ИНСТИТУТ ЯДЕРНОЙ ФИЗИКИ СО АН СССР

В.Н. Гончаров, В.Р. Козак, А.А. Никифоров

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДЛЯ МИКРОЭВМ ОДРЕНОК. ФОРТРАН-1900

ПРЕПРИНТ 88-81

НОВОСИБИРСК

Программное обеспечение для микроЭВМ Одренок. Фортран-1900

В.Н. Гончаров, В.Р. Козак, А.А. Никифоров

Институт ядерной физики
630090, Новосибирск 90, СССР

АННОТАЦИЯ

В препринте описан способ компиляции программ с языка ФОРТРАН-1900 посредством стандартного компилятора для ЭВМ ILC-1900 (#XFAP) под управлением специальной программы (#FORT), мониторирующей обращения к внешним устройствам и обеспечивающей стыковку компилятора с операционной и файловой системой микроЭВМ Одренок. Описаны также изменения библиотеки компилятора, обеспечивающие стыковку операторов ввода/вывода компилированной программы с существующей системой, а также адаптация графического пакета PIGS, позволяющая использовать его в программах, написанных на языке ФОРТРАН-1900. В Приложении дано описание работы с мониторирующей программой для пользователя, кратко описан язык ФОРТРАН-1900 и некоторые особенности операций ввода/вывода, связанные с изменением соответствующих подпрограмм.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В ИЯФ СО АН СССР в 1984 г. была создана микроЭВМ Одренок [1]. Достаточно высокое быстродействие, повышенная точность представления чисел (24 разряда для целых и 48 разрядов для чисел с плавающей точкой одинарной длины), размер оперативной памяти (64К 24-разрядных слов), встроенные команды работы с аппаратурой в стандарте КАМАК обусловили ее широкое применение для управления экспериментальными физическими установками [2, 3]. Для решения задач управления хорошо подходил и применяемый язык программирования — TRAN [4], созданный для мини-ЭВМ ОДРА-1325. Ограничение набора операторов фортрановского типа с некоторыми расширениями позволило создать компактный компилятор с высокой скоростью процесса компиляции. Достаточная мощность языка и легкость его освоения привели к монополизации им системы подготовки программ как на мини-ЭВМ ОДРА-1325, так и на микроЭВМ Одренок. Этот язык вполне устраивал как физиков — создателей программ управления установками, так и разработчиков электронной аппаратуры, широко использующих ЭВМ для наладки и тестирования аппаратуры.

Мощность микроЭВМ и ее постоянно растущая доступность привели к расширению круга пользователей за счет так называемых «считарей» — физиков, пишущих сложные расчетные программы, как правило, на крупных ЭВМ. Этих пользователей возмож-

ности Одренка зачастую вполне устраивали (для не очень больших программ), так как более низкая производительность с лихвой окупалась постоянным доступом к своей персональной ЭВМ и интерактивным режимом работы. Однако, все такие пользователи традиционно писали свои программы на языке ФОРТРАН, привыкли к нему, имели значительное количество созданных программ и подпрограмм, которыми хотели пользоваться и на Одренке, а также нередко обменивались программами с пользователями других типов ЭВМ. Потребности нового круга пользователей Одрят инициировали работы по постановке на микроЭВМ ФОРТРАН-компилятора.

После появления ФОРТРАН-компилятора на Одренке обнаружилось, что существовал круг пользователей, всегда тяготевших к этому языку. Часть пользователей писала на языке TRAN подпрограммы, имитирующие работу с комплексными числами, которых в этом языке нет, некоторым пользователям не хватало точности чисел с плавающей точкой одинарной длины, часть пользователей заимствовала ФОРТРАН-программы с ЭВМ других типов и испытывала трудности с переводом их на другой язык. Некоторых пользователей привлекали большие возможности языка, а других — развитая диагностика ошибок в процессе исполнения программы. Таким образом, круг реальных пользователей ФОРТРАНа на микроЭВМ Одренок оказался существенно шире, чем предполагалось вначале.

Ниже описывается способ постановки стандартного компилятора фирмы ICL на микроЭВМ Одренок, функции и способ построения программы-препроцессора, изменения библиотеки компилятора, адаптация известного пакета графических средств PIGS к системе компиляции на языке ФОРТРАН-1900. В Приложении приведено описание препроцессора для пользователя, описание особенностей ФОРТРАНа-1900 на микроЭВМ Одренок, а также краткое описание языка ФОРТРАН-1900.

ПОСТАНОВКА ФОРТРАН-КОМПИЛЯТОРА НА МИКРОЭВМ ОДРЕНОК

После возникновения потребности в компиляторе с языка ФОРТРАН рассматривались различные варианты решения этой проблемы. Самый очевидный вариант — написать соответствующую программу — был отброшен с самого начала, так как считалось,

что круг пользователей ФОРТРАНа будет весьма ограниченным и затраты времени на создание нового компилятора являются нецелесообразными. Вариант с использованием текста какого-либо компилятора также был отброшен: существующие тексты написаны на языках, которые не реализованы на Одренке, а перевод таких текстов на TRAN или PLAN является трудоемким как в переводе, так и в отладке.

Вследствие описанных причин было принято решение воспользоваться совместимостью по системе команд микроЭВМ Одренок и машин серии ОДРА (ICL-1900) и попытаться использовать один из стандартных компиляторов фирмы ICL. Главная проблема заключалась в том, что при создании операционных систем для мини-ЭВМ ОДРА и микроЭВМ Одренок, были нарушены спецификации ввода/вывода фирмы ICL и, соответственно, программы под стандартной операционной системой и программы под нашей операционной системой оказались несовместимыми по большинству экстракодов (кодов, интерпретируемых операционной системой). Таким образом, для запуска стандартного компилятора фирмы ICL, необходимо либо обслужить эти экстракоды операционной системой, что весьма затруднительно, так как одни и те же экстракоды должны для различных программ трактоваться разными способами, либо заменить их соответствующими последовательностями непосредственно в бинарной программе. Надо отметить, что в 1984 г. группой физиков такая работа была проведена: в не используемую при работе область бинарной программы были введены коды, позволившие считывать входной файл с барабана и транслировать его. Однако, низкий уровень сервиса при таком подходе и ограниченные ресурсы мини-ЭВМ ОДРА-1325, обусловили игнорирование такого компилятора пользователями и, через некоторое время, он был утрачен.

В рамках существующей операционной системы наиболее предпочтительным представляется следующий подход: все взаимодействие с операционной системой и пользователем осуществляется специальной программой-монитором, которая должна интерпретировать запросы на ввод/вывод программы компилятора. Такой подход позволяет дать пользователю максимальный сервис, обеспечить стыковку компилятора как с файловой системой, так и с имеющейся системой компиляции. Кроме этого, на мониторирующую программу можно возложить также и функции препроцессора, производящего дополнительную обработку текстовых строк.

Отдельная программа также дает возможность без особых ухищрений добавлять новые функции.

В рамках описанного подхода в 1986 г. была создана программа #FORT, которая осуществляет стыковку компилятора как с файловой системой, так и с системой компиляции. Для работы на микроЭВМ Одренок, на которой отсутствует режим расширенной адресации, возможно использовать только некоторые из существующих стандартных ФОРТРАН-компиляторов, из которых был выбран #XFAP: сравнительно компактный, но достаточно мощный. #XFAP предназначен для работы с перфолентой или с перфокартами. Программа #FORT обрабатывает входные строки с терминала либо из входного файла, распознает и исполняет директивы, относящиеся к ней, а остальные строки преобразует в перфоленточный стандарт и передает их компилятору #XFAP. От компилятора #FORT получает либо запрос на вывод листинга, либо запрос на ввод следующей входной строки, либо запрос на вывод полукомпилированного рекорда. Поскольку структура файла трансляции (SCRF) в существующей системе отличается от той, с которой работает #XFAP, мониторирующая программа дополнительно преобразует его и записывает в файл. По окончании компиляции автоматически запускается консолидатор и процесс сборки программы не отличается от принятого.

В фортрановской программе можно использовать практически все подпрограммы, написанные для языка TRAN, что существенно облегчает работу на ФОРТРАНе с конкретными нестандартными устройствами (например, с аппаратурой в стандарте КАМАК).

После стыковки компилятора с существующей файловой системой можно было создавать программы на языке ФОРТРАН-1900, которые могли использовать все операторы языка, кроме операторов ввода/вывода. Вместо операторов ввода/вывода можно было воспользоваться имеющимися терминальными и барабанными пакетами. Однако, гибкость и универсальность стандартных операторов ввода/вывода инициировали их реализацию. Задача существенно облегчалась тем обстоятельством, что ФОРТРАН-компиляторы фирмы ICL для реализации ввода/вывода использовали набор подпрограмм в стандартной библиотеке — для каждого типа устройства ввода/вывода существовала отдельная подпрограмма. В процессе компиляции в зависимости от используемых в программе устройств к программе присоединялись соответствующие подпрограммы. Таким образом, для обеспечения стыковки ФОРТРАН-программы с терминалом и файловой системой, необходимо

было только изменить соответствующие подпрограммы, что и было проделано авторами.

МОНИТОРИРУЮЩАЯ ПРОГРАММА #FORT

Для управления компилятором посредством специальной программы было использовано то обстоятельство, что микроЭВМ Одренок не имеет защиты от превышения адреса программой. Это дает любой программе доступ к командам и данным другой программы, расположенной ниже ее. Для правильного взаимодействия программы должны располагаться непосредственно друг за другом, иначе (если между ними есть еще одна программа) могут быть ошибки, связанные с реорганизацией памяти (например, программа, находящаяся между взаимодействующими программами, изменяет свой размер или завершает свою работу).

Программа #FORT после запуска загружает в оперативную память компилятор #XFAP. После этого #FORT запрашивает у операционной системы свой размер для определения относительного расположения в оперативной памяти компилятора и вычисляет его контрольную сумму. Если контрольная сумма не совпадает с той, которая должна быть, то программа останавливается с соответствующим сообщением на терминале. Несовпадение контрольной суммы может быть, как правило, в двух случаях: если между мониторирующей программой и компилятором загрузилась другая программа, либо при порче компилятора на диске.

После проверки контрольной суммы, #FORT вносит изменения в команды компилятора. #XFAP имеет в своем теле несколько десятков экстракодов, относящихся к внешним устройствам (прикрепление перфоратора или читника перфоленты, освобождение их, промотка перфоленты и т. д.). Все несущественные команды заменяются нейтральной командой, а там, где проверяются ответы (например, ответ на запрос прикрепления устройства), подставляется нужный ответ. После этого остаются три команды ввода/вывода: запрос на ввод строки и запрос на вывод либо строки листинга или полукомпилированного рекорда, а также команда вывода сообщения о завершении компиляции. Эти команды заменяются экстракодами системе: остановить #XFAP и стартовать #FORT с нужного входа.

После изменения компилятора #FORT проверяет наличие на

диске файла для компиляции (SCRF) и, если он существует, то программа уставляет внутренние указатели на нужные адреса, и запускает компилятор #XFAP с 20-й ячейки. В противном случае на терминале сообщается об ошибке. После запуска компилятора, мониторирующая программа останавливает себя с интервалом повторной активации около 10 секунд. Это вызвано тем обстоятельством, что в процессе компиляции может возникнуть внутренняя ошибка в компиляторе. В таком случае компилятор будет остановлен. По истечении периода активации, #FORT будет запущен операционной системой, зарегистрирует соответствующую ошибку и сообщит об этом на терминале.

При нормальной работе #XFAP подготовит свои внутренние таблицы и перезапустит #FORT с 21-й ячейки, остановив себя. После такого перезапуска, мониторирующая программа считывает содержимое адресного счетчика компилятора (8-е слово) и определяет причину запроса. Это может быть запрос входной строки или какая-то другая причина. Все запросы обрабатываются различным образом.

В случае запроса входной строки #FORT должен передать компилятору текстовую строку в перфоленточном формате. Возможный источник входной строки — терминал либо файл. Обработка входной строки сделана единообразным способом: программа имеет стек на 10 файлов (в стеке хранятся имена файлов и текущие адреса). Если стек пуст (например, сразу после запуска), ввод производится с терминала. Если в строке указано: читать из файла, то указатель стека инкрементируется и новое имя файла и текущий адрес записываются в стек. При достижении конца текста в файле, #FORT декрементирует указатель стека и продолжает считывать исходный текст из предыдущего файла, а если указатель оказался равным нулю, то с терминала. Чтение текста из файла буферируется для ускорения работы и, если следующая запрошенная строка оказывается уже в буфере, обращения к диску не происходит. Механизм буферирования каких-либо особенностей не содержит и поэтому не описывается.

При вводе строки с терминала, можно пользоваться стандартными редакторскими клавишами ⟨IC⟩, ⟨DC⟩ и перемещением курсора влево или вправо. Завершается ввод строки так же, как и для компилятора #TRAN, либо клавишей ⟨ETX⟩, либо ⟨LF⟩.

После завершения ввода с терминала строка преобразуется в рекорд, такой же, как и рекорды в текстовом файле, и далее обрабатывается аналогично строкам из файла. Каждая строка про-

сматривается мониторирующей программой и, если распознается директива для нее, то она исполняется, а в компилятор может не передаваться. Если строка не предназначена монитору, то она преобразуется в перфоленточный вид, пересылается во входной буфер компилятора, а сам компилятор запускается с адреса, следующего за адресом остановки.

При запросе компилятора на вывод листинга #FORT считывает строку, подготовленную компилятором, и выводит строку на терминал. Каких-либо особенностей процесс вывода листинга не имеет. Вывод листинга можно переадресовать на принтер, подключенный к другому терминальному каналу.

При выводе полукомпилированных рекордов необходимо некоторое преобразование форматов. Во-первых, необходимо преобразовать перфоленточный формат полукомпилированного рекорда в формат, используемый в существующей системе компиляции. Во-вторых, в существующей системе компиляции и системе компиляции фирмы ICL не совпадают наборы типов полукомпилированных рекордов. Мониторирующая программа осуществляет соответствующие преобразования.

#XFAP имеет встроенный консолидатор, который автоматически запускается по получению строки с оператором FINISH. Поэтому, когда мониторирующая программа обнаруживает такую строку, она заменяет ее строкой, содержащей символ «;», при приеме которой компилятор не запускает свой консолидатор, а просто сообщает о конце процесса компиляции. Это сообщение заменяется командой монитору, который делает соответствующие записи в файл трансляции (записывается имя программы, ее приоритет, список библиотек) и запускает программу #CNSL, которая составляет протокол сборки и, в свою очередь, загружает сборщик программы (#XP@@). Таким образом, процесс сборки не отличается от сборки программы, транслированной компилятором #TRAN. Это позволяет объединять модули, предварительно компилированные различными компиляторами.

Встроенный препроцессор в мониторирующей программе просматривает каждую строку, которая может либо содержать директиву для него, либо быть строкой, подлежащей некоторым преобразованиям, либо должна передаваться компилятору без изменений. Набор команд невелик и включает в себя лишь необходимый минимум, обеспечивающий традиционные возможности для пользователя: управление листингом, ссылки на библиотеки и другие текстовые файлы и операторы условной компиляции. Кроме распозна-

вания и исполнения директив, #FORT осуществляет преобразование восьмеричных констант, которых не существует в ФОРТРАНе-1900, в десятичные, а также преобразование литеральных записей в Н-формат. Многие пользователи привыкли к литеральным записям в языке TRAN, которые очень удобны, так как позволяют не подсчитывать число символов при выводе и поэтому препроцессор делает это преобразование по стандартным правилам: литерал начинается с апострофа и завершается им, а два апострофа стоящие рядом, трактуются как один апостроф. В строке может быть несколько литералов, однако литерал не должен разбиваться на две строки.

ПОДПРОГРАММЫ ВВОДА/ВЫВОДА

В ФОРТРАН-программе для работы с терминалом, принтером и файлами, можно использовать все подпрограммы и пакеты, созданные ранее на микроЭВМ Одренок. Однако, несмотря на все очевидные достоинства специализированных подпрограмм, работа с этими устройствами посредством стандартных операторов ввода/вывода языка ФОРТРАН имеет как свои преимущества, так и своих сторонников, что инициировало работу по стыковке фортрановских программ с перечисленными внешними устройствами.

Для обеспечения работы с терминалами, принтерами и файлами посредством стандартных операторов языка ФОРТРАН необходимо заменить соответствующие подпрограммы ввода/вывода. Поскольку в использованном перфоленточном компиляторе #XFAP не существуют внешние устройства типа терминала, магнитного диска или барабана, было принято решение использовать для работы программы с терминалом устройства TP и TR, соответствующие перфоратору и считывателю с перфоленты. Для работы с принтером используется устройство LP, а для работы с файлами задействованы устройства типа CP, CR — первоначально перфоратор и считыватель с перфокарт. Кроме этого, можно использовать устройство типа CD — вывод на консоль оператора. При использовании этого оператора вывод будет производиться на терминал. Если при работе с перфоратором допускается только запись, а при работе с читником — только чтение, то после замены подпрограмм и чтение и запись допускаются на любое устройство (для терминала или файла допустимо и то, и другое), а выбор

допустимых операций осуществляется оператором в описателе программы. Если устройство объявлено только для ввода, то запись на него вызовет остановку по ошибке, и т. п.

Из программы допускается только чтение или запись в уже имеющиеся файлы. Создание, уничтожение и другие манипуляции с файлами необходимо осуществлять соответствующими подпрограммами или специальными программами.

Следует отметить, что развитость операций ввода/вывода дает особенно большие удобства при работе с текстовыми файлами, так как они позволяют работать со строками. Существуют операторы, которые перемещают скрытый в подпрограммах указатель адреса на первую строку файла (REWIND) или на предыдущую строку файла (BACKSPACE), причем пользователю не нужно следить за адресами и правильностью рекордовой структуры. Кроме этого, значительные удобства предоставляет однородность работы с разнородными устройствами, например, вывод на терминал можно легко продублировать выводом в текстовый файл и т. п.

В ФОРТРАН-1900 программа состоит из описателя и тела программы, состоящего из подпрограмм. В описателе определяются устройства ввода/вывода, имя программы, ее приоритет, уровень трассировки (диагностики), уровень листинга и прочие нестандартные описания. В подпрограммах же используются стандартные операторы языка ФОРТРАН-4 с некоторыми отклонениями. Таким образом, описатель можно назвать нестандартной частью программы. В нем соответствующими операторами (INPUT, OUTPUT, USE) определяются логические номера каналов ввода/вывода и типы устройств, а также некоторая дополнительная информация, в том числе и требуемая новыми подпрограммами (например, имя файла и комментарий к нему: первые два слова файла, и т. п.).

Поскольку описаний подпрограмм ввода/вывода компилятора найдено не было, они были реассемблированы из полукомпилированной формы в библиотеке, и после анализа взаимодействия с вызывающей программой написаны заново. Это было проделано с подпрограммами %FIOLP, %FIOPT и %FIOCARD.

Из особенностей реализации следует отметить лишь то, что ввод информации сделан на основе редактора строки, что позволяет при вводе информации пользоваться стандартными редакторскими символами терминала (курсор влево и вправо, IC, DC). Основная подпрограмма вывода на терминал и принтер способна выводить информацию на терминал типа VT340 или другой анало-

гичный, принтеры типов DZM-180, D-180, D100, ROBOTRON CM 6329.02-M. Тип принтера указывается при декларации устройства LP пользователем в текстовом виде.

Описание деклараций устройств ввода/вывода, дополнительная диагностика подпрограмм ввода/вывода и примеры их использования приведены в Приложении.

ГРАФИЧЕСКИЙ ПАКЕТ PIGS

В Институте некоторое время назад на ЭВМ различных типов был внедрен графический пакет PIGS. Он может работать на ЭВМ Электроника-79 под операционной системой RSX-11 и на микроЭВМ Электроника-60 под операционной системой RT-11. Графический пакет поддерживает такие устройства, как ЦДР, графический терминал (модернизированный VDT 52100C), графопостроитель. Несмотря на то, что подавляющее большинство пользователей вполне устраивают возможности подпрограмм библиотеки для ЦДР-2, нередко приходится создавать относительно сложные программные конструкции при создании графических образов, особенно в трехмерном пространстве. PIGS представляется нам наиболее подходящим для внедрения на микроЭВМ Одренок, так как он не только обладает большими возможностями, но и довольно компактен, а также существует на ЭВМ других типов как в Институте, так и вне его. Пользуясь пакетом, пользователь работает в двух- или трехмерной системе координат, определенной им самим. Манипуляции в этой системе координат пакет проецирует в выбранную двухмерную мировую систему координат. При этом система координат пользователя может быть повернута и сдвинута. Выбранную пользователем прямоугольную часть мировой системы координат (окно в мировой системе координат) пакет проецирует в окно в двухмерной целочисленной (8190×6240) системе координат пакета. Все действия во внутренней системе координат пакета записываются в метафайл, который может быть перенесен на другой тип ЭВМ и соответствующей программой быть выведен на графическом устройстве этой другой ЭВМ. Пакет позволяет проводить линии различной толщины, штриховые и штрих-пунктирные линии, выводить текстовую информацию как средствами аппаратуры (например, посредством знакогенератора ЦДР-2),

так и программными средствами. В случае использования программного знакогенератора можно изменять масштаб и ориентацию символов.

Вторая причина, инициировавшая работу по установке пакета на микроЭВМ Одренок, была скорее политическая. Дело в том, что в среде пользователей господствует мнение, что микроЭВМ Одренок не только несовместима по системе команд с другими типами ЭВМ, но даже и по операторам ФОРТРАНа и авторам хотелось оценить уровень этой несовместимости по трудоемкости перевода какого-либо стандартного пакета на язык ФОРТРАН-1900. Как показал опыт адаптации пакета PIGS к ФОРТРАНу-1900, трудности в основном были связаны с необходимостью написания подпрограмм, реализующих стыковку с конкретными устройствами, стандартная же часть пакета не создала ощутимых трудностей.

В реализации пакета на микроЭВМ Одренок поддерживаются только ЦДР-2 и графическая DZM-180 — самые распространенные устройства. В ближайшее время предполагается реализация поддержки других принтеров с графическими возможностями (ROBOTRON CM 6329.02-M, D-100).

Подробное описание пакета с особенностями варианта для микроЭВМ Одренок в настоящее время распространяется среди пользователей стандартным образом.

ЗАКЛЮЧЕНИЕ

Среди пользователей микроЭВМ Одренок распространено мнение, что для эффективной работы на микроЭВМ требуется иметь специализированную операционную систему и специализированный язык программирования, ориентированные на конкретные применения. Авторы считают, что эта точка зрения отражает мнение бывших пользователей ЭВМ ОДРА-1325 и ОДРА-1305, работавших на комплексах ВЭПП-3, ВЭПП-4. В свое время это представление сыграло очень полезную роль, способствовав созданию на мини-ЭВМ с ограниченными ресурсами программную среду, обеспечившую плодотворную работу непрофессиональных программистов по созданию управляющих программ для крупных физических установок. Однако, в настоящее время круг пользователей микроЭВМ Одренок и их задач расширился настолько, что пренебрегать

стандартными языками программирования уже неразумно. Представляется целесообразным рассмотреть возможность написания или адаптации других распространенных языков высокого уровня, таких как PASCAL, PL и особенно языка C. Наличие компиляторов с различных языков высокого уровня позволит со сравнительно небольшими затратами адаптировать к Одренку разнообразные программные средства с других типов ЭВМ. Существует также ошибочное мнение о несовместимости программ на языке ФОРТРАН-1900 и языке ФОРТРАН на других типах ЭВМ. Все компиляторы являются в той или иной мере машинно-зависимы. В этом смысле ФОРТРАН-1900 отходит от стандарта ФОРТРАН-4 не больше, чем компиляторы на других типах ЭВМ. Совместимость диалектов ФОРТРАНа можно проиллюстрировать двумя примерами. В 1986 г. была предпринята попытка перевести на язык TRAN игровую программу ADVENTURE из варианта для PDP-11 (авторы к этому не имеют отношения). Программа состояла из примерно 1—1.5 тысяч строк на языке ФОРТРАН и из нескольких подпрограмм на языке MACRO-11. После первого же дня работы выяснилось, что перевод на TRAN без радикальных изменений структуры программы невозможен сразу по нескольким причинам (например, переполнение таблиц переменных компилятора и т. п.). С другой стороны, когда текст программы был адаптирован к ФОРТРАНу-1900 и подпрограммы на MACRO-11 были переписаны на TRANe, программа заработала без достаточного понимания алгоритмов ее работы. Основную часть времени «переводчики» затратили на разбирательство функций MACRO-подпрограмм, на написание и отладку их.

Вторым примером, иллюстрирующим тот же тезис, может служить постановка пакета PIGS. В 1987 г. была предпринята попытка перевести его на TRAN. После нескольких недель преодолевания трудностей работа сама собой сошла на нет. Работа по адаптации пакета к языку ФОРТРАН-1900 и проверке работоспособности полученной версии заняла всего несколько дней. После этого мы могли заняться адаптацией пакета к тем устройствам, которые распространены на наших машинах, не тратя время на изучение устройства пакета, объем которого превышает 1000 строк.

Поскольку в Институте мало профессиональных программистов, представляется разумным не писать заново подобные программные средства, а создать на микроЭВМ более-менее стандартную среду, на которую можно переносить уже готовые программы

и подпрограммы с других типов ЭВМ (разумеется, если имеются исходные тексты).

В заключение следует отметить, что библиотеку для научных расчетов FSCE, предназначенную для использования на ФОРТРАНе, можно использовать также и в программах, написанных на TRANe. Для этого достаточно переместить из ФОРТРАНовской библиотеки SRF5 в LIBR подпрограммы, на которые есть ссылки в подпрограммах научной библиотеки.

Авторы считают своим приятным долгом выразить признательность Г.С. Пискунову и С.Д. Белову за полезные консультации по системе компиляции и стандартам фирмы ICL, а также Вечеславову В.В., убедившему авторов в необходимости внедрения ФОРТРАНа на микроЭВМ Одренок.

ЛИТЕРАТУРА

1. *Пискунов Г.С., Тарарышкин С.В.* Двадцатичетырехразрядная ЭВМ в стандарте КАМАК.—Автометрия, 1986, № 4, с.32—38.
2. *Алешаев А.Н. и др.* Построение распределенных систем управления крупными электрофизическими установками на базе сетей специализированных микроЭВМ в ИЯФ СО АН СССР и их программное обеспечение.—Автометрия, 1986, № 4, с.39—45.
3. *Мезенцев Н.А., Пиндюрин В.Ф.* Автоматизация экспериментов с использованием синхротронного излучения в ИЯФ СО АН СССР.—Препринт ИЯФ СО АН СССР 87-107. Новосибирск, 1987, 23 с
4. *Белов С.Д.* Система компиляции на управляющих машинах комплекса ВЭПП-4.—Работы молодых специалистов, выполненные в ИЯФ СО АН СССР в 1977-1978 годах.—Новосибирск, 1978.—(Отчет/ ИЯФ СО АН СССР).
5. Методические указания по программированию на ФОРТРАНе. Тюмень, 1974.

ПРИЛОЖЕНИЕ

ОПИСАНИЕ ПРОГРАММЫ #FORT

Программа #FORT предназначена для стыковки компилятора с языка ФОРТРАН-1900 #XFAP с существующей файловой и операционной системой. #FORT осуществляет обмен с диском, запуск #CNSL, вывод листинга на терминал, если это требуется, распознает некоторые директивы в тексте программы и осуществляет преобразования исходных строк.

Для компиляции ФОРТРАН-программ на диске необходимо иметь программы #FORT, #XFAP (можно в системной директории), библиотеку SRF5 и загрузчик #XP@@. Все остальное не является необходимым. #FORT принимает директивы с терминала либо из файла и распечатывает на терминале листинг и диагностические сообщения, если это необходимо. Листинг можно перенаправить на другой терминал или на принтер типа DZM-180 директивой операционной системе:

```
AL FORT 30 *13
```

последняя цифра (3) указывает номер терминального канала, куда перенаправляется весь вывод.

В тексте можно использовать восьмеричные константы, которые преобразуются препроцессором в десятичные перед передачей их компилятору.

Из стандартных подпрограмм можно использовать все, кроме тех, которые, используя литеральные константы, сами определяют ее длину. Например, из библиотеки ODLB не будет работать только сегмент DTEXT. Если подпрограмме не нужно определять длину константы, т. е. длина указывается в явном виде или в тексте существует символ-терминатор, понятный подпрограмме, то подпрограммой можно пользоваться.

Примеры:

```
CALL DRUS(1,4HFIL0,LF,IC)
CALL OUTXM(-5,5H↑*A1=)
CALL VIDA(2,'_ –/')
CALL INTXM(L,'_DIR_TEX_LIST_ _')
```

Управляющие знаки образуются по тем же правилам, что и в TRANe.

Как ввод, так и вывод можно производить подпрограммами видеопакетов, соблюдая правила употребления литералов, описанные выше.

Примеры:

```
CALL TEX('ABCD') или
CALL TEXT('↑*X=')
```

в ФОРТРАНе работают неправильно,

```
CALL OUTXM(-4,'ABCD') или
CALL OUTX(-4,'↑*X=')
```

работают и в ФОРТРАНе и в TRANe.

#FORT распознает и исполняет некоторые директивы, описанные ниже.

R:NAME — директива, указывающая, что следует считывать исходный текст из файла с именем NAME. При достижении конца файла, #FORT начинает просматривать предыдущий файл, в котором встречалась такая же директива, а если предыдущих файлов нет (стек имен пуст), то #FORT переходит на прием строки с терминала.

LIBRARY(LIBR) — оператор, указывающий, что при сборке программы должна использоваться библиотека с именем LIBR. Таких операторов в тексте может встретиться не более трех. Кроме этого, #FORT всегда имитирует оператор LIBRARY(SRF5) — ссылка на библиотеку SRF5, которая содержит различные подпрограммы, используемые компилятором, поэтому эту библиотеку можно не указывать специально.

COMP(D) — компилировать строки, помеченные в поле комментария символом D.

COMM(D) — игнорировать строки, помеченные в поле комментария символом D.

Эти два оператора позволяют использовать в тексте отладочные строки. Каждая строка может быть помечена в первой колонке строки символом D. В режиме отладочной компиляции, включаемом директивой COMP(D), символ D устраняется препроцессором и строка компилируется обычным образом. В режиме обычной компиляции (по умолчанию или после директивы COMM(D)) этот символ заменяется на C и строка трактуется как комментарий. Этот механизм облегчает отладку программ.

Препроцессор распознает также строку с оператором LIST. Этот оператор передается компилятору, а #FORT в конце компиляции запускает консолидатор с соответствующей ячейки, чтобы в процессе составления протокола сборки на терминале распечатывался лидер программы.

Внешне #FORT работает аналогично #TRANy. После за-

грузки #FORT проверяет наличие файла SCRF, загружает компилятор #XFAP, проверяет его контрольную сумму и, если все хорошо, просит ввести строку текста. Так же, как и для #TRANa, строка из файла и строка с терминала для #FORTa не различаются, т. е. можно писать программу прямо с терминала, но можно и ввести имя файла:

```
R:NAME
```

Используются те же спецзнаки, что и в #TRANe: ⟨LF⟩ и ⟨ETX⟩ — нормальное завершение строки, ⟨DEL⟩ — аннулирование набранной строки и переввод ее. Вводимую строку можно редактировать, т. е. пользоваться курсорами, символами ⟨IC⟩ и ⟨DC⟩.

В публичной директории существует краткое описание языка ФОРТРАН-1900. Файл называется OFO0.

Ниже приведены примеры программ, написанных на ФОРТРАНе.

```
      LIBRARY(LIBK)
      LIBRARY(LIBR)
      LIST
      PROGRAM(AAAA)
      COMPRESS INTEGER AND LOGICAL
      PRIORITY 74          [ ПРИОРИТЕТ
      NOTRACE
      END
      MASTER AAAA          [ ИСПОЛЬЗОВАНИЕ ВИДЕОПАКЕТА
    1 CALL VIDA(2,4H_ _/)
      CALL OUTXM(-4,4H↑*X=)
      IC=INDIG(0,X)
      IF(IC-2)0,77,77
      CALL OUTXM(—6,6H X*X=)
      CALL OUDIG(0,X*X,7)
D     GO TO 1
   77 PAUSE
      END
      FINISH

      LIST
      PROGRAM(ABCD)
      PRIORITY 67
      COMPRESS INTEGER AND LOGICAL
      TRACE
```



```
      OUTPUT 9=CD0
      INPUT 1=TR0
      OUTPUT 2=TP0
      END
      MASTER AAAA
      READ(1,99)I1
   99 FORMAT(I7)
      WRITE(2,100),I1
  100 FORMAT(3HI1=,I7)
      PAUSE 1
      END
      FINISH
```

ОСОБЕННОСТИ КОМПИЛЯТОРА #XFAP НА МИКРОЭВМ ОДРЕНОК

Компилятор #XFAP предназначен для работы с перфолентой, перфокартами, принтером, магнитофоном и консолью оператора. Для работы с обычными терминалами, принтерами и барабаном были переписаны подпрограммы в библиотеке компилятора. В данной реализации компилятора для работы с описанными устройствами задействованы следующие каналы:

— устройства TP и TR (перфолента) трактуются как терминал;
— устройства CP и CR (перфокарты) трактуются как барабан;
— устройство LP трактуется как принтер;
— устройство CD трактуется как консоль оператора (или терминал, вывод на который производится в специфической форме);
— устройство MT не реализовано.

Работа с терминалом

Для работы с терминалом нужно описать канал ввода/вывода в описателе программы одним из следующих способов:

```
INPUT 1=TP0
OUTPUT 2=TP3
USE 7=TR2
```

TPi и TRj эквивалентны и никак не различаются, работать можно с обоими устройствами, при этом программа будет работать с тем терминалом, к которому она приписана. Вывод на терминал не имеет каких-либо особенностей. Ввод с терминала осуществляется с помощью подпрограммы ввода текста с редактиро-

ванием INVTED. Соответственно, при вводе можно пользоваться клавишами редактирования: ⟨курсор влево⟩, ⟨курсор вправо⟩, ⟨IC⟩, ⟨DC⟩. Терминатор ввода можно изменить или ввести несколько терминаторов с помощью подпрограммы VIDA по стандартным правилам:

CALL VIDA(2,'↑#↑—')

объявляет два терминатора ⟨ETX⟩ и ⟨RET⟩. Если в числе терминаторов объявлен символ ⟨курсор вниз⟩, то при вводе этого символа, программа завершает свою работу и уничтожает себя в памяти ЭВМ.

По умолчанию ввод завершается клавишей ⟨ETX⟩.

При пользовании терминальными подпрограммами из библиотеки LIBK (INVTED, LINED, INVT, VIDA...) следует помнить, что они имеют общий набор терминаторов со стандартными операторами ввода/вывода, так как он определяется одной и той же подпрограммой.

Работа с барабаном

При работе с барабаном файлы считаются последовательными. Для работы с файлами нужно описать канал ввода/вывода в описателе программы следующим образом:

ОПЕРАТОР K=Xn(NAMECOMMENT,L)

Здесь в качестве оператора могут употребляться INPUT (для ввода), OUTPUT (для вывода), USE (для ввода и вывода).
K — номер логического канала, X — тип устройства (можно употреблять CP и CR, эти устройства в подпрограммах не различаются), n — номер устройства, при работе с файлами от 0 до 7 (см. ниже), NAME — 4-символьное имя файла, которому будет ставится в соответствие указанный логический номер.

COMMENT — комментарий (два первые слова в файле). Комментарий может содержать либо 4, либо 8 символов.

L — ключ записи. Если ключ отсутствует, то запись в файл будет производиться с начала. Если присутствует ключ APPE, то файл будет дописываться.

Комментарий и ключ записи могут опускаться.

Примеры:

INPUT 1=CR0(NAME)
описание файла NAME, который разрешено только читать.

OUTPUT 4=CP1(OPISTEXT)
описание файла OPIS, в который можно только писать. Первое слово файла будет содержать в символьном виде TEXT.

USE 5=CR3(TFORTEXTFORT)
описание файла TFOR, в который разрешена и запись и чтение. Первые два слова файла будут содержать в символьном виде TEXTFORT (комментарий файла).

USE 6=CP4(TFORTEXTFORT,APPE)
описание файла для записи/чтения. Запись будет производиться только после конца имеющегося текста (файл будет дописываться). Если файл не содержит текстовой информации (например, был создан программой #ZJAA), то при обращении к нему с ключом APPE будет зарегистрирована ошибка чтения.

Работа с файлами реализуется посредством стандартного пакета DRUM, поэтому одновременно можно работать не более, чем с восемью файлами. Номер файла (для пакета DRUM) определяется как номер устройства (CP_i, i — номер файла). Если номер равен 0, то файл переоткрывается на каждую операцию записи/чтения. Это резко замедляет обмен с барабаном, однако вероятность сбоев при реорганизации файловой структуры довольно мала. Если номер устройства лежит в пределах 1—7, то файл открывается только при первом обращении к нему и потом не переоткрывается. Это ускоряет работу, но в процессе работы такой программы запрещается реорганизация файловой системы (процедура CLEAR).

Кроме стандартных диагностических сообщений, программа может остановиться с дополнительными сообщениями:

NO NAME — на барабане отсутствует файл NAME;

END NAME — при записи в файл был достигнут конец файла;

DR — при чтении или записи произошел сбой.

В данной версии компилятора работают следующие операторы ФОРТРАНа:

REWIND K — перевести указатель строк логического номера K на первую строку.

BACKSPACE K — перевести указатель строк логического номера K на предыдущую строку.

ENDFILE K — записать признак конца текста в файл после последней прочитанной строки (при записи признак конца текста ставится автоматически).

Часто требуется работать с файлом, имя которого вводится с терминала. Существует подпрограмма, позволяющая изменить имя файла для любого канала. Обращение к подпрограмме:

CALL NEWNAME(K,NAME),

где K — номер устройства (канал), а NAME — имя файла, представляющее собой последовательность символов, которая может включать в себя слова комментария, а завершаться должна символом _.

Примеры:

```
CALL NEWNAME(0,'ABCD_')
CALL NEWNAME(0,'ABCDTEXTFORT_')
```

Работа с принтером

Из программы можно работать с принтерами различного типа, подключенными к одному из терминальных каналов. Вывод может производиться либо на принтер, либо на терминал. Для работы с принтером нужно описать канал вывода в описателе программы следующим образом:

OUTPUT K=LPn(NAME)

K — номер логического канала, n — номер терминального канала (1—7), NAME — в символьном виде имя принтера.

Принтеры, подключаемые к микроЭВМ Одренок, имеют различную кодировку (КОИ-7, КОИ-8, ASCII и другие, как например у D100) и управляются семью или восемью битами. Для согласования с принтерами различного типа при определении соответствующего канала необходимо указать имя принтера. Имя принтера может быть произвольной длины, но анализируются только первые 4 символа. Подпрограммы различают следующие типы принтеров:

MERA — терминал, или DZM-180, или D180.
DZM180 — терминал, или DZM-180, или D180.
D100 — принтер типа D100.
ROBOTRON — принтер типа ROBOTRON CM 6329.02.

Пример:

OUTPUT 7=LP3(D100)

вывод на принтер типа D100, подключенный к терминальному каналу номер 3.

Работа с консолью оператора

Из программы можно выводить информацию на консоль оператора. Для микроЭВМ Одренок консоль оператора совпадает с терминалом приписки, однако вывод происходит в другой форме:

#NAME DISPLAY:-TEXT

Всегда распечатывается стандартное сообщение, включающее имя программы, а затем до нескольких десятков символов выводимого текста (ограничение определяется операционной системой). Этот способ вывода реализован, так как он обладает некоторыми привилегиями.

КРАТКОЕ ОПИСАНИЕ ЯЗЫКА FORTRAN-1900

Настоящее описание не претендует на полноту и подробность. Для этого существуют соответствующие руководства, к сожалению немногочисленные. Ниже приводятся краткие сведения справочного характера для пользователей, которые когда-то работали на каком-либо ФОРТРАНе или TRANe, т. е. для подготовленных пользователей. Кроме основных сведений, здесь приводятся справочные материалы: список библиотечных и стандартных функций, список ошибок, выдаваемых компилятором ,и возникающих при работе программы. Желающие разобраться в каких-либо тонкостях языка ФОРТРАН-1900 должны qбратиться к соответствующим руководствам [5].

Общие сведения

ИМЕНА переменных, массивов и функций могут содержать до 32 символов, первый из которых должен быть буквой.

СТРОКА имеет до 72 колонок, из них первые 5 колонок употребляются для меток, а 6-я колонка — позиция продолжения. Если в позиции продолжения стоит пробел или 0, считается, что это обычная строка; если там поместить какой-либо другой символ, то

эта строка считается продолжением предыдущей. Количество строк, рассматриваемых как одна, не может быть более 20 (строк-продолжения не более 19). Буква С в первой позиции строки указывает, что эта строка занята под комментарий и транслятором не анализируется.

ДАННЫЕ могут быть целыми, вещественными, вещественными с двойной точностью, комплексными и логическими. Если переменная не описана, то она будет трактоваться как целая, если ее имя начинается с букв I, J, K, L, M, N; она будет трактоваться вещественной в остальных случаях.

ТИП переменной определяется, кроме описанного правила, оператором описания типа INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL, за которым следует список.

Величина типа INTEGER занимает одно машинное слово и может принимать целое значение в пределах ± 8388607. Формы записи: 29 99999 +144 −1760. Пример:

```
INTEGER S,IT,PR,TOM
```

Величина типа REAL занимает два машинных слова и может принимать значение в пределах $\pm 5.6 \cdot 10^{\pm 76}$ с точностью до 11 значащих цифр. Формы записи: 298. +37.6 −.33334 1234E7 +98.4E−16 Пример:

```
REAL ITEN, JOULES, ANSWER
```

Величина типа DOUBLE PRECISION может принимать значение в пределах $\pm 5.6 \cdot 10^{\pm 76}$ с точностью до 20 значащих цифр. Формы записи: 0.5D−56 76D6 .31D+3 1.02030405060708D−15 Пример:

```
DOUBLE PRECISION PI, VAR1, VAR2, MEAN
```

Величина типа COMPLEX состоит из двух вещественных чисел, соответствующих действительной и мнимой частям комплексного числа. Комплексные константы записываются в виде: (3.4, 2.) (−4.0, 2.34). Они соответствуют комплексным числам $(3.4+2I)$ и $(-4+2.34I)$. Пример:

```
COMPLEX A, B, C
```

Величина типа LOGICAL может принимать одно из значений: .TRUE. или .FALSE. Пример:

```
LOGICAL A, B, C
```

В отличие от ПЕРЕМЕННЫХ, КОНСТАНТЫ могут быть шести типов (шестой тип — текстовые константы). Они записываются

в следующем виде: 4HABCD 8HTEST-012. В текстовых константах пробелы не игнорируются. Текстовые константы могут употребляться либо в качестве параметра подпрограммы, либо в операторе DATA.

Программа может иметь МАССИВЫ размерностью до 32. Размерность массива определяется оператором DIMENSION. Пример:

```
DIMENSION A(15), ARR(5,5,10)
```

Этим оператором описывается массив из 15 ячеек и трехмерный массив с 250 ячейками. DIMENSION описывает только размерность массива, но не описывает его тип, поэтому, как правило, ему должно предшествовать описание типа. Возможные варианты описания 5·10 массива A:

```
      INTEGER A
      DIMENSION A(10,5)
или   INTEGER A(10,5)
или   INTEGER A
      COMMON /AT/A(10,5)
```

Массив хранится «по столбцам» в последовательных ячейках памяти, так что самый левый индекс меняется наиболее быстро, а последующие индексы менее быстро. Например, 3×3×3 массив A хранится следующим образом:

```
A(1,1,1), A(2,1,1), A(3,1,1), A(1,2,1),
A(2,2,1), A(3,2,1), A(1,3,1), A(2,3,1),
A(3,3,1), A(1,1,2) ..... A(3,3,3).
```

COMMON-блоки в ФОРТРАНе аналогичны COMMON-блокам в TRANe.

Примеры:

```
COMMON//X,ITEM,NAME/JOHN/A,B,C,...
COMMON/SET3/AR1,AR2,AR3(5,10)/SET4/LAST
```

Оператор EQUIVALENCE

Оператор EQUIVALENCE позволяет физически совместить две переменные, которые, имея различные имена, будут обращаться к одной и той же ячейке физической памяти. Формат:

```
EQUIVALENCE (k1),k2,...,(ki),
```

где каждая k — список переменных и элементов массива, которым присваивается транслятором одна и та же память. Список не должен включать фиктивных аргументов функций и подпрограмм.

Каждая переменная должна появляться не более чем в одном списке. Списки не должны приводить к попытке присвоить переменной более одной ячейки памяти. Ниже приведены примеры:
правильный— EQUIVALENCE (X(1),A).....(X(1),B)
неправильный— EQUIVALENCE (X(1),ARR(1)).....(X(1),ARR(3))

Оператор DATA

Оператор DATA позволяет определить начальные значения переменных и элементов массивов. Он имеет вид:

DATA R1/d1/, R2/d2/,...Rn/dn/

где каждая R—список переменных, элементов массивов и наименования массивов, а каждая d—соответствующий список начальных значений для них. Наименование массива эквивалентно списку всех элементов массива, записанных по порядку. Любой элемент информации в d-списке может следовать за «j*», где j—целая без знака, такой элемент списка эквивалентен повторению элемента информации j раз. Примеры:

DATA A,B(1),C/2*1.0,3.5/

присвоит первым двум элементам начальное значение 1.0, а элементу C—значение 3.5

DATA HEAD /8HANSWER =/
DATA ANDY /38.3/, TERRY,BOB /2*—3.9/
DATA ITEM(1),ITEM(2)/2*326/

Выражения

В FORTRAN-1900 существуют следующие АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ: +, —, *, /, ** (возведение в степень).

Порядок расчета:
обращения к функциям и скобочные выражения;
возведение в степень;
умножение и деление;
сложение и вычитание.

Внутри каждого из классов порядок выполнения операций ведется слева направо.

В выражении могут участвовать переменные разных типов. Ниже приведены таблицы, по которым можно определить тип результата в зависимости от типов аргументов.

Для операций +, —,*,/:

	INTEGER	REAL	DOUBLE PREC.	COMPLEX
INTEGER	INTEGER	REAL	DOUBLE PREC.	COMPLEX
REAL	REAL	REAL	DOUBLE PREC.	COMPLEX
DOUBLE PR.	DOUBLE PREC.	DOUBLE PREC.	DOUBLE PREC.	N
COMPLEX	COMPLEX	COMPLEX	N	COMPLEX

Для операций ** (возведения в степень):

	INTEGER	REAL	DOUBLE PREC.	COMPLEX
INTEGER	INTEGER	REAL	DOUBLE PREC.	N
REAL	REAL	REAL	DOUBLE PREC.	N
DOUBLE PR.	DOUBLE PREC.	DOUBLE PREC.	DOUBLE PREC.	N
COMPLEX	COMPLEX	N	N	N

Примечания:
N—запрещенная комбинация.
Нулевое основание не должно возводиться в нулевую степень.
Отрицательный элемент не должен возводиться в степень REAL или DOUBLE PRECISION.

ЛОГИЧЕСКИЕ ВЫРАЖЕНИЯ—это последовательность операторов и элементов типа LOGICAL. Его значение всегда либо .TRUE., либо .FALSE. Простейшей формой логического выражения является единственный элемент типа LOGICAL, который может быть константой, переменной, элементом массива или обращением к функции. Обычно применяются выражения:

E1 r E2,

где E1 и E2—арифметические выражения, а r—один из следующих операторов отношения:

Оператор Представление;
.LT. меньше чем;
.LE. меньше или равно;
.EQ. равно;
.NE. не равно;
.GT. больше чем;
.GE. больше или равно.

Выражение принимает значение .TRUE. или .FALSE., если оно истинно или ложно, соответственно. Допустимые комбинации E1 и E2 даются в таблице.

Для операций **:

	INTEGER	REAL	DOUBLE PREC.	COMPLEX
INTEGER	Y	Y	Y	N
REAL	Y	Y	Y	N
DOUBLE PR.	Y	Y	Y	N
COMPLEX	Y	N	N	N

Примечания:
N — запрещенная комбинация.
Y — разрешенная комбинация.

Кроме описанных операторов, существуют еще и следующие:

.NOT.A — имеет величину .TRUE. если A есть .FALSE. и наоборот.

A.AND.B — имеет величину .TRUE. если A и B одновременно являются .TRUE., а иначе результат — .FALSE.

A.OR.B — имеет величину .FALSE. если A и B одновременно являются .FALSE., а иначе результат — .TRUE.

Порядок вычисления:

выполнение операций во внутренних скобках;
операции типа .NOT.;
операции типа .AND.;
операции типа .OR.

Внутри каждого из классов порядок выполнения операций ведется слева направо.

Примеры:

```
A.LT.B+1.0.AND.(X.LT.0.01.OR.COUNT.EQ.0)
(AB.OR.AC).AND..NOT.(CHECK.OR.MISS)
```

АРИФМЕТИЧЕСКОЕ ПРИСВАИВАНИЕ имеет вид V=E. Если V и E имеют разный тип, то правила присваивания можно определить из таблицы:

Тип V	Тип V	Правило
INTEGER	INTEGER	Присваивание
INTEGER	REAL	Фиксация и присваивание
INTEGER	DOUBLE PRECISION	Фиксация и присваивание
INTEGER	COMPLEX	Запрещено
REAL	INTEGER	Переформировывание в REAL и присваивание
REAL	REAL	Присваивание
REAL	DOUBLE PRECISION	Выделение главной значащей части результирующего значения и присваивание

REAL	COMPLEX	Запрещено
DOUBLE PRECISION	INTEGER	Переформировывание в DOUBLE PRECISION и присваивание
DOUBLE PRECISION	REAL	Переформировывание в DOUBLE PRECISION и присваивание
DOUBLE PRECISION	DOUBLE PRECISION	Присваивание
DOUBLE PRECISION	COMPLEX	Запрещено
COMPLEX	INTEGER	Запрещено
COMPLEX	REAL	Запрещено
COMPLEX	DOUBLE PRECISION	Запрещено
COMPLEX	COMPLEX	Присваивание

ЛОГИЧЕСКОЕ ПРИСВАИВАНИЕ иллюстрируется примерами:

```
COURSE=.TRUE.
QUAD=B**2.GT.4*A*C
STATUS=STUDNT.AND.AGE.LT.21
```

МНОГОКРАТНЫМ ПРИСВАИВАНИЕМ можно присвоить одно и то же значение нескольким переменным:

```
SUM,SUMSQ,ITEM(N)=0
PARAM,START=(N+Q-1)/L
VOL(1),VOL(2),VOL(3)=986.1/LIM
```

Операторы

Безусловный оператор GO TO передает управление на оператор с меткой. Примеры:

```
GO TO 3
GO TO 9999
```

Вычисляемый GO TO передает управление на ту метку, номер которой в списке равен текущему значению INDEX. Нуль в списке меток означает передачу управления следующему оператору после GO TO. Пример:

```
GO TO (30,101,0,0,88), INDEX
```

Назначаемый GO TO работает совместно с оператором присваивания ASSIGN, который присваивает операторную метку INTEGER — переменной. После этого можно использовать назначаемый GO TO, который в зависимости от значения этой метки осуществляет переход. Ниже приводятся соответствующие примеры:

```
      ASSIGN 57 TO MEANS
      .....
      GO TO MEANS, (92,3,9999,57,3) — переход на метку 57.
```

или

```
      ASSIGN 57 TO MEANS
      .....
      GO TO MEANS — переход на метку 57.
```

Логический IF проверяет истинно или ложно логическое выражение. Если оно истинно, то будет выполняться оператор, справа от оператора IF. Примеры:

```
      IF(B*B.LT.4.0*A*C) GO TO 100
      IF(SU+TERM.GE.9E7.OR.TERM.LT.1E—2) CALL CHECK
      IF(A.LT.0.0) A=—100.0*A
```

Арифметический IF работает следующим образом: вычисляется выражение в скобках, если результат отрицательный, управление передается на левую метку, если равен нулю, — на среднюю метку, если положительный, — на правую метку. Примеры:

```
      IF(B*B—4.0*A*C)100,101,102
      IF(B*B—4.0*A*C)100,0,102
```

Оператор DO — оператор цикла. Он может иметь следующие формы:

```
      DO n I=IBEG,IEND,ISTEP
      DO n I=IBEG,IEND
```

Здесь n — это метка конечного оператора цикла, в программе она должна встречаться позже оператора DO. Конечный оператор цикла не может быть оператором GO TO, RETURN, STOP, PAUSE, DO, IF. Если это получается, то можно ввести пустой оператор CONTINUE.

Несколько операторов цикла могут заканчиваться на одной и той же метке. Внутри цикла можно использовать I, IBEG, IEND, ISTEP, но нельзя их изменять. После выполнения последнего оператора цикла, значение переменной I не определено. Шаг цикла (ISTEP) должен быть положителен. Пример:

```
      DO 3 I=1,100
      IF(A(I))4,3,3
    4 A(I)=A(I)+10.0
    3 CONTINUE
```

Оператор PAUSE может иметь следующие формы:


```
      PAUSE
      PAUSE 74
      PAUSE AB
```

В всех случаях программа останавливается с распечаткой #NAME HALTED, но во втором и третьем случаях печатаются еще и соответствующие символы, позволяющие понять на какую из PAUSE программа попала.

Оператор STOP аналогичен оператору PAUSE, но при его исполнении программа должна выбрасываться из памяти машины. В примитивных версиях компилятора этот оператор не работает.

Оператор CALL передает управление SUBROUTINE-сегменту и определяет действительные аргументы программы. Пример:

```
      CALL ITERATION(N1,15.0,RR)
```

Оператор RETURN возвращает управление из SUBROUTINE или FUNCTION-сегмента в вызывающий сегмент.

Структура программы

Каждая программа должна содержать один и только один сегмент MASTER. Он начинается с оператора MASTER m, где m — наименование сегмента. Последней строкой сегмента должен быть END. Наименование MASTER-сегмента не должно появляться в операторе CALL. Обращение к другим сегментам осуществляется оператором CALL.

Кроме MASTER-сегмента, программа может иметь и сегменты типа SUBROUTINE. Этот сегмент начинается с оператора SUBROUTINE

```
      SUBROUTINE NAME(A1,A2,A3,...Ai)
```

Далее идет последовательность каких-либо операторов и последней строкой является строка с END. Подпрограмма должна иметь хотя бы один оператор RETURN, который передает управление оператору следующему за CALL (которым было обращение к подпрограмме).

Внутри подпрограммы могут быть обращения к другим подпрограммам, но не допускаются обращения к ней самой ни в явном виде ни в косвенном (через обращения к другим подпрограммам).

Обмен параметрами с другими подпрограммами может быть

осуществлен как через список параметров, так и через общие COMMON-блоки.

ФУНКЦИИ-СЕГМЕНТы очень похожи на подпрограммы. Начинаются они с оператора:

```
                    FUNCTION NAME(A1,A2,A3,...Ai)
          INTEGER FUNCTION NAME(A1,A2,A3,...Ai)
             REAL FUNCTION NAME(A1,A2,A3,...Ai)
          COMPLEX FUNCTION NAME(A1,A2,A3,...Ai)
DOUBLE PRECISION FUNCTION NAME(A1,A2,A3,...Ai)
          LOGICAL FUNCTION NAME(A1,A2,A3,...Ai)
```

В остальном функция не отличается от подпрограммы, за исключением того, что результат она может возвращать через переменную, имя которой совпадает с именем функции:

```
K=MULT(2,7)
IZ=13*MULT(3,2)
```

Если тип функции описан, то в любом сегменте, который обращается к функции, его тоже нужно описывать.

Обмен параметрами с другими подпрограммами может быть осуществлен как чертз список параметров, так и через общие COMMON-блоки.

Порядок операторов в сегментах

Операторы во всех сегментах должны располагаться в следующем порядке:

DIMENSION и операторы описания типа;
COMMON-операторы;
DATA-операторы;
EQUIVALENCE-операторы;
определения функций операторов;
рабочие операторы.

Операторы FORMAT могут располагаться в любом месте сегмента.

BLOCK DATA-сегмент

Этот сегмент не содержит выполняемых операторов и предназначен для присвоения начальных значений в COMMON-блоках посредством операторов DATA.

Операторы в сегменте BLOCK DATA должны располагаться в следующем порядке:

DIMENSION и операторы описания типа;

COMMON-операторы;
EQUIVALENCE-операторы;
DATA-операторы;
END.

Пример:

```
      BLOCK DATA
      INTEGER BETA,GAMMA
      LOGICAL OMEGA
      DIMENSION KAPPA(10,50)
      COMMON/B1/BETA,ALFA,PI/B2/KAPPA,OMEGA,IOTA
      DATA PI/3.14159/,OMEGA/.TRUE./
      END
```

Ввод и вывод данных

Ввод данных осуществляется оператором READ, а вывод — оператором WRITE, которые взаимодействуют с оператором FORMAT, несущим информацию о способе представления данных.

Операторы READ, WRITE обычно содержат список элементов, величины которых передаются. Примеры вывода списка:

```
WRITE (2,11) A,I(J,K),Y(2*L-3),ARRAY
WRITE (1,10) MATRIX,X,B,AR(3,7,2),AR(3,7,3),AR(3,7,I+2)
WRITE (3,87) REL,INT,COMP,DUBL,LOGL
```

Элемент списка может быть DO-списком. Пример:

```
WRITE (1,10) (A(I),I=1,10)
```

должен иметь такой же эффект, как список ввода/вывода:

```
WRITE (1,10) A(1),A(2),A(3),...A(10)
```

DO-списки могут быть вложенными (т. е. любой элемент может быть тоже DO-списком) до любой глубины.

ФОРМАТНЫЙ WRITE может иметь одну из следующих форм:

```
WRITE(K,N) M
WRITE(K,N)
```

Здесь K — логический номер устройства; N — метка оператора FORMAT, в соответствии с которым будет вывод; M — список ввода/вывода. Список может отсутствовать при выводе пустых записей (содержащих только текст).

```
      WRITE (3,19)
19    FORMAT(/16HEXAMPLE OF TITLE/)
```

Возможна форма НЕФОРМАТНОГО WRITE:

```
      WRITE(K) M
```

при выполнении оператора величины элементов в списке M будут выводиться в том порядке, в котором они расположены в списке во внутренней машинной форме.

Оператор READ работает аналогично оператору WRITE только он предназначен для ввода, а не для вывода и описываться не будет.

Спецификация формата

Оператор FORMAT — это организационный оператор вида FORMAT(s), где s — форматная спецификация. Каждый оператор FORMAT должен быть помечен и может располагаться в любом месте сегмента. К одному оператору FORMAT могут обращаться несколько операторов типа WRITE и READ.

Спецификация формата — это ряд косых черточек, запятых и описателей полей, заключенных в скобки. Описатели полей имеют вид

```
s r F w d
s r E w d
s r G w d
s r D w d
  r I w
  r L w
  r A w
  w H h1h2h3h4....hi
  w X
```

Здесь заглавные буквы указывают на вид преобразования и называются кодами преобразования; w представляет ширину поля во внешней строке символов; d представляет число позиций для символов в дробной части числа (за исключением G-кода при выводе); s — фактор шкалы, а r — фактор повторения.

Передача целых значений осуществляется I-кодом, REAL-значений — E,F,G-кодами, DOUBLE PRECISION-значений — посредством D-кода, а LOGICAL-значений — посредством L-кода. Поскольку COMPLEX-значение можно рассматривать как два

REAL-значения, то передача COMPLEX-значений осуществляется E-, F-, G-кодами.

Информация в символьной форме выдается посредством A- или H-кодов. X-код определяет игнорируемые поля (вывод стольких пробелов).

Фактор шкалы имеет вид nP где n — целая константа, возможно со знаком «—» впереди. Он может появиться с F-, E-, G- или D-кодом. Фактор шкалы обычно используется, когда числа слишком велики или малы. Положительная или отрицательная величина n определяет степень 10, на которую число делится при вводе или умножается при выводе. Фактор шкалы воздействует на все последующие E-, F-, G-, D-описатели до тех пор, пока не встретится новый фактор шкалы.

Описатели поля и группы описателей поля должны отделяться запятой, одной или несколькими косыми черточками, которые могут появляться перед или после описателей поля. Запятая только разделяет поля, косая черточка тоже разделяет поля, а несколько косых черточек, кроме этого, вызывает перевод строки на (n — 1) строку:

```
      FORMAT(I4,I7/E7.2///I5)
```

вывод двух целых чисел, одного вещественного, две пустые строки и вывод целого числа.

Скобки в начале и в конце спецификации дают особый случай:

```
      FORMAT(///I7///)
```

вывод трех пустых строк, целого числа и еще трех пустых строк.

Любой элемент может включать перед собой фактор повторения, например, 3I5 соответствует записи I5,I5,I5. Фактор повторения может относиться к группе полей:

```
      FORMAT(I3,3I5,6(/E7.2,E9.3),2E9.2)
      FORMAT(I4,L6,10(10X,3L12,I4)/2I5)
```

При обработке форматного списка, он просматривается слева направо. Если список ввода/вывода короче списка форматного, то выводится только то, что нужно. Если вы пытаетесь вывести больше элементов, чем нужно, то список формата просматривается от самой правой открывающей скобки.

Любой форматный оператор ввода/вывода может вместо FORMAT-метки содержать наименование массива, в котором должна быть записана форматная спецификация либо посредством записи

по А-формату, либо занесена как текстовая константа. Информация в массиве не должна содержать Н-описателей полей.

Описатели полей

I-поле предназначено для ввода/вывода целого числа и записывается как I7, I5, I2, ... Цифра определяет количество знаковых позиций для вывода. Все числа выравниваются по правому краю, т. е. первые незначащие позиции будут заполнены пробелами. Если число не влезает в отведенное поле, то перед ним будет печататься звездочка и само число окажется несколько смещенным вправо.

E-поле предназначено для ввода/вывода REAL-значения или одной из компонент COMPLEX-значения и записывается как E14.5, E7.3 3PE14.5 Первая цифра определяет количество знаковых позиций для вывода, а вторая — количество цифр в дробной части этого числа. При выводе дробная часть округляется. Выводится число в следующем виде:

Для E14.5	+12345678	----›	0.12346E 08
	−1.23	----›	−0.12300E 01
	+0.000123	----›	0.12300E−03
	−0.03	----›	−0.30000E−02
Для 3PE14.5	+1234.5678	----›	123.457E 01
	−1.23	----›	−123.000E−02
	−0.03	----›	−300.000E−05

F-поле предназначено для ввода/вывода REAL-значения или одной из компонент COMPLEX-значения и записывается как F14.5, F7.3 3PF14.5 Первая цифра определяет количество знаковых позиций для вывода, а вторая — количество цифр в дробной части этого числа. При выводе дробная часть округляется. Выводится число в следующем виде:

Для F10.4	+5227.3278	----›	5227.3278
	−345.6789	----›	−345.6789
	+12.3	----›	12.3000
	−3.21989623	----›	−3.2199
Для 3PF14.4	+5227.3278	----›	5227327.8000
	+12.3	----›	12300.0000

G-поле предназначено для ввода/вывода REAL-значения или одной из компонент COMPLEX-значения и записывается как G14.5, G7.3 3PG14.5 Работает это поле следующим образом: если

можно вывести в F-формате, то оно в нем и выводится, иначе оно выводится в E-формате.

D-поле предназначено для ввода/вывода DOUBLE PRECISION и записывается как G14.5, G7.3 3PG14.5 Работает это поле как F-формат.

L-поле предназначено для ввода/вывода LOGICAL переменных и записывается как L7, L2, L12, ... При выводе выводится сколько-то пробелов и в последней позиции буква T или F, как значения .TRUE. или .FALSE.

A-поле предназначено для ввода/вывода символьной информации и записывается как A7, A2, A12, ... Цифра указывает число символов в выводимом поле.

X-поле предназначено для вывода нескольких пробелов и записывается как X7, X2, X12, ... Цифра указывает число выводимых пробелов.

H-поле предназначено для вывода символьной информации, хранимой в форматной спецификации и имеет вид:

iHh1h2h3h4....hi

где i-количество выводимых символов.

Примеры:

```
11HTHE TIME IS
WRITE (3,20)IDATE
20 FORMAT(21HMATRIX MULTIPLICATION,I6)
```

Внутри программы операторы READ и WRITE обращаются к устройствам по их логическому номеру, который должен быть предварительно определен. Это делается следующим образом:

```
INPUT 1=TR0
```

определяет, что под номером 1 будет при вводе использоваться устройство TR0.

```
OUTPUT 2=LP2
```

определяет, что под номером 2 будет при вводе использоваться устройство LP2.

Логические номера могут быть в пределах от 0 до 4095. Одно устройство может иметь несколько логических номеров.

В ФОРТРАН-1900 допускается осуществлять ввод и вывод информации в массив операторами ввода/вывода аналогично операциям с устройствами. Для этого необходимо определить канал ввода/вывода как работающий с массивом, а в процессе исполнения программы нужно связать данный канал с определенным мас-

сивом. Это делается соответствующими операциями:

USE k=/ARRAY

определяет канал k как работающий с массивом

CALL DEFBUF(k,n,arrayname)

связывает данный канал k с массивом с именем arrayname, а n — длина массива в символах.

Программист может подавить ввод/вывод по какому либо каналу следующим способом:

INPUT 7=/NONE
OUTPUT 2=/NONE

Средства отладки

В ФОРТРАН-1900 существует мощное отладочное средство — так называемая трассировка, имеющая три уровня. Управление трассировкой производится включением в описатель программы одного из следующих операторов:

NOTRACE или
TRACE 0

включается нулевой уровень трассировки. В этом случае, если в процессе исполнения программы будет зарегистрирована ошибка, на устройство вывода будет выведено сообщение об ошибке, а программа будет остановлена либо уничтожена в памяти ЭВМ.

TRACE 1

включается первый уровень трассировки. В этом случае, если в процессе исполнения программы будет зарегистрирована ошибка, то управление может быть передано подпрограмме пользователя, если использовать вспомогательную подпрограмму FTRAP.

TRACE 2

включается второй уровень трассировки. В этом случае, если в процессе исполнения программы будет зарегистрирована ошибка, то на устройство вывода будет распечатано сообщение об ошибке, а также предыстория процесса в виде:

—4 901 ARTH 0.56227 E76

что означает следующее:

—4 — это четвертая строка программы перед последней выполненной;

901 — оператор помечен меткой 901 в исходном тексте;

ARTH — исполнялся арифметический оператор;

0.56227 E76 — это значение присваивалось переменной оператором.

Операторы ФОРТРАНа и их обозначение в списке трассировки приведены ниже.

Арифметическое присвоение	ARTH
Логическое присвоение	LOGC
Арифметический IF	IF
Логический IF	LIF
GOTO	GO
DO	DO — в начале цикла LOOP — в конце цикла
Вычисляемый GOTO	CGO
Назначаемый GOTO	AGO
READ	READ
WRITE	WRTE
PAUSE	PAUS
STOP	STOP
CALL	CALL
RETURN	RETN
CONTINUE	CONT
ENDFILE	ENDF
BACKSPACE	BACK
REWIND	REW
ASSIGN	ASGN

Информация трассировки выводится на первый канал, описанный как OUTPUT. Рекомендуется использовать для вывода консоль терминала, т. е. в программе первый канал ввода/вывода должен определяться строкой:

OUTPUT 1=CD0 или
OUTPUT 9=CD0

Стандартные внутренние функции

Функция и определение	Кол-во аргум.	Наименование	Тип аргумента	Тип функции
Абсолютное значение |a|	1	ABS	REAL	REAL
	1	IABS	INTEGER	INTEGER
	1	DABS	DOUBLE	DOUBLE
Выделение целой части	1	AINT	REAL	REAL
(знак, умноженный на				
наибольшее целое	1	INT	REAL	INTEGER
меньше или равно |a|	1	IDINT	DOUBLE	INTEGER
Ближайшее целое (знак,				
умноженный на	1	NINT	REAL	INTEGER
ближайшее целое [0.5=1])	1	ANINT	REAL	REAL
Остаток от деления	2	AMOD	REAL	REAL
X=MOD(a1,a2) [=a1-{a1/a2}*a2]	2	MOD	INTEGER	INTEGER
Выбор наибольшего значения	>1	AMAX0	INTEGER	REAL
	>1	AMAX1	REAL	REAL
	>1	MAX0	INTEGER	INTEGER
	>1	MAX1	REAL	INTEGER
	>1	DMAX1	DOUBLE	DOUBLE
Выбор наименьшего значения	>1	AMIN0	INTEGER	REAL
	>1	AMIN1	REAL	REAL
	>1	MIN0	INTEGER	INTEGER
	>1	MIN1	REAL	INTEGER
	>1	DMIN1	DOUBLE	DOUBLE
Преобразование INTEGER				
в REAL	1	FLOAT	INTEGER	REAL
Преобразование REAL				
в INTEGER	1	IFIX	REAL	INTEGER
Присвоение знака a2				
к модулю a1	2	SIGN	REAL	REAL
	2	ISIGN	INTEGER	INTEGER
	2	DSIGN	DOUBLE	DOUBLE
Положительная разность	2	DIM	REAL	REAL
	2	IDIM	INTEGER	INTEGER
Выделение REAL из DOUBLE	1	SNGL	DOUBLE	REAL
Выделение реальной части из комплексного числа	1	REAL	COMPLEX	REAL

Выделение мнимой части из комплексного числа	1	AIMAG	COMPLEX	REAL
Формирование REAL в DOUBLE	1	DBLE	REAL	DOUBLE
Формирование COMPLEX из действительной и мнимой частей	2	CMPLX	REAL	COMPLEX
Получение сопряженного комплексного числа	1	CONJG	COMPLEX	COMPLEX

Основные библиотечные функции

Функция	Определение	Кол-во аргум.	Наименование	Тип аргумента	Тип функции
Экспонента	e(a)	1	EXP	REAL	REAL
		1	DEXP	DOUBLE	DOUBLE
		1	CEXP	COMPLEX	COMPLEX
10 в степени a	10^a	1	EXP10	REAL	REAL
Логарифм	LOG e(a)	1	ALOG	REAL	REAL
натуральный		1	DLOG	DOUBLE	DOUBLE
		1	CLOG	COMPLEX	COMPLEX
Логарифм	LOG10(a)	1	ALOG10	REAL	REAL
десятичный		1	DLOG10	DOUBLE	DOUBLE
Логарифм двоичный	LOG2 (a)	1	ALOG2	REAL	REAL
Синус в радиан.	SIN(a)	1	SIN	REAL	REAL
натуральный		1	DSIN	DOUBLE	DOUBLE
		1	CSIN	COMPLEX	COMPLEX
Косинус в радиан.	COS(a)	1	COS	REAL	REAL
натуральный		1	DCOS	DOUBLE	DOUBLE
		1	CCOS	COMPLEX	COMPLEX
Тангенс в радиан.	TAN(a)	1	TAN	REAL	REAL
Котангенс в рад.	COT(a)	1	COT	REAL	REAL
Гипербол. синус	SINH(a)	1	SINH	REAL	REAL
Гипербол. косинус	COSH(a)	1	COSH	REAL	REAL
Гипербол. тангенс	TANH(a)	1	TANH	REAL	REAL
Гипер. котангенс	COTH(a)	1	COTH	REAL	REAL
Квадрат. корень	$\sqrt{a}$	1	SQRT	REAL	REAL
		1	DSQRT	DOUBLE	DOUBLE
		1	CSQRT	COMPLEX	COMPLEX

Арксинус	ASIN(a)	1	ASIN	REAL	REAL
Арккосинус	ACOS(a)	1	ACOS	REAL	REAL
Арктангенс	ATAN(a)	1	ATAN	REAL	REAL
		1	DATAN	DOUBLE	DOUBLE
	ATAN(a1/a2)	1	ATAN2	REAL	REAL
		1	DATAN2	DOUBLE	DOUBLE
Арккотангенс	ACOT(a)	1	ACOT	REAL	REAL
Обратный гипербол. синус		1	ASINH	REAL	REAL
Обратный гипербол. косинус		1	ACOSH	REAL	REAL
Обратный гипербол. тангенс		1	ATANH	REAL	REAL
Обратный гипербол. котангенс		1	ACOTH	REAL	REAL
Остаток от деления a1(mod a2)		2	DMOD	DOUBLE	DOUBLE
Модуль	|a|	1	CABS	COMPLEX	REAL

Ошибки при трансляции

1. Неправильное использование скобки.
2. Не хватает скобок.
3. Опущен один из знаков /,)
4. Неправильный вид выражения.
5. Недопустимое значение постоянной.
6. Использована переменная не вида INTEGER.
7. Массив не описан или описан дважды.
8. Первый знак названия не буква или название опущено.
9. Слишком длинный оператор.
10. Оператор записан не полностью.
11. Метка использована дважды или ошибки в колонках с 1 по 5.
12. Переход к несоответствующей метке, например, переход к метке оператора FORMAT.
13. Неверен список индексов массива (например, слишком много или слишком мало) или наименование массива использовано как переменная без индексов.
14. Неправильное наименование, например, подпрограмма имеет неправильное наименование.
15. Ошибка в метке, использованной в операторе DO.
16. Ошибка в метке.
17. Нераспознанный оператор.
18. Ошибка при трансляции в строке управляющего сегмента.
19. Выражение слишком сложное или слишком много сегментов, включенных в программу (более 150 сегментов).
20. Ошибка транслятора (не должна появляться).
21. Левая часть арифметического оператора присваивания является выражением или вместо оператора отношения .EQ. использован знак =.
22. Не описан массив или описание функции-оператора появилось в несоответствующем месте.

23-25. Не использованы.

Значение следующих ошибок зависит от типа строки, в которой они встретились.

Ошибка 26

CALL — вызываемая подпрограмма имеет такое же наименова-

ние, как и сегмент, содержащий оператор CALL.

FUNCTION — ошибка в наименовании сегмента.

ОБРАЩЕНИЕ К ФУНКЦИИ — функция имеет то же наименование, что и MASTER-сегмент или сегмент, в котором эта функция используется.

SUBROUTINE — ошибка в наименовании сегмента.

GOTO — неправильный вид оператора GOTO.

DO — оператор DO записан не полностью.

IF — оператор DO использован в логическом IF.

FORMAT — ошибка в метке.

READ или WRITE — программный номер устройства не является ни константой, ни переменной типа INTEGER.

INTEGER, REAL, LOGICAL, DOUBLE PRECISION, COMPLEX — наименование, приведенное в списке, уже имеет присвоенный тип или оператор типа появляется в несоответствующем месте. COMMON — неправильное наименование общего блока.

DIMENSION — ошибка в размерности массива.

EQUIVALENCE — в списке участвует формальный параметр.

DATA — несоответствующее наименование в списке, например, наименование подпрограммы.

Ошибка 27

FUNCTION — неверен список параметров после наименования функции.

ОБРАЩЕНИЕ К ФУНКЦИИ — параметр функции имеет такое же наименование, как и MASTER-сегмент или сегмент, в котором используется данная функция.

GOTO — в операторе перехода с переключателем ошибка после закрывающей скобки.

CALL — вызываемая подпрограмма имеет такое же наименование, как и сегмент, содержащий оператор CALL.

READ или WRITE — ссылка на оператор FORMAT не является ни меткой, ни наименованием массива.

COMMON — недопустимый элемент списка, например, формальный параметр.

EQUIVALENCE — недопустимый элемент списка, например, константа или наименование сегмента.

DATA — неправильный вид константы.

Ошибка 28

DO — параметры не являются выражениями типа INTEGER.

READ, WRITE — ошибка в списке.

EQUIVALENCE — массив, приведенный в списке, не описан.

DATA — неправильный вид константы типа COMPLEX.

ОШИБКА 29

EQUIVALENCE — противоречивый список.

DATA — слишком много или слишком мало констант.

35. Программа требует память более 32K или полей объема более 4K.
36. Неполная программа: пропущен сегмент, используемый в программе, или пропущен сегмент описания программы, или сегмент описания не является первым вводимым сегментом.

Ошибки при выполнении программы

0. При вводе числа вместо цифрового знака встречен запрещенный знак.
1. Ошибка в индексе массива. Адрес массива выходит за пределы, отведенные для этого массива. Появляется в случае TRACE 2.
2. Список управления слежением (TRACE) имеет неправильную форму.
3. Превышена глубина наложения (OVERLAY).
5. Менее двух параметров задано для библиотечной функции, определяющей минимальное или максимальное значение числа.
7. Недостаточное количество параметров в обращении к подпрограмме.
8. Слишком много параметров в вызванном сегменте, написанном на языке PLAN.
10. Неправильная спецификация формата, например, опущен разделитель поля, знак помещен не на месте, встретился нераспознанный символ.
11. Код преобразования E применен к переменной типа INTEGER, DOUBLE PRECISION, LOGICAL.
12. Код преобразования I применен к переменной типа REAL, DOUBLE PRECISION, LOGICAL, COMPLEX.
14. Код преобразования L применен к переменной типа INTEGER,

DOUBLE PRECISION, REAL.

15. Код преобразования F применен к переменной типа INTEGER, LOGICAL. Код преобразования D применен к переменной типа INTEGER, REAL, COMPLEX, LOGICAL.
16. Ширина поля в коде преобразования A, L, H или X не указана или равна 0.
17. Данные типа LOGICAL не начинаются знаками T, F, .T или .F
18. Директива FORMAT не соответствует списку ввода/вывода.
20. Программный номер устройства, используемый в программе, не описан в управляющих строках.
21. Форматный вводной рекорд слишком короткий.
22. Неформатный вводной рекорд слишком короткий.
23. Запрещенная операция при работе с внешними устройствами, например, чтение с перфоратора и т.п.
31. Отсутствие имени файла.
32. Некорректный формат входного файла.
34. При чтении был достигнут конец файла.
36. Попытка прочитать очень длинную форматную запись.
37. Попытка прочитать очень длинную запись как форматную.
47. Параметр подпрограммы FERROR превосходит допустимый предел.
50. Переполнение во время выполнения арифметических операций (в режиме TRACE 2).
53. Результат функции EXP выходит за допустимые пределы.
54. Результат функции DEXP выходит за допустимые пределы.
55. Неположительное значение параметра функции ALOG.
56. Неположительное значение параметра функции DALOG.
57. Отрицательный параметр функции SQRT.
58. Отрицательный параметр функции DSQRT.
59. Нулевые параметры функции ATAN2.
60. Нулевые параметры функции DATAN2.
61. Нулевой второй параметр функции ISIGN.
62. Нулевой второй параметр функции SIGN.
63. Нулевой второй параметр функции DSIGN.
64. Нулевой второй параметр функции MOD.
65. Нулевой второй параметр функции AMOD.
66. Нулевой второй параметр функции DMOD.
67. Значение параметра функции ACOSH не удовлетворяет условию $X < 1$.
68. Значение параметра функции ATANH не удовлетворяет условию $-1 < X < 1$.

69. Значение параметра функции ACOTH не удовлетворяет условию $ABS(X) > 1$.
70. Значение параметра функции ASIN не удовлетворяет условию ABS(X).LE.1
71. Значение параметра функции ACOS не удовлетворяет условию ABS(X).LE.1
72. Результат функции TAN выходит за допустимые пределы.
73. Результат функции CABS выходит за допустимые пределы.
74. Результат функции CSIN выходит за допустимые пределы.
75. Результат функции CCOS выходит за допустимые пределы.
76. Значение параметра функции NINT выходит из допустимых пределов.
77. Результат функции IFIX выходит за допустимые пределы.
78. Результат функции INT выходит за допустимые пределы.
79. Глубина обращения к подпрограммам превышает емкость списка TRACE.
80. Ошибочный блок информации, используемый подпрограммой GENAH.
100. Программа дошла до оператора END.

В.Н. Гончаров, В.Р. Козак, А.А. Никифоров

Программное обеспечение для микроЭВМ Одренок.
Фортран-1900

Ответственный за выпуск С.Г.Попов

Работа поступила 23 мая 1988 г.
Подписано в печать 31.05. 1988 г. МН 08366
Формат бумаги 60×90 1/16 Объем 2,9 печ.л., 2,4 уч.-изд.л.
Тираж 290 экз. Бесплатно. Заказ № 81

Набрано в автоматизированной системе на базе фотонаборного автомата ФА1000 и ЭВМ «Электроника» и отпечатано на ротапринте Института ядерной физики СО АН СССР,
Новосибирск, 630090, пр. академика Лаврентьева, 11.