC           CVTTSD2SI   ECX,XMM12

                    ;------------ DIVPD ------------
66460F5E647B05       DIVPD   XMM12,[RBX+R15*2]+5
66450F5EE4           DIVPD   XMM12,XMM12

                    ;------------ DIVSD ------------
F2460F5E647B05       DIVSD   XMM12,[RBX+R15*2]+5
F2450F5EE4           DIVSD   XMM12,XMM12

                    ;------------ LFENCE -----------
420FAE6C7B05         LFENCE   [RBX+R15*2]+5

                    ;---------- MASKMOVDQU ---------
66450FF7E4           MASKMOVDQU   XMM12,XMM12

                    ;------------ MAXPD ------------
66460F5F647B05       MAXPD   XMM12,[RBX+R15*2]+5
66450F5FE4           MAXPD   XMM12,XMM12

                    ;------------ MAXSD ------------
F2460F5F647B05       MAXSD   XMM12,[RBX+R15*2]+5
F2450F5FE4           MAXSD   XMM12,XMM12

                    ;------------ MFENCE -----------
420FAE747B05         MFENCE   [RBX+R15*2]+5
```

```
                    ;------------ MINPD ------------
66460F5D647B05       MINPD   XMM12,[RBX+R15*2]+5
66450F5DE4           MINPD   XMM12,XMM12

                    ;------------ MINSD ------------
F2460F5D647B05       MINSD   XMM12,[RBX+R15*2]+5
F2450F5DE4           MINSD   XMM12,XMM12

                    ;------------ MOVAPD -----------
66460F28647B05       MOVAPD   XMM12,[RBX+R15*2]+5
66450F28E4           MOVAPD   XMM12,XMM12
66460F29647B05       MOVAPD   [RBX+R15*2]+5,XMM12

                    ;----------- MOVDQ2Q -----------
F2410FD6D4           MOVDQ2Q   MM2,XMM12

                    ;------------ MOVDQA -----------
66460F6F647B05       MOVDQA   XMM12,[RBX+R15*2]+5
66450F6FE4           MOVDQA   XMM12,XMM12
66420F7F647B05       MOVDQA   [RBX+R15*2]+5,XMM12

                    ;------------ MOVDQU -----------
F3460F6F647B05       MOVDQU   XMM12,[RBX+R15*2]+5
F3450F6FE4           MOVDQU   XMM12,XMM12
F3420F7F647B05       MOVDQU   [RBX+R15*2]+5,XMM12

                    ;------------ MOVHPD -----------
66460F16647B05       MOVHPD   XMM12,[RBX+R15*2]+5
66460F17647B05       MOVHPD   [RBX+R15*2]+5,XMM12

                    ;------------ MOVLPD -----------
66460F12647B05       MOVLPD   XMM12,[RBX+R15*2]+5
66460F13647B05       MOVLPD   [RBX+R15*2]+5,XMM12

                    ;----------- MOVMSKPD ----------
66410F50CC           MOVMSKPD   ECX,XMM12

                    ;----------- MOVNTDQ -----------
66460FE7647B05       MOVNTDQ   [RBX+R15*2]+5,XMM12

                    ;------------ MOVNTI -----------
0FC30C6B             MOVNTI   D PTR [RBX+RBP*2],ECX

                    ;----------- MOVNTPD -----------
66460F2B647B05       MOVNTPD   [RBX+R15*2]+5,XMM12

                    ;----------- MOVQ2DQ -----------
F3440FD6D4           MOVQ2DQ   XMM12,MM2

                    ;------------ MOVUPD -----------
660F10647B05         MOVUPD   XMM12,[RBX+R15*2]+5
660F10E4             MOVUPD   XMM12,XMM12
66460F11647B05       MOVUPD   [RBX+R15*2]+5,XMM12

                    ;------------ MULPD ------------
66460F59647B05       MULPD   XMM12,[RBX+R15*2]+5
66450F59E4           MULPD   XMM12,XMM12

                    ;------------ MULSD ------------
F2460F59647B05       MULSD   XMM12,[RBX+R15*2]+5
F2450F59E4           MULSD   XMM12,XMM12

                    ;------------- ORPD ------------
66460F56647B05       ORPD   XMM12,[RBX+R15*2]+5
66450F56E4           ORPD   XMM12,XMM12

                    ;------------ PADDQ ------------
420FD4547B05         PADDQ   MM2,[RBX+R15*2]+5
0FD4D2               PADDQ   MM2,MM2
66460FD4647B05       PADDQ   XMM12,[RBX+R15*2]+5
66450FD4E4           PADDQ   XMM12,XMM12

                    ;------------ PAUSE ------------
F390                 PAUSE
```

```
                    ;----------- PMULUDQ -----------
420FF4547B05        PMULUDQ   MM2,[RBX+R15*2]+5
0FF4D2              PMULUDQ   MM2,MM2
460FF4647B05        PMULUDQ   XMM12,[RBX+R15*2]+5
450FF4E4            PMULUDQ   XMM12,XMM12

                    ;------------ PSHUFD -----------
66460F70647B0512    PSHUFD   XMM12,[RBX+R15*2]+5,12H
66450F70E412        PSHUFD   XMM12,XMM12,12H

                    ;----------- PSHUFHW -----------
F3460F70647B0512    PSHUFHW   XMM12,[RBX+R15*2]+5,12H
F3450F70E412        PSHUFHW   XMM12,XMM12,12H

                    ;----------- PSHUFLW -----------
F2460F70647B0512    PSHUFLW   XMM12,[RBX+R15*2]+5,12H
F2450F70E412        PSHUFLW   XMM12,XMM12,12H

                    ;------------ PSLLDQ -----------
66440F73FC12        PSLLDQ   XMM12,12H

                    ;------------ PSRLDQ -----------
66440F73F412        PSRLDQ   XMM12,12H

                    ;------------ PSUBQ ------------
420FFB547B05        PSUBQ   MM2,[RBX+R15*2]+5
0FFBD2              PSUBQ   MM2,MM2
460FFB647B05        PSUBQ   XMM12,[RBX+R15*2]+5
450FFBE4            PSUBQ   XMM12,XMM12

                    ;---------- PUNPCKLQDQ ---------
66460F6C647B05      PUNPCKLQDQ   XMM12,[RBX+R15*2]+5
66450F6CE4          PUNPCKLQDQ   XMM12,XMM12

                    ;---------- PUNPCKHQDQ ---------
66460F6D647B05      PUNPCKHQDQ   XMM12,[RBX+R15*2]+5
66450F6DE4          PUNPCKHQDQ   XMM12,XMM12

                    ;------------ SHUFPD -----------
66460FC6647B0512    SHUFPD   XMM12,[RBX+R15*2]+5,12H
66450FC6E412        SHUFPD   XMM12,XMM12,12H

                    ;------------ SQRTPD -----------
66460F51647B05      SQRTPD   XMM12,[RBX+R15*2]+5
66450F51E4          SQRTPD   XMM12,XMM12

                    ;------------ SQRTSD -----------
F2460F51647B05      SQRTSD   XMM12,[RBX+R15*2]+5
F2450F51E4          SQRTSD   XMM12,XMM12

                    ;------------ SUBPD ------------
66460F5C647B05      SUBPD   XMM12,[RBX+R15*2]+5
66450F5CE4          SUBPD   XMM12,XMM12

                    ;------------ SUBSD ------------
F2460F5C647B05      SUBSD   XMM12,[RBX+R15*2]+5
F2450F5CE4          SUBSD   XMM12,XMM12

                    ;----------- UCOMISD -----------
66460F2E647B05      UCOMISD   XMM12,[RBX+R15*2]+5
66450F2EE4          UCOMISD   XMM12,XMM12

                    ;----------- UNPCKHPD ----------
66460F15647B05      UNPCKHPD   XMM12,[RBX+R15*2]+5
66450F15E4          UNPCKHPD   XMM12,XMM12

                    ;----------- UNPCKLPD ----------
66460F14647B05      UNPCKLPD   XMM12,[RBX+R15*2]+5
66450F14E4          UNPCKLPD   XMM12,XMM12

                    ;------------ XORPD ------------
66460F57647B05      XORPD   XMM12,[RBX+R15*2]+5
66450F57E4          XORPD   XMM12,XMM12
```

```
                    ;----------- FISTT16P ----------
42DF4C7B05          FISTT16P   [RBX+R15*2]+5

                    ;----------- FISTT32P ----------
42DB4C7B05          FISTT32P   [RBX+R15*2]+5

                    ;----------- FISTT64P ----------
42DD4C7B05          FISTT64P   [RBX+R15*2]+5

                    ;----------- ADDSUBPD ----------
66460FD0647B05      ADDSUBPD   XMM12,[RBX+R15*2]+5
66450FD0E4          ADDSUBPD   XMM12,XMM12

                    ;----------- ADDSUBPS ----------
F2460FD0647B05      ADDSUBPS   XMM12,[RBX+R15*2]+5
F2450FD0E4          ADDSUBPS   XMM12,XMM12

                    ;----------- MOVDDUP -----------
F2460F12647B05      MOVDDUP   XMM12,[RBX+R15*2]+5
F2450F12E4          MOVDDUP   XMM12,XMM12

                    ;----------- MOVSHDUP ----------
F3460F16647B05      MOVSHDUP   XMM12,[RBX+R15*2]+5
F3450F16E4          MOVSHDUP   XMM12,XMM12

                    ;----------- MOVSLDUP ----------
F3460F12647B05      MOVSLDUP   XMM12,[RBX+R15*2]+5
F3450F12E4          MOVSLDUP   XMM12,XMM12

                    ;------------ LDDQU ------------
F2460FF0647B05      LDDQU   XMM12,[RBX+R15*2]+5

                    ;------------ HADDPD -----------
66460F7C647B05      HADDPD   XMM12,[RBX+R15*2]+5
66450F7CE4          HADDPD   XMM12,XMM12

                    ;------------ HADDPS -----------
F2460F7C647B05      HADDPS   XMM12,[RBX+R15*2]+5
F2450F7CE4          HADDPS   XMM12,XMM12

                    ;------------ HSUBPD -----------
66460F7D647B05      HSUBPD   XMM12,[RBX+R15*2]+5
66450F7DE4          HSUBPD   XMM12,XMM12

                    ;------------ HSUBPS -----------
F2460F7D647B05      HSUBPS   XMM12,[RBX+R15*2]+5
F2450F7DE4          HSUBPS   XMM12,XMM12

                    ;----------- MONITOR -----------
0F01C8              MONITOR

                    ;------------ MWAIT ------------
0F01C9              MWAIT

                    ;----------- VBLENDPD ----------
C423190D647B0512    VBLENDPD   XMM12,XMM12,[RBX+R15*2]+5,12H
C443190DE412        VBLENDPD   XMM12,XMM12,XMM12,12H
C4431D0DE412        VBLENDPD   YMM12,YMM12,YMM12,12H
C4231D0D647B0512    VBLENDPD   YMM12,YMM12,[RBX+R15*2]+5,12H
```

Предметный указатель английских названий