

**БИБЛИОТЕЧКА
ПРОГРАММИСТА**

**А.И. САЛТЫКОВ
Г.И. МАКАРЕНКО**

Программирование на языке ФОРТРАН



БИБЛИОТЕЧКА
ПРОГРАММИСТА

А. И. САЛТЫКОВ, Г. И. МАКАРЕНКО

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ФОРТРАН

Под редакцией
Н. Н. ГОВОРУНА



ИЗДАТЕЛЬСТВО «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
Москва 1976

Программирование на языке фортран.
А. И. Салтыков, Г. И. Мака-
ренко. Главная редакция физико-мате-
матической литературы издательства
«Наука», М., 1976.

В книге излагаются основы програм-
мирования на алгоритмическом языке фор-
тран и автокоде madlen для БЭСМ-6.
Изложение ведется применительно к кон-
кретной версии языка фортран, принятой
в мониторной системе «Дубна». Приво-
дятся необходимые сведения о машине
БЭСМ-6 и ее математическом обеспечении.
Специальная глава посвящена вопросам
оптимизации программ.

Книга рассчитана на широкий круг
программистов — вычислителей, исполь-
зующих БЭСМ-6; она может также служить
учебным пособием для студентов вузов
и техникумов,

ОГЛАВЛЕНИЕ

Предисловие редактора	6
От авторов	8
Г л а в а I. Машина БЭСМ-6 и ее математическое обеспечение	11
§ 1. Краткая характеристика БЭСМ-6	12
§ 2. Представление команд и чисел	14
§ 3. Структура памяти. Буферные регистры	16
§ 4. Некоторые особенности системы команд БЭСМ-6	18
§ 5. Арифметические операции и особенности их выполнения	21
§ 6. Основные экстракоды	23
§ 7. Операционная система «Дубна» и ее составные части	25
§ 8. Пакет задачи пользователя	29
Г л а в а II. Язык фортран в системе «Дубна»	40
§ 9. Фортран как язык программирования	40
§ 10. Запись программы на фортране. Операторы фортрана	41
§ 11. Типы величин, используемых в фортране	46
§ 12. Запись констант на фортране	49
§ 13. Переменные величины	52
§ 14. Стандартные математические функции	54
§ 15. Арифметические операции и правила их выполнения, Арифметические выражения	55
§ 16. Логические выражения и выражения отношения	61
§ 17. Оператор присваивания	65
§ 18. Операторы условного перехода. Операторы GO TO и CONTINUE	68
1. Арифметический оператор IF (71). 2. Оператор перехода GO TO (72). 3. Оператор CONTINUE (72). 4. Логический оператор IF (72).	
§ 19. Операторы перехода: вычисляемый GO TO и GO TO по предписанию. Оператор ASSIGN	74
§ 20. Оператор DO	78
§ 21. Массивы переменных. Операторы описания типа	86
§ 22. Подпрограмма-функция	94

§ 23. Функции-операторы	100
§ 24. Подпрограмма (SUBROUTINE)	102
§ 25. Общие блоки. Эквивалентности. Данные	105
1. Оператор COMMON (105). 2. Оператор EQUIVALENCE (110). 3. Оператор DATA (113).	
§ 26. Формальные и фактические параметры. Оператор EXTERNAL	118
§ 27. Структура фортранной программы. Операторы ENTRY и RETURN	124
§ 28. Операторы ввода-вывода	127
1. Оператор вывода PRINT (127). 2. Оператор ввода READ (131). 3. Оператор PUNCH (131). 4. Операторы обмена информацией с внешними запоминающими устройствами (131). 5. Бесформатный оператор вывода (134). 6. Форматный оператор вывода (134). 7. Бесформатный оператор ввода (135). 8. Форматный оператор ввода (135). 9. Операторы управления магнитной лентой (135).	
§ 29. Оператор FORMAT	136
1. Спецификации преобразования при выводе (139). 2. Спецификации преобразования при вводе (144). 3. Редакционные спецификации (148). 4. Новая единица записи (150). 5. Масштабный коэффициент (151). 6. Повторяемые спецификации (152). 7. Управление расположением строк при печати (153). 8. Взаимодействие управления форматом со списком оператора ввода-вывода (154). 9. Переменный оператор FORMAT (157).	
§ 30. Операторы ENCODE И DECODE	159
§ 31. Диагностика ошибок	161
1. Диагностика при декодировке пакета задачи (162). 2. Диагностика при трансляции с фортрана на автокод (163). 3. Диагностика при трансляции с автокода на язык загрузки (166). 4. Диагностика при загрузке подпрограмм в память машины (168). 5. Диагностика при счете (169). 6. Диагностика по операторам FORMAT (173). 7. Ошибка по выходу на карту *END FILE (174).	
§ 32. Советы и рекомендации	174
1. О некоторых особенностях фортрана (174). 2. Об использовании целых величин (175). 3. О некоторых способах устранения переполнений и машинных нулей (175). 4. Об использовании величин с двойной точностью (176). 5. О блокировке авоста (177). 6. Об управлении загрузкой подпрограмм (177). 7. Определение времени счета (178). 8. Об одном способе «редактирования» пакета (178).	
§ 33. Примеры и упражнения	179
Ответы. Указания. Решения	185

Г л а в а III. Описание автокода madlen	193
§ 34. Операторы автокода	194
§ 35. Идентификаторы	195
§ 36. Мнемокоды	196
§ 37. Полный адрес	198
§ 38. Указатель индекс-регистра	199
§ 39. Метки	199
§ 40. Константы	200
§ 41. Адреса типа «литерал»	204
§ 42. Описания	206
§ 43. Параметрические команды	214
§ 44. Данные и рассылки	215
§ 45. Комментарии	216
§ 46. Правила оформления автокодных подпрограмм	217
§ 47. Базирование	221
§ 48. Примеры автокодных подпрограмм	225
§ 49. Стандартный массив	226
§ 50. Диагностика ошибок	228
§ 51. Управляющие карты, редактирование, сервис	229
§ 52. Советы и рекомендации	230
§ 53. Некоторые приемы программирования на автокоде	232
Г л а в а IV. Оптимизация фортранных подпрограмм	235
§ 54. Оптимизирующие возможности транслятора с фортрана	235
§ 55. Оптимизация путем изменения способа адресации величин	237
§ 56. Оптимизация индексных выражений	238
§ 57. Оптимизация с помощью автокода	239
§ 58. Способы экономии памяти. Сегментация задачи	241
Приложения	247
1. Система команд БЭСМ-6 (247). 1А. Экстракоды (249). 2. Символы ISO (250). 3. Символы GOST (АЦПУ-128) (251). 4. Символы TEXT (252). 5. Символы TEL (252).	
Литература	253

ПРЕДИСЛОВИЕ РЕДАКТОРА

Вычислительная машина БЭСМ-6 получила за последние годы достаточно широкое распространение. Круг пользователей этой машины составляет десятки тысяч человек. Пользователей привлекают как высокие технические параметры БЭСМ-6, так и развитое математическое обеспечение.

В последние годы значительную популярность в нашей стране получил алгоритмический язык фортран. Относительная простота и легкость использования этого языка, развитые средства организации ввода-вывода и обмена информацией, наличие возможности управления распределением памяти, богатая библиотека стандартных программ, доступная программам на этом языке, а также простая возможность написания в программе некоторых частей на машинно-ориентированном языке (автокоде) делают язык фортран особенно удобным при создании больших систем обработки экспериментальных данных и для проведения научно-технических расчетов.

Большой интерес к БЭСМ-6 и к алгоритмическому языку фортран вызывает большой спрос на литературу по этим вопросам. Настоящая книга вместе с готовящейся к выпуску книгой В. Я. Карпова «Алгоритмический язык фортран» в известной мере восполнит пробел в литературе по программированию на БЭСМ-6 и предоставит широким кругам пользователей информацию по языку фортран и особенностям его применения на машине БЭСМ-6.

В книге авторы приводят краткие сведения о технических характеристиках ЭВМ БЭСМ-6 и о системе ее команд. Дается информация о мониторной системе «Дубна» и правилах оформления пакета задачи пользо-

вателя. Основная часть книги относится к описанию языка фортран, особенностям программирования на этом языке и вопросам оптимизации программ в мониторной системе «Дубна». В небольшой главе дается описание автокода madlen, входящего в мониторную систему. Эта глава предназначена для системных программистов и для тех читателей, которые уже имеют опыт программирования и стремятся максимально оптимизировать свои программы.

Изложение материала в книге имеет четко выраженную практическую направленность и ведется с учетом особенностей программирования в мониторной системе «Дубна». В настоящее время на БЭСМ-6 в основном используются только две операционные системы ДИСПАК и «Дубна». Мониторная система «Дубна» (трансляторы с языка фортран, автокод madlen, загрузчик и др.) входит в состав обеих систем (ДИСПАК и «Дубна»), и поэтому основное содержание книги будет полезно практически всем пользователям БЭСМ-6.

Приемы программирования, рассмотренные в книге на конкретных примерах, а также задачи и упражнения помогут читателю приобрести навыки, необходимые для решения вычислительных задач на машине. В книге уделено много внимания вопросам поиска ошибок в программах на основании информации об этих ошибках (диагностики), выдаваемой системой математического обеспечения.

Книга будет интересна и полезна как широкому кругу пользователей машины БЭСМ-6, так и читателям, интересующимся вопросами программирования на языке фортран.

Н. Н. Говорун

ОТ АВТОРОВ

БЭСМ-6 относится к вычислительным машинам, получившим широкое распространение за последние 10 лет. Эта машина уже в первые годы эксплуатации была оснащена развитым математическим обеспечением [6], [14], [16], [18], [21], [25].

Для программирования на БЭСМ-6 пользователю в первую очередь необходима информация о средствах описания алгоритмов решения вычислительных задач. Сюда входят сведения о языках программирования, а также о библиотечных программах, предназначенных для решения типовых вычислительных задач. Пользователю необходима также некоторая информация об организации вычислительного процесса на машине и о сервисе, предоставляемом системой математического обеспечения. Желательно также, чтобы он имел некоторое представление о самой машине.

В предлагаемой книге содержатся некоторые основные сведения, необходимые для программирования на БЭСМ-6 в системе математического обеспечения «Дубна».

Книга состоит из четырех глав. В первой, вводной главе, даются краткие сведения о машине БЭСМ-6 и ее математическом обеспечении. Вторая, основная глава, содержит достаточно подробное описание версии языка фортран, используемой в системе «Дубна». В третьей главе описывается машинно-ориентированный язык программирования — автокод madlen. Четвертая, заключительная глава, содержит описание некоторых практических приемов оптимизации программ.

При изложении материала мы руководствовались двумя основными принципами. Во-первых, стремились дать читателю как можно больше сведений практи-

ческого характера, которые могут пригодиться при составлении программ и отладке их на машине. Во-вторых, мы старались излагать материал по возможности доступно и просто, ориентируясь на читателя с небольшим опытом программирования.

Основной материал книги изложен в главе II, описывающей версию языка фортран, известную под названием фортран-Дубна. На русском языке издан ряд книг, описывающих те или иные версии фортрана (например, [15] и [30]). Однако при программировании на конкретной машине необходимо знать и особенности той версии фортрана, которая является входным языком транслятора, эксплуатируемого на этой машине. Долгое время почти единственным описанием версии фортран-Дубна было пособие [41], впервые изданное в 1969 г. малым тиражом.

Автокод madlen широко используется при составлении системных программ, в которых необходимо полностью учесть особенности машины БЭСМ-6. Знание этого языка может пригодиться квалифицированным пользователям, стремящимся максимально оптимизировать свои программы.

Несмотря на возросшую скорость работы ЭВМ и увеличение объема их оперативной памяти, вопросы рационального использования ресурсов машины продолжают оставаться достаточно актуальными. Поэтому мы включили в книгу небольшую главу, посвященную рассмотрению некоторых практических приемов экономии машинного времени и места в оперативной памяти. Среди многочисленных работ по этому вопросу отметим [23].

Мы рекомендуем читателю при первом знакомстве с книгой после прочтения Введения и § 1 главы I сразу же переходить к чтению главы II или главы III в зависимости от того, какой язык программирования его интересует. Изложение материала этих глав построено так, что в большинстве необходимых случаев даются ссылки на соответствующие параграфы главы I.

Книга предназначена в основном для читателей, программирующих на БЭСМ-6. Она может быть использована в качестве учебного пособия для студентов вузов, а также быть полезной при самостоятельном изучении программирования.

Мы признательны чл.-корр. АН СССР Н. Н. Говоруну, согласившемуся быть ее титульным редактором.

В процессе работы над книгой мы постоянно пользовались консультациями участников работы по созданию системы «Дубна» А. И. Волкова, Н. С. Заикина, Л. Г. Каминского, И. Н. Силина и В. П. Широкова.

Текст книги был прочитан Г. Л. Мазным, оказавшим нам большую помощь своими советами. Отдельные главы книги были прочитаны А. В. Ракитским, сделавшим ряд критических замечаний и предложений по улучшению изложения материала.

Большую помощь при редактировании отдельных параграфов нам оказали И. Н. Мишенева, А. К. Полянский и А. П. Стельмах.

Всем названным товарищам выражаем нашу глубокую благодарность.

Мы будем благодарны читателям, которые пришлют нам свои замечания и пожелания.

Дубна, 1976

Авторы

ГЛАВА I

МАШИНА БЭСМ-6 И ЕЕ МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

Введение

Вычислительная машина БЭСМ-6 принадлежит к числу серийно выпускаемых ЭВМ большой производительности. Она оснащена развитым математическим обеспечением. Пользователь БЭСМ-6 имеет возможность программировать на языках высокого уровня (фортран, алгол и др.) и на машинно-ориентированных языках нижнего уровня (автокод). При этом отдельные части задачи могут быть написаны на различных языках, используемых в системе математического обеспечения.

В данной главе даются краткие сведения о машине БЭСМ-6 и ее математическом обеспечении. Изложение ведется применительно к системе математического обеспечения «Дубна» [27], которая в дальнейшем часто будет называться *мониторная система* или просто *система*. Иногда вместо термина «мониторная система» используют термин «система программирования».

В книге будут встречаться числа, записанные в десятичной и восьмеричной системе. Для отличия десятичных чисел от восьмеричных у последних будет ставиться буква В справа. Например, 17В означает восьмеричное число 17. В качестве разделителя целой и дробной части числа будет использоваться точка, а не запятая. Для обозначения пробелов в тех случаях, когда это необходимо, будет использоваться символ — (крытце).

§ 1. Краткая характеристика БЭСМ-6

БЭСМ-6 является универсальной вычислительной машиной с быстродействием около 1 млн. одноадресных операций в секунду. Основные ее параметры таковы.

Цикл оперативной памяти — 2 *мксек*, среднее время выполнения сложения с плавающей запятой — 1.2 *мксек*, умножения — 2.0 *мксек*, деления — 5.5 *мксек*, логических операций — 0.5 *мксек* (см. приложение 1).

Длина машинного слова — 48 двоичных разрядов. Диапазон представимых чисел (по абсолютной величине) от 10^{-19} до 10^{19} . Точность представления чисел — 12 десятичных знаков.

Объем оперативной памяти от 32К*) до 128К (в большинстве машин используется два сегмента памяти по 32К каждый). Объем промежуточной внешней памяти на магнитных барабанах до 512К. Внешние накопители информации включают в себя также до 32 магнитофонов. Возможно также подключение накопителей на съемных магнитных дисках.

Обмен информацией с магнитными барабанами (МБ), магнитными лентами (МЛ) и магнитными дисками (МД) производится массивами длиной в 1К. Каждый такой массив соответствует одной странице (или листу) оперативной памяти, одному тракту МБ, одной зоне МЛ или двум дорожкам МД. Для МБ допускается обмен массивами длиной 256 слов, каждый из которых соответствует одному абзацу оперативной памяти и занимает один сектор МБ. При записи массивов во внешнюю память к каждому машинному слову приформировываются дополнительные контрольные разряды (на разных устройствах по-разному). При считывании информации из внешней памяти производится проверка контрольных разрядов и контрольной суммы.

Ввод и вывод информации производится с помощью алфавитно-цифровых печатающих устройств (АЦПУ-128), устройств вывода на перфокарты (ПЭМ-80) и перфоленту, устройств ввода с перфокарт (ВУ-700 или УВвК-601) и с перфоленты (ФС-1501). К БЭСМ-6 могут быть подключены также телетайпы (электри-

*) 1К составляет 1024 машинных слова,

ческие пишущие машинки типа «Консул»), дисплей и графопостроители.

Подготовка перфокарт может производиться на устройствах с поколонной кодировкой и надпечаткой (типа ЕС-9015, АРИТМА-130, ICL-72 и т. п.) либо на устройствах типа УПП, а также на телетайпах или электрических пишущих машинках, подсоединенных к устройствам подготовки данных для машин типа БЭСМ-4.

Подготовка 5-дорожечной перфоленты может производиться на стандартных телетайпах (аналогичных телетайпам, используемым в отделениях связи). Подготовка 8-дорожечной перфоленты производится на специальных телетайпах.

Программы пользователя и данные к ним, вместе взятые, будем называть *задачей пользователя* или просто *задачей*.

Управление работой БЭСМ-6 во время счета задачи пользователя осуществляется специальной управляющей программой-диспетчером. С целью повышения эффективности использования устройств БЭСМ-6 ее работа организована в мультипрограммном режиме. В оперативной памяти может одновременно находиться до 3 задач пользователей и до 4 служебных задач. Каждой задаче диспетчер выделяет определенные ресурсы машины (оперативную память, внешние устройства и т. п.) и определенный лимит машинного времени.

До выхода на счет задача должна пройти определенный этап предварительной обработки. Сюда входит ввод информации с перфокарт (или других носителей) в машину, преобразование этой информации в машинное представление, трансляция отдельных подпрограмм задачи, подготовка печати на бумажной ленте АЦПУ (листинге) текстов транслируемых подпрограмм и т. д.

Управление процессом обработки задачи до выхода ее на счет производится мониторной системой (см. § 7). Задача пользователя обслуживается мониторной системой на основании информации, указанной в управляющих картах (см. § 8). Взаимодействие задачи с диспетчером в процессе счета производится через экстракоды (см. § 6).

Отметим, что работа диспетчера и некоторых блоков мониторной системы производится в так называемом *диспетчерском режиме*, в отличие от режима выполнения задачи пользователя, называемого *математическим режимом*. В математическом режиме работа производится с использованием оперативной памяти, выделенной для данной задачи. Такую память называют *математической оперативной памятью*. В математическом режиме нельзя использовать некоторые команды БЭСМ-6, выполняемые только в диспетчерском режиме.

§ 2. Представление команд и чисел

Машинное слово БЭСМ-6 состоит из 48 *двоичных разрядов* (или битов), нумеруемых справа налево числами от 1 до 48. Кроме того, имеется два дополнительных *контрольных разряда*, формируемых при записи информации в память. Эти разряды недоступны программе пользователя.

В каждом машинном слове можно расположить две команды или одно число.

Команды и числа функционально отличаются тем, что первые считываются из памяти в *управляющее устройство* (УУ), а вторые — в *арифметическое устройство* (АУ). Отметим, что способ учета контрольных разрядов при считывании из памяти команд и чисел различен. Это может сказаться при попытке использования числа как команды. В этом случае фиксируется ошибка, и задача снимается со счета (см. § 31). Запись информации в память с контрольными разрядами, удовлетворяющими правилам контроля команд, обычно производится посредством экстракода записи команд (см. § 6).

При размещении в слове двух команд разряды $48 \div 25$ соответствуют левой команде, а разряды $24 \div 1$ — правой команде:

левая команда	правая команда
48	25 24 1

Адрес слова соответствует адресу его левой команды,

т. е. при передаче управления слову с определенным адресом происходит переход на его левую команду. Распределение разрядов каждой половины слова для размещения отдельных частей команды (см. [22]) при программировании на автокоде нет необходимости учитывать. Отметим, что каждая команда содержит указатель индекс-регистра, код операции и адресную часть.

При размещении в слове числа (или кода) разряды слова рассматриваются либо как равноправные, либо как определенные функциональные части числа с плавающей запятой. Последний способ интерпретации разрядов слова имеет место лишь при выполнении арифметических операций (см. § 5).

Если машинное слово рассматривается как число с плавающей запятой, то отдельные его разряды распределяются следующим образом:



Положительная мантисса представляется в прямом коде с двоичным нулем в разряде знака мантиссы. Отрицательная мантисса представляется в дополнительном коде с двоичной единицей в разряде знака.

Положительный порядок числа представляется прямым кодом с 1 в разряде знака порядка. Отрицательный порядок представляется дополнительным кодом с 0 в разряде знака. Иначе говоря, порядок n представляется в семи старших разрядах слова как $100B + n$. Такой способ представления порядка используется во многих ЭВМ.

Значение мантиссы заключено в пределах от 0 до $1-2^{-40}$. Положительная мантисса называется нормализованной, если ее значение больше или равно 0.5. Отрицательная нормализованная мантисса, рассматриваемая в прямом коде, имеет значение, большее 0.5.

Значение числа с мантиссой M и порядком P равно $2^P \cdot M$. Поскольку значение порядка заключено между -64 и $+63$, то для нормализованных чисел, отличных от нуля, получаем диапазон возможных значений (по модулю) от 2^{-65} до $2^{63} \cdot (1 - 2^{-40})$ или, округленно, от 10^{-19} до 10^{+19} . Для ненормализованных чисел нижний предел диапазона равен 2^{-104} . Число, меньшее (по модулю) минимально допустимого значения, представляется нулем (так называемым машинным нулем).

Чаще всего используются числа, представленные в нормализованном виде. Целые числа (соответствующие, например, целым константам фортрана) представляются в ненормализованном виде с порядком 40 и с запятой, фиксированной после младшего разряда мантиссы. Отметим, что на БЭСМ-6 отсутствует полный набор операций над целыми числами.

Длина мантиссы машинного слова определяет точность машинного представления чисел. На БЭСМ-6 относительная погрешность представления для нормализованных чисел не превосходит 2^{-40} или, округленно, 10^{-12} .

§ 3. Структура памяти. Буферные регистры

Оперативная память БЭСМ-6 выполнена в виде отдельных сегментов объемом по 32К каждый. В зависимости от комплектации (или, как говорят, конфигурации) машины ее оперативная память может включать в себя от 1 до 4 таких сегментов. Наибольшее распространение получила конфигурация БЭСМ-6 с двумя сегментами памяти. Каждый сегмент памяти состоит из 32 страниц (или листов) одинакового размера.

Для обозначения оперативной памяти часто используют термин МОЗУ (магнитное оперативное запоминающее устройство).

Каждой задаче выделяется определенное число страниц МОЗУ, но не более 32. Память, выделенная задаче, имеет свою (математическую) нумерацию адресов. Соответствие между физическими и математическими номерами страниц памяти определяется с помощью специальной таблицы [24], формируемой диспетчером (так называемой *таблицы приписки*).

Рассмотрим некоторые особенности организации обмена информацией между МОЗУ БЭСМ-6 и ее арифметическим и управляющим устройствами подробно описанными в работе [24].

Цикл оперативной памяти БЭСМ-6, т. е. время считывания из памяти (или записи в память) одного машинного слова, равен 2 *мксек*. Это значительно превышает время выполнения команд в УУ и время выполнения многих команд в АУ (см. приложение 1). Однако, благодаря параллельной работе нескольких блоков МОЗУ в режиме считывания, среднее время ожидания информации из памяти значительно сокращается.

В БЭСМ-6 имеются так называемые *буферные* (или быстрые) регистры. Эти регистры являются промежуточной сверхбыстрой памятью и предназначены для хранения небольшого количества чисел и команд с целью ускорения обмена информацией между памятью и арифметическим и управляющим устройствами.

Команды, подлежащие выполнению, заблаговременно считываются из памяти и помещаются в буферные регистры (или стек) команд. Емкость стека — 4 машинных слова (8 команд). Если при повторном использовании команды выясняется, что она находится в стеке, то обращения за ней в МОЗУ не происходит. Такая ситуация возникает, например, тогда, когда в программе имеется цикл, состоящий не более чем из 6 команд. В этом случае к моменту выполнения последней команды цикла первая его команда еще не успевает покинуть стек, так что выполнение цикла значительно ускоряется.

При записи слова в МОЗУ оно сначала попадает в один из восьми буферных регистров записи (БРЗ). Если происходит обращение к памяти для считывания слова из ячейки по определенному адресу и запрашиваемое слово находится на БРЗ, то считывание его производится из БРЗ за более короткое время, чем из МОЗУ. При записи слова с новым адресом в БРЗ происходит вытеснение оттуда (в МОЗУ) того машинного слова, которое используется наиболее редко. Поэтому слова с адресами, наиболее часто используемыми в программе, могут продолжительное время находиться на БРЗ.

§ 4. Некоторые особенности системы команд БЭСМ-6

При рассмотрении команд [22] мы будем иметь в виду лишь те из них (см. приложение 1), которые выполняются в математическом режиме. Команды мы будем обозначать их мнемосодами на автокоде madlen (см. § 36).

Рассмотрим сначала основные регистры БЭСМ-6, участвующие в выполнении команд.

Большинство команд БЭСМ-6 оперирует с содержимым 48-разрядного *сумматора*, который будет обозначаться через А. Состояние сумматора изменяется после выполнения арифметических, логических и ряда других операций. В зависимости от текущего состояния сумматора может изменяться порядок выполнения последующих команд.

Продолжением сумматора является 48-разрядный *регистр младших разрядов* У, в который помещается часть результата при выполнении ряда арифметических операций, а также операций сдвига и некоторых других операций. Для использования информации из регистра У надо передать его содержимое на сумматор по команде УТА. Отметим, что многие команды гасят содержимое этого регистра.

Регистр режима и блокировок (или регистр признаков), обозначаемый через R, состоит из 6 разрядов. Содержимое каждого его разряда (при нумерации их справа налево) определяет следующие признаки:

- 1 — блокировка нормализации,
- 2 — блокировка округления,
- 3 — признак логической группы,
- 4 — признак группы умножения,
- 5 — признак группы сложения,
- 6 — блокировка прерывания по переполнению.

Код 1 в данном разряде регистра R означает установку соответствующего признака, а код 0 — его отмену. Например, содержимое R, равное 23В, означает установку признаков блокировки нормализации, блокировки округления и группы сложения и отмену остальных признаков.

Состояние признаков групп (разряды 3÷5) может изменяться многими командами. При этом каждая из команд (кроме NTR и XTR) либо устанавливает один

из признаков группы (отменяя два остальных), либо сохраняет текущее состояние всех трех признаков группы. Команды с мнемокодами NTR и XTR изменяют состояние всех шести разрядов регистра R.

Текущее состояние признаков группы вместе с текущим состоянием сумматора определяют условия передачи управления по командам UZA и U1A (или UIA). Эти условия выглядят следующим образом:

1) установлен признак логической группы, и на сумматоре находится ненулевой код, т. е. код, содержащий 1 хотя бы в одном из его разрядов;

2) установлен признак группы умножения, и 48-й разряд сумматора равен 0 (что соответствует отрицательному знаку порядка);

3) установлен признак группы сложения, и 41-й разряд сумматора равен 1 (что соответствует отрицательному знаку мантиссы).

При выполнении любого из этих условий по команде UZA происходит переход на следующую команду, а по команде U1A (UIA) — на левую команду машинного слова, указанного по ее исполнительному адресу. При нарушении любого из перечисленных условий команда UZA передает управление левой команде слова по своему исполнительному адресу, а команда U1A (UIA) — следующей команде.

Регистр изменения команд C, состоящий из 15 разрядов, используется для образования исполнительного адреса команды. Состояние регистра C устанавливается командами UTC и WTC, остальные команды гасят признак необходимости его использования.

В формировании исполнительного адреса большинства команд может участвовать и один из 15 индекс-регистров, которые называются также *модификаторами* и имеют номера от 1 до 15. Все индекс-регистры 15-разрядные. Содержимое индекс-регистра может быть передано в 15 младших разрядов сумматора и, обратно, содержимое 15 младших разрядов сумматора можно передать в индекс-регистр.

Имеются команды условной передачи управления по нулевому (VZM) и ненулевому (V1M или VIM) состоянию модификатора, указанного в команде. В этом случае модификатор не участвует в формировании ее исполнительного адреса. Команда VLM позволяет

организовывать циклы на заданное число повторений, определяемое начальным состоянием модификатора, в ней указанного.

Выполнение команд производится по их исполнительному адресу. Исполнительный адрес получается путем сложения адресной части команды с содержимым регистра С и (для многих, но не для всех команд) с содержимым модификатора, указанного в команде. У результирующего адреса берутся 15 младших разрядов, которые и образуют исполнительный адрес. Отметим, что содержимое регистра С учитывается лишь при условии, что предыдущей командой была команда с мнемокодом UTC или WTC. Отметим также, что при выполнении некоторых команд (например, E + N, ASN) учитываются лишь 7 младших разрядов, а при выполнении команды NTR — 6 младших разрядов исполнительного адреса. При выполнении команды YTA исполнительный адрес рассматривается особо (см. [22]).

Многие команды (в том числе команды, выполняющие арифметические и логические операции) имеют 12-разрядную адресную часть. Такие команды, называемые командами с коротким адресом, позволяют адресовать (без модификации адресной части) лишь ячейки памяти с адресами от 00000В до 07777В и от 70000В до 77777В. Это может создать определенные неудобства при программировании. Однако применение автоматического базирования (см. § 47) позволяет устранить практически все трудности, связанные с использованием короткоадресных команд.

Индекс-регистр с номером 15 играет в БЭСМ-6 особую роль — он является счетчиком магазина. *Магазин* — это особым образом организованная область памяти. При считывании из магазина оттуда выбирается последнее записанное в него число. После считывания оно исключается из магазина. При записи числа в магазин оно попадает в «верхушку» магазина, проталкивая имеющиеся в нем числа на одну позицию в глубь магазина. Принцип работы магазина иногда поясняют фразой: «Последним пришел — первым ушел».

В системе «Дубна» под магазин отводится 377В ячеек памяти, начиная с адреса 53401В, 55401В или 73401В (в зависимости от объема памяти, выделенной

задаче). Начальный адрес магазина устанавливается мониторной системой.

Любая команда, требующая обращения к памяти, а также команда WTC может быть использована в режиме магазина. Для этого в команде должен быть указан 15-й индекс-регистр и нулевой исполнительный адрес (формируемый без участия 15-го индекс-регистра). Имеется несколько специальных команд (XTS, STX и др.), работающих исключительно в магазинном режиме.

В приложении 1 приведены основные характеристики команд БЭСМ-6. Для каждой команды указано состояние сумматора А, регистра младших разрядов Y и регистра признаков R после ее выполнения, а также время выполнения команды в УУ и в АУ. Время выполнения указано в машинных тактах. Длительность одного машинного такта равна 0.11 микросекунды.

§ 5. Арифметические операции и особенности их выполнения

К арифметическим операциям относятся операции, выполняемые командами с мнемокодами $A + X$, $A - X$, $X - A$, AMX , AVX , A/X , $A * X$, $E + X$, $E - X$, $E + N$, $E - N$ (см. приложение 1). Выполнение этих операций отличается от выполнения остальных операций рядом особенностей, а именно:

1. При выполнении арифметических операций машинное слово рассматривается как число с плавающей запятой.

2. Арифметические операции могут выполняться в одном из четырех режимов:

- 1) с нормализацией и с округлением,
- 2) с нормализацией и с блокировкой округления,
- 3) с блокировкой нормализации и с округлением,
- 4) с блокировкой нормализации и с блокировкой округления.

Чаще всего используется режим с нормализацией и с блокировкой округления, соответствующий 0 в первом и 1 во втором разряде регистра R. Этот режим является стандартным в системе «Дубна».

3. Арифметические операции могут выполняться как над нормализованными, так и над ненормализован-

ными числами. Исключением является операция деления, требующая нормализованного делителя. Нормализация и округление результата производится в зависимости от текущего состояния первого и второго разрядов регистра R.

4. При выполнении арифметических операций могут возникнуть внутренние прерывания (авосты), вызванные переполнением АУ или делением на нуль (либо на ненормализованное число). При этом управление передается диспетчеру, который снимает задачу со счета. Снятие задачи со счета по причине авоста можно заблокировать (см. § 32).

5. После выполнения операций $A+X$, $A-X$, $X-A$, AMX , $A \cdot X$ мантисса результата состоит из 80 разрядов, причем младшие 40 разрядов располагаются в регистре Y. Эти разряды могут быть выданы на сумматор по команде YTA.

Нормализация и округление результата производятся (если необходимо) после выполнения операции. При этом нормализация предшествует округлению. Исключение составляет операция деления, после которой округление результата не производится, так как он получается равновероятно с недостатком или с избытком. Если результат деления допускает точное машинное представление, то он получается точным (например, $9.0/6.0 = 1.5$).

Проверка АУ на переполнение производится после нормализации результата. Переполнение АУ означает, что порядок результата после его нормализации превосходит 63. Если порядок результата после нормализации окажется меньше -64 , то производится гашение всех разрядов результата, т. е. выдается так называемый машинный нуль.

Округление состоит в логическом сложении младшего разряда сумматора с единицей в следующих случаях:

1) если нормализация не нужна, а в младших сорока разрядах мантиссы результата есть хотя бы одна единица;

2) если до нормализации в сорока младших разрядах мантиссы была хотя бы одна единица, и при нормализации из младших разрядов в старшие не перешла ни одна единица.

§ 6. Основные экстракоды

Экстракоды позволяют задаче пользователя обращаться к диспетчеру для выполнения определенных действий. В частности, через экстракоды производится обмен информацией с внешними устройствами и вычисление основных элементарных функций.

Экстракоды имеют восьмеричные коды операций от 050 до 077 и адресные части, соответствующие командам с коротким адресом. На языке madlen экстракодам соответствуют, в частности, мнемокоды от *50 до *77 (см. приложение 1А).

После выполнения команды экстракода управление передается левой команде следующего машинного слова. Экстракоды «портят» содержимое 14-го индекс-регистра. Содержимое остальных индекс-регистров и регистра признаков сохраняется. В зависимости от характера экстракода его исполнительный адрес используется по-разному.

Экстракоды с мнемокодами от *50 до *57 и с нулевым исполнительным адресом производят вычисление элементарных функций \sqrt{x} , $\sin x$, $\cos x$, $\arctg x$, $\arcsin x$, $\ln x$, e^x и $[x]$ соответственно. Значение аргумента задается на сумматоре, результат также получается на сумматоре. Допускается ненормализованное значение аргумента. Результат всегда выдается в нормализованном виде.

В случае обращения к вычислению \sqrt{x} при $x < 0$, $\arcsin x$ при $|x| > 1$, $\ln x$ при $x \leq 0$ и e^x при $x \geq 63 \ln 2$ (≈ 43.5) происходит внутреннее прерывание (авост) с выдачей диагностики. При этом указывается название элементарной функции, вызвавшей прерывание.

Вычисление элементарных функций производится за время от 55 до 90 мксек, кроме функции $[x]$, вычисляемой за 25 мксек. Точность вычисления значений упомянутых функций для «хороших» значений аргумента — не менее 11 верных десятичных знаков (см. [34]).

Экстракод с мнемокодом *63 позволяет считывать на сумматор содержимое ячейки памяти по физическому адресу, соответствующему исполнительному адресу этого экстракода. Это бывает необходимо,

например, при работе системных программ, использующих информацию из диспетчерских таблиц.

Экстракод печати имеет мнемокод *64 или PRINT. По его исполнительному адресу задается информация, определяющая характер размещения материала на бумаге и форму печати (десятичные числа, текст и т. п.).

Заметим, что вместо непосредственного обращения к этому экстракоду значительно удобнее использовать системные программы, предназначенные для вывода информации на печать. К этим программам можно обращаться как из автокодных, так и из фортранных подпрограмм (см. § 52).

Отметим два особых случая использования экстракода печати. При работе печатающего устройства БЭСМ-6 в стандартном режиме после вывода каждых 66 строк автоматически происходит переход на новый лист бумаги. В ряде случаев (например, при выдаче графиков или гистограмм) бывает необходимо заблокировать такое «листование». Это делается обращением к экстракоду печати с нулевым исполнительным адресом. Режим блокировки листования действует до его отмены, которая производится обращением к экстракоду печати с исполнительным адресом 1.

Экстракод с мнемокодом *70 (а также TAPE и DRUM) предназначен для обмена информацией с внешними запоминающими устройствами. Обычно не возникает необходимости в его непосредственном использовании ввиду наличия большого числа системных программ (см. [27]), позволяющих удобно организовывать обмен информацией с внешними запоминающими устройствами.

Экстракод конца задачи имеет мнемокод *74 и SJ. Он предназначен для передачи управления диспетчеру по окончании задачи. Однако лучше использовать фортранный оператор STOP или автокодный оператор, CALL, STOP*. Это гарантирует дозапись материала, накопленного задачей, на внешние носители информации в случае, когда задача использует внешнюю память (МЛ, МД или МБ).

Экстракод с мнемокодами *75 и CTX предназначен для записи слова в память с контрольными разрядами команд. При этом записываемое слово фактически за-

носятся в МОЗУ, а не остается на БРЗ (см. § 3), как при обычной записи в память (например, по команде АТХ). Этот экстракод используется в основном в системных программах.

§ 7. Операционная система «Дубна» и ее составные части

Операционная система «Дубна» (ОС «Дубна») предназначена для организации вычислительного процесса на БЭСМ-6. Состав системы и ее возможности могут меняться в зависимости от конфигурации машины.

Основными частями ОС «Дубна» являются диспетчер (например, ДД-73) и мониторная система «Дубна». Диспетчер и мониторная система относительно автономны и допускают различные варианты совместного использования. Мониторная система «Дубна» эксплуатируется совместно со многими вариантами диспетчера и входит в состав не только ОС «Дубна», но и некоторых других операционных систем ([18], [20], [21]).

Мониторная система «Дубна» [14] создана в 1969 году коллективом, состоящим в основном из сотрудников Объединенного института ядерных исследований в Дубне, под руководством чл.-корр. АН СССР Н. Н. Говоруна и доктора физ.-мат. наук И. Н. Силина. В последующие годы происходило развитие и усовершенствование этой системы. Были созданы также новые варианты диспетчера, в том числе ДД-71, ДД-73 ([9], [36]) и другие.

Основными частями мониторной системы являются монитор, трансляторы с входных языков программирования, загрузчики программных модулей, редактор текстов и библиотекарь.

Монитор организует процесс обработки задачи мониторной системой. Анализируя информацию, содержащуюся в управляющих картах (см. § 8), монитор вызывает соответствующие системные программы для выполнения требуемых действий.

Важными составными частями мониторной системы являются трансляторы с языков программирования, используемых в этой системе. В системе «Дубна» все трансляторы имеют своим выходным языком так называемый язык загрузки.

Основными языками программирования в системе «Дубна» являются фортран, автокод madlen и алгол. В последнее время к этой системе подключены трансляторы с некоторых других алгоритмических языков, а также многоязыковая система программирования МУЛЬТИТРАНС [1].

Входным языком для транслятора с фортрана [4], созданного в 1969 г., служит версия фортран-Дубна, которая является расширением версии ЦЕРН-фортран. Версия фортран-Дубна близка к версии фортран-IV (см. [8]) и практически совместима с версиями фортрана, получившими распространение на многих отечественных и зарубежных ЭВМ.

Трансляция подпрограмм с фортрана осуществляется в два этапа. На первом этапе производится трансляция с фортрана на автокод. Затем вызывается транслятор с автокода, который выдает окончательный результат трансляции в виде программного модуля на языке загрузки, или стандартного массива.

Транслятор с фортрана обладает достаточно высокой скоростью работы — около 10—20 фортранных операторов в секунду. Качество транслированной программы зависит от состава операторов исходной программы. Трансляция арифметических выражений, не содержащих переменных с индексами, производится практически оптимально. В последнее время создан вариант транслятора, позволяющий в ряде случаев оптимизировать трансляцию выражений, содержащих переменные с индексами (см. [31], [32]).

Диагностика ошибок, выдаваемая транслятором, достаточно точно определяет характер каждой ошибки. Это позволяет легко их обнаруживать и исправлять даже пользователям с небольшим опытом программирования.

Транслятор с автокода [11], созданный А. И. Волковым, производит трансляцию с машинно-ориентированного языка madlen. При разработке этого языка был учтен опыт создания подобного рода языков и, в частности, языка SIBESM-6 (см. [17], [38]).

В системе Дубна автокод является самостоятельным языком нижнего уровня и одновременно промежуточным языком при трансляции подпрограмм с фортрана. В результате трансляции с автокода получается так

называемый стандартный массив, который преобразуется загрузчиком в машинное представление.

Объем транслятора с автокода — около 4000 слов. Скорость трансляции — около 200 автокодных операторов в секунду при работе в автономном режиме и вдвое больше при работе совместно с фортранным транслятором. Транслятор с автокода производит редактирование исходного текста подпрограммы, позволяя выдавать его на листинге в виде, удобном для чтения. После трансляции производится запись транслированной подпрограммы во временную библиотеку, создаваемую на время прохождения каждой задачи.

Транслятор с алгола создан в АН ГДР и подключен к мониторной системе в 1972 г. (см. [39]). Входной язык этого транслятора представляет собой подмножество алгола-60 с некоторыми расширениями. В частности, предусмотрена возможность выполнения операций над строками текстовых символов. Транслятор сразу выдает стандартный массив, минуя выход на автокод.

Преобразование стандартного массива в рабочую программу и размещение ее в оперативной памяти производятся загрузчиком. В системе «Дубна» имеется два загрузчика — динамический и статический. Чаще используется первый из них. Динамический загрузчик [35] производит сборку всей рабочей программы из отдельных подпрограмм. Каждая подпрограмма (на языке загрузки) должна быть либо во временной библиотеке, либо в одной из системных библиотек, либо в одной из личных библиотек. Загрузчик размещает эти подпрограммы в памяти машины для последующего счета. При каждой загрузке отдельной подпрограммы в память машины происходит настройка адресов ее команд по начальному адресу расположения этой подпрограммы в МОЗУ.

При работе статического загрузчика все подпрограммы должны быть предварительно преобразованы в их машинное представление и записаны в специальную библиотеку разделов. Загрузка каждого такого раздела производится на фиксированное место в памяти, определенное при формировании библиотеки разделов. Такой способ загрузки используется в основном в больших задачах, которые не могут целиком

поместиться в памяти и требуют разбиения на сменяющие друг друга разделы (см. § [27]).

Редактор текстов [13] служит для редактирования текстовой информации, расположенной во внешней памяти. Информация для работы редактора задается в виде управляющих карт, описывающих отдельные этапы редактирования, и карт, содержащих изменения, вносимые в редактируемые тексты. Редактор выполняет также всевозможный сервис: нумерацию карт, их копирование, выдачу карт с текстами в COSY-виде (см. § 8) и т. д.

Основная функция *библиотекаря* — ведение библиотек. В системе имеется постоянная библиотека, обычно располагаемая на МБ, библиотеки общего пользования [19], и личные библиотеки пользователей, расположенные на МЛ или МД, а также временная библиотека, создаваемая на время прохождения задачи.

Библиотекарь производит такие операции, как запись программ в библиотеки, считывание их во временную библиотеку, исключение отдельных подпрограмм из библиотеки и т. д. Он производит также операции сервисного характера: печать каталогов библиотек, выдачу отдельных подпрограмм на перфокарты и т. п.

Основная функция *диспетчера* — обработка потоков задач в мультипрограммном режиме. Потоки задач формируются на МЛ или МД (буферах ввода). Одновременно могут обрабатываться отдельные задачи, вводимые с читающего устройства.

Результаты счета могут быть записаны на МЛ или МД (буферы вывода) либо выданы непосредственно на внешние выводные устройства.

В последнее время в рамках диспетчера разработана система «мультитайп» [7], позволяющая организовать режим мультидоступа с удаленных терминалов, оборудованных телетайпами, а также дисплеями.

Диспетчер позволяет накапливать статистику о работе всех внешних устройств, а также распределять машинное время между пользователями и вести его учет. В процессе обработки задач диспетчер реагирует на сигналы, поступающие с операторского устройства, а также информирует человека-оператора о ходе процесса обработки задач и о сбоях в работе отдельных устройств машины.

§ 8. Пакет задачи пользователя

Каждая задача для счета на БЭСМ-6 оформляется в виде так называемого пакета. *Пакет задачи* — это перфокарты или их копии в буфере системы «мульти-тайп», содержащие информацию об этой задаче. Пакет содержит информацию, необходимую самой задаче (подпрограммы и данные), и информацию, необходимую системе для обработки этой задачи на машине (управляющие карты).

Остановимся подробнее на вопросе о том, какую информацию о задаче необходимо сообщить системе посредством управляющих карт.

Каждой задаче необходимы определенные ресурсы машины и системы. Эти ресурсы, в частности, включают в себя:

- 1) объем оперативной памяти, отводимый для задачи;
- 2) лимит машинного времени, требуемый для решения задачи;
- 3) внешние устройства ввода-вывода, которые использует задача;
- 4) внешние запоминающие устройства, используемые задачей;
- 5) библиотеки программ, к которым обращается задача.

В большинстве случаев задача должна заказывать требуемые ресурсы. Устройства ввода-вывода, магнитные барабаны, а также нестандартные внешние устройства (графопостроители и др.) выделяются задаче без заказа. Кроме того, каждая задача получает доступ к программам из постоянной библиотеки. В эту библиотеку входят, например, подпрограммы вычисления стандартных математических функций (см. § 14), умножение и деление целых величин и др.

В системе предусмотрено стандартное выделение некоторых ресурсов в случае отсутствия их заказа. Такими стандартно выделяемыми ресурсами, в частности, являются:

- 1) 24К ячеек оперативной памяти, из них несколько более 21К для самой задачи и около 3К для нужд системы по обслуживанию данной задачи;

2) 5 минут коммерческого машинного времени, т. е. времени, в течение которого задача занимает какие-либо устройства машины.

Вместо стандартно выделяемых 24К оперативной памяти задача может заказать больший объем: 25К или 32К. Из них в обоих случаях около 3К будет отдано системе. Лимит коммерческого времени заказывается в часах и минутах.

Остальные ресурсы (например, магнитные ленты, выводной перфоратор, библиотеки программ) заказываются в случае необходимости. Отметим, что в процессе счета задача не может затребовать ресурсов сверх заказанных.

Кроме информации о ресурсах, задача должна указать информацию о режиме использования системы. Иначе говоря, задача должна указать, какие виды обслуживания со стороны системы ей требуются. Например, задаче требуется провести трансляцию с фортрана с выдачей текста транслируемых подпрограмм, после чего выйти на счет.

Основная информация о режиме работы системы может быть представлена в следующем виде:

1) какие трансляторы и в каком порядке должны быть вызваны для трансляции отдельных подпрограмм задачи;

2) какие дополнительные действия требуются от системы в процессе трансляции (печать текстов подпрограмм; выдача оттранслированных подпрограмм на перфокарты и т. д.);

3) требуется ли задаче выход на счет;

4) в случае выхода задачи на счет, надо ли печатать таблицу загрузки, содержащую адреса подпрограмм и COMMON-блоков в памяти машины;

5) какие данные использует задача — текстовые или двоичные;

6) какие дополнительные операции до выхода задачи на счет надо выполнить (редактирование, запись в личные библиотеки, объявление главной подпрограммы и др.).

Отметим, что информация, содержащаяся в пунктах 1) и 2), касается только процесса трансляции. Информация же из пунктов 4) и 5) учитывается лишь при выходе задачи на счет.

В случае отсутствия информации, перечисленной в пунктах 1)–5), система устанавливает стандартный режим работы, а именно:

1) если не указан режим трансляции, то устанавливается режим трансляции с фортрана;

2) если не указаны никакие дополнительные действия системы в процессе трансляции, то пользователь получает на листинге тексты подпрограмм на их входных языках, снабженные некоторой дополнительной информацией;

3) если не указано, что задаче требуется выход на счет, то выхода на счет не происходит;

4) если отсутствует информация относительно выдачи таблицы загрузки, то в случае фактического выхода задачи на счет эта таблица печатается на листинге;

5) если не указан характер данных, то данные рассматриваются как текстовые.

Кроме информации, зависящей от самой задачи, системе требуется некоторая обязательная информация о каждой задаче. Эта обязательная информация располагается на трех управляющих картах и одной дополнительной (не управляющей) карте.

Управляющая карта начинается с символа * в первой колонке перфокарты. После этого символа указывается некоторый фиксированный текст, определяющий тип управляющей карты. Далее может быть указана некоторая дополнительная информация в зависимости от типа управляющей карты.

Пакет задачи должен начинаться с управляющей карты вида

*NAME — <имя пользователя>

Здесь имя пользователя может состоять из произвольного набора символов, содержащего хотя бы один символ, отличный от пробела. При отсутствии этой карты выдается диагностика, и обработка пакета прекращается.

Последней картой пакета должна быть управляющая карта *END_FILE. Вслед за этой картой обычно ставится карта, обозначающая признак конца ввода пакета (так называемый диспетчерский конец). Эта карта имеет пробивки во всех 12 строках колонок 1-й и 41-й (и только в них).

Обязательной управляющей картой является также карта вида

***PASS** — <шифр пользователя>

Здесь шифр пользователя означает специальное наименование (шифр, или пароль), присвоенное данному пользователю и открывающее ему доступ к машине. Этот шифр, состоящий не более чем из 6 букв и цифр и начинающийся с буквы, заранее вносится в каталог пользователей данной машины, который хранится в системе. Шифр пользователя печатается на листинге в виде ***** с целью сохранения его тайны. Этот шифр используется также для накопления системой статистики по прохождению задач. Указанная карта обычно ставится непосредственно после карты ***NAME**.

Прежде чем переходить к описанию других управляющих карт, сделаем одно замечание относительно формы записи этих карт. Некоторые управляющие карты, кроме основного текста, содержат дополнительную информацию (например, уже упоминавшиеся карты ***NAME** и ***PASS**). Эта информация отделяется от основного текста карты в одних случаях пробелом, в других случаях двоеточием. Если разделителем служит двоеточие, то информация, расположенная справа от него, имеет, как говорят, «плавающий формат», т. е. она не привязана к определенным колонкам перфокарты. Если разделителем служит пробел, то информация справа от него чаще всего имеет фиксированный формат. Некоторые управляющие карты имеют несколько эквивалентных записей в плавающем и фиксированном форматах. В частности, карта ***PASS** допускает запись в виде

***PASS:** <шифр пользователя>

В начале пакета, т. е. после карт ***NAME** и ***PASS**, но ранее карт с подпрограммами, должны быть расположены карты заказа ресурсов.

Управляющая карта заказа ресурсов имеет вид

***ASSIGN** <наименование ресурса>

Заказ некоторых ресурсов можно производить в плавающем формате (обычно без слова **ASSIGN**). В таких случаях будет, как правило, приводиться более удобная «плавающая» форма записи.

Наиболее часто используется карта заказа лимита коммерческого машинного времени:

*TIME: <часы> . <минуты>

Здесь <часы> и <минуты> занимают по две колонки и разделяются точкой. Например, карта *TIME : 02 . 15 означает заказ 2 часов 15 минут коммерческого времени.

Довольно часто задача может использовать библиотеки программ общего пользования. Заказ такой библиотеки производится картой вида

*LIBRARY: <номер библиотеки>

Количество библиотек общего пользования и их состав может быть различным в зависимости от характера задач, решаемых в данном вычислительном учреждении. Получила распространение библиотека программ ОИЯИ, содержащая более 300 программ и, в частности, многие широко употребляемые программы численного анализа, написанные на фортране и автокоде madlen. Программы из этой библиотеки (в оттранслированном виде) оформлены в системе «Дубна» в виде библиотек общего пользования с номерами 1 и 2. Общая библиотека с номером 3 содержит оттранслированные алгольные процедуры (численного анализа и др.).

Среди прочих карт заказа ресурсов отметим карту заказа перфоратора:

*ASSIGN — PUNCH

которая необходима в случае использования выводного перфоратора (как при счете, так и при трансляции).

Особую группу карт заказа ресурсов составляют карты заказа магнитных лент и файлов на магнитных дисках.

Приведем новую форму записи карты заказа магнитных лент (см. [8]):

*TAPE: <№ кассеты> — <имя ленты>,

<указатель МЛ>, $\left\{ \begin{array}{c} R \\ W \end{array} \right\}$ <№ зоны подвода>

Здесь № кассеты обычно задается числом от 1 до 4095, имя ленты — идентификатором, содержащим не более

6 символов. В качестве указателя МЛ можно использовать двузначное восьмеричное число, первая цифра которого (от 3 до 6) обозначает математический номер направления, а вторая (от 0 до 7) — математический номер устройства. Можно указывать также «фортранный» номер магнитофона (от F1 до F15). Указатель R означает, что МЛ будет использоваться только в режиме чтения, а указатель W — что МЛ можно использовать как в режиме записи, так и в режиме чтения. Номер зоны подвода указывается в восьмеричном виде.

Рассмотренная карта содержит полную информацию о заказываемой МЛ и поэтому она заменяет карту заказа МЛ в старой форме записи (см. [27]) и одновременно карту *СНЕСК. Отметим, что личная лента пользователя должна быть поименована. Если надо отключить проверку имени ленты, то на месте ее имени указывают NO_СНЕСК.

На машинах, имеющих накопители на МД, можно заказывать файлы на дисках. Для дисков, отведенных в личное пользование, требуются предварительные процедуры, связанные с подготовкой их к работе (см. [8]). В случае использования рабочих файлов (или SCRATCH-файлов) никаких дополнительных процедур, кроме заказа этих файлов, не требуется.

Заказ рабочего файла на дисках производится картой вида

*FILE : SCRATCH, <указатель МЛ>, W, <длина>

Здесь указатель МЛ соответствует той рабочей МЛ, вместо которой будет использован данный рабочий файл. Длина файла указывается в виде восьмеричного числа, обозначающего количество зон на диске (объемом 1К каждая).

Пр и м е р. *FILE : SCRATCH, 41000, W, 40
Здесь заказан рабочий дисковый файл длиной 40В зон, заменяющий рабочую МЛ с математическим номером 41 или, что то же самое, МЛ с фортранным номером 1.

После карт заказа ресурсов располагаются (в произвольном порядке) подпрограммы, входящие в состав пакета. Эти подпрограммы могут быть написаны на фортране, автокоде, алголе или других входных языках системы либо представлены в оттранслированном виде, т. е. в виде стандартных массивов (CM).

Каждой группе подряд стоящих подпрограмм, написанных на одном и том же языке программирования, должна предшествовать одна из управляющих карт:

***FORTRAN, *ASSEMBLER или *ALGOL**

Первая из этих карт устанавливает признак трансляции с фортрана, вторая — с автокода и третья — с алгола. В дальнейшем возможно появление дополнительных управляющих карт этого типа.

Действие каждой из трех описанных карт распространяется на все подпрограммы, расположенные после данной карты, кроме подпрограмм, представленных в виде стандартных массивов. Каждая из них отменяет действие любой из двух других карт. Перед началом обработки пакета стандартно устанавливается признак трансляции с фортрана.

Рассмотрим пример возможной компоновки пакета:

```
*NAME_ИВАНОВ, 10.12.75
*PASS_SATURN
*TIME : 00.10
*LIBRARY : 1
*LIBRARY : 2
<подпрограммы на фортране и CM>
*ASSEMBLER
<подпрограммы на автокоде и CM>
*FORTRAN
*NO_LIST
<подпрограммы на фортране и CM>
*EXECUTE
<карты данных>
*END_FILE
<диспетчерский конец>
```

Здесь задача заказывает 10 минут коммерческого времени и библиотеки общего пользования с номерами 1 и 2. Поскольку первой расположена группа подпрограмм на фортране (среди которых могут быть и стандартные массивы), карта ***FORTRAN** перед ними не обязательна. Карта ***NO_LIST** описана ниже.

Управляющая карта *EXECUTE означает, что задаче требуется выход на счет. После этой карты могут быть расположены карты данных и обязательные карты *END_FILE и <диспетчерский конец>.

Отметим, что карта *EXECUTE, кроме требования выйти на счет, означает также, что для загрузки подпрограмм и COMMON-блоков будет использоваться динамический загрузчик. Если задача имеет разделы, сменяющие друг друга в памяти (см. § 58), то для загрузки каждого раздела может оказаться выгоднее использовать статический загрузчик. В этом случае вместо карты *EXECUTE ставится карта *CALL_EXECUTE. Заметим, что для работы статического загрузчика требуется информация о структуре раздела задачи (см. [27]).

Дадим пояснение к понятию стандартного массива.

Стандартный массив — это результат трансляции отдельной подпрограммы, выданный, например, на перфокарты в специальной кодировке. Каждая карта стандартного массива имеет отличительный признак — пробивки в 5-й и 7-й строках первой колонки. Карты стандартного массива нумеруются в восьмеричном виде, начиная с нулевой карты, которая имеет характерные пробивки, образующие символы наименования подпрограммы. Использование стандартных массивов значительно сокращает размеры пакета, экономит машинное время (так как не требуется повторной трансляции соответствующих подпрограмм) и, кроме того, исключает возможность ввода карт, расположенных с нарушением порядка их нумерации.

Для получения стандартных массивов служит управляющая карта *PUNCH (с картой *ASSIGN_PUNCH в начале пакета), располагаемая перед подпрограммами, которые подлежат выдаче в виде стандартных массивов. Действие этой карты распространяется на все последующие подпрограммы (кроме, естественно, тех из них, которые уже представлены стандартными массивами). Отмена режима перфорации стандартных массивов производится картой *NO_PUNCH.

Рассмотрим теперь вопрос о способах кодировки символов при пробивке перфокарт на различных устройствах подготовки данных.

В пакете задачи допускаются карты, пробитые в построчной кодировке на устройствах УПП БЭСМ-6 и УПП М-220. Карты, пробитые по колонно, могут быть представлены в кодировке CDC, см. рис. 1 на стр. 43 (используемой в устройствах ICL-72, АРИТМА-130 и некоторых других), в коде КПК-12 СССР, см. рис. 2 на стр. 44 (используемой, например, в устройствах ЕС-9015), а также в редко применяемой кодировке ИСТ. Отметим, что код КПК-12 СССР отличается от распространенной кодировки IBM добавлением русских букв.

Карты, пробитые в построчной кодировке, могут располагаться среди по колонно пробитых карт без всякой информации о способе их кодировки.

Для того чтобы отличать карты, пробитые в разных по колонных кодировках, применяются карты $\begin{smallmatrix} 7 \\ 9 \end{smallmatrix}$ CDC,

$\begin{smallmatrix} 7 \\ 9 \end{smallmatrix}$ IBM, $\begin{smallmatrix} 7 \\ 9 \end{smallmatrix}$ ИСТ, указывающие тип кодировки CDC, IBM (КПК-12 СССР) и ИСТ соответственно. Здесь символы $\begin{smallmatrix} 7 \\ 9 \end{smallmatrix}$ означают пробивки в 7-й и 9-й строках первой ко-

лонки. Если в пакете используется только по колонная кодировка CDC, то тип кодировки можно не указывать. При наличии карты, указывающей тип по колонной кодировки, все последующие по колонно пробитые карты будут восприниматься в этой кодировке. Таким образом, каждая смена типа по колонной кодировки должна сопровождаться картой, указывающей очередной тип кодировки.

Подпрограммы, входящие в состав пакета, могут быть представлены в виде так называемых COSY-массивов (сокращение от английского COmpress SYmbol). В COSY-массив можно объединить несколько подпрограмм, написанных на одном и том же языке программирования. Каждому такому массиву присваивается определенное наименование, которое пробивается на первой карте этого массива тем же способом, что и наименование стандартного массива. Использование COSY-массивов позволяет (как и в случае стандартных массивов) сократить объем пакета и избежать ввода неверно заданного массива карт (так как карты пронумерованы). Однако, в отличие от стандартных

массивов, в COSY-массивах сохраняется текстовый вид информации. Эти массивы транслируются наравне с подпрограммами, заданными в текстовом виде.

Получение COSY-массивов и работа с ними (внешение исправлений, изъятие и дополнение информации) производится с помощью редактора текстов (см. [13]). Отличительный признак COSY-массива — пробивки в строках 7-й и 9-й первой колонки.

Рассмотрим еще несколько управляющих карт.

Если необходимо отключить печать текстов транслируемых подпрограмм, то перед соответствующей группой подпрограмм должна быть поставлена управляющая карта *NO_LIST. Восстановление стандартного режима печати производится картой *STANDARD_LIST.

В ряде случаев бывает необходимо выдать на печать не только текст подпрограммы на фортране, но и текст ее автокодного представления, полученного после трансляции с фортрана. В этом случае перед нужной подпрограммой ставится карта *FULL_LIST.

Действие карт *NO_LIST и *STANDARD_LIST и *FULL_LIST распространяется на все подпрограммы, расположенные после данной карты. При этом отменяется режим, установленный предыдущей картой из числа рассматриваемых. Отметим, что перед началом обработки пакета устанавливается режим, соответствующий карте *STANDARD_LIST. В случае, когда требуется провести счет (т. е. при наличии карты *EXECUTE или *CALL_EXECUTE), иногда бывает необходимо выполнить некоторые дополнительные действия перед выходом задачи на счет.

Задача может затребовать больший объем оперативной памяти, чем стандартно выделяемые 24K МОЗУ. Заказ памяти объемом 25K производится картой *CALL_FULLMEMORY, а заказ памяти объемом 32K — картой *CALL_FICMEMORY.

В некоторых случаях возникает необходимость объявления главной (или головной) подпрограммы. Стандартно главной подпрограммой объявляется фортранная PROGRAM (либо ее автокодный или алгольный аналог). Можно объявить главной другую подпрограмму.

В этом случае ставится управляющая карта

*MAIN_ <наименование главной подпрограммы>.

Подпрограмма, объявляемая главной, не должна иметь параметров.

Для того чтобы избежать прекращения счета при некоторых ошибках, связанных с вводом и выводом информации (см. § 31), можно использовать управляющую карту *NO__OPT (или, что то же самое, *NO__OPTIMIZATION).

Последние четыре управляющие карты можно располагать в любом месте пакета (вне подпрограмм), но после карт заказа ресурсов и до карты *EXECUTE (или *CALL__EXECUTE).

После карты *EXECUTE (или ее аналога) располагаются (если необходимо) карты данных, вводимые в процессе счета задачи. Данные обычно пробиваются в текстовом виде, т. е. в виде последовательности символов, заданных в любой из допустимых кодировок. Однако иногда приходится вводить так называемые двоичные данные (например, карты, пробитые в двоичном коде машины БЭСМ-4). Каждый массив двоичных данных снабжается признаком начала (карта *BINARY) и признаком конца (карта *END__BINARY). Отметим, что двоичные данные нельзя вводить фортран-ным оператором READ. Для ввода их служит системная подпрограмма BINARY (см. [27]).

Мы рассмотрели далеко не полный перечень возможностей системы Дубна. В частности, осталась нерассмотренной работа важного раздела системы — редактора текстов (см. [13]). Достаточно подробное и полное описание возможностей, предоставляемых пользователю системы Дубна, дано в [27].

Г Л А В А II

ЯЗЫК ФОРТРАН В СИСТЕМЕ «ДУБНА»

§ 9. Фортран как язык программирования

Фортран принадлежит к числу наиболее распространенных языков программирования высокого уровня. Языки подобного типа принято называть машинно независимыми или проблемно ориентированными. Первое из этих названий подчеркивает независимость данного языка от конкретной ЭВМ. Второе название означает, что этот язык в основном предназначен для решения задач определенного типа.

Область применения фортрана — это в первую очередь задачи научно-технического характера. Такие задачи характеризуются достаточно большим количеством вычислительных операций (в основном арифметических) на единицу вводимой информации.

Название языка фортран (FORTRAN) происходит от английских слов FORMula TRANslation — перевод формул. Тем самым подчеркивается предпочтительная область применения этого языка — задачи, связанные с расчетами по формулам. Однако фортран можно успешно использовать и для решения задач иного характера, например, экономических.

Рассмотрим главные особенности фортрана как языка программирования.

Прежде всего отметим *принцип автономности*. Он состоит в том, что программа на фортране строится из автономных, независимо составляемых и транслируемых частей, называемых *подпрограммами*. Каждую подпрограмму можно автономно отладить и записать в библиотеку программ в оттранслированном виде.

Использование библиотечных программ значительно упрощает процесс составления и отладки больших программ.

Другой важной особенностью фортрана является так называемый *принцип умолчания*. Суть его в том, что в фортране разрешается не описывать многие объекты. Например, можно не описывать тип переменной величины. В этом случае переменной будет приписан тип вещественный или целый в зависимости от первой буквы ее наименования (см. § 13).

Следующей особенностью фортрана является *развитый аппарат ввода-вывода* информации в виде, удобном для использования. Это позволяет, например, выдавать на печать готовые документы: таблицы, гистограммы, графики.

Отметим, наконец, что текст программы на фортране легко читается, т. е. по фортранной записи нетрудно понять смысл алгоритма.

В процессе развития фортрана возникло несколько версий этого языка.

Здесь описывается версия фортрана для БЭСМ-6, известная под названием фортран-Дубна [41]. Эта версия близка к версии фортран-IV (см. [15]) и к стандарту языка фортран [40].

§ 10. Запись программы на фортране.

Операторы фортрана

Запись программы на фортране имеет вид текста, состоящего из отдельных символов. Множество символов, используемых в данном языке, обычно называют *алфавитом* этого языка.

Алфавит фортрана состоит из следующих символов:

- 1) 26 прописных букв латинского алфавита от A до Z,
- 2) 10 цифр от 0 до 9,
- 3) 10 специальных символов:

+	плюс	(левая скобка
—	минус)	правая скобка
*	звездочка	,	запятая
/	косая черта	.	точка
=	равно	'	апостроф

Транслятор на БЭСМ-6 допускает также использование заглавных букв русского алфавита и символа \$, или ◇. В комментариях и в текстовых величинах разрешается использовать любые символы, перечисленные в приложении 2.

Программа на фортране, как уже было сказано, состоит из отдельных составных частей — подпрограмм. Каждая подпрограмма является независимой от других подпрограмм в смысле выбора обозначений. Это надо понимать так, что любое обозначение (кроме наименований подпрограмм и общих блоков), принятое в данной подпрограмме, имеет силу только внутри этой подпрограммы и не затрагивает обозначений, принятых в других подпрограммах.

Подпрограмма на фортране состоит из отдельных частей — *операторов*. Каждый оператор представляет собой некоторый самостоятельный фрагмент подпрограммы. В фортране имеются операторы двух типов — выполняемые и невыполняемые (или декларативные).

Каждый *выполняемый оператор* определяет некоторую самостоятельную часть всего вычислительного процесса, реализованного в данной подпрограмме. Выполнение этих операторов производится, вообще говоря, в порядке их следования в подпрограмме.

Невыполняемые операторы, в отличие от выполняемых, сообщают транслятору некоторую дополнительную информацию, необходимую для трансляции подпрограммы. При этом порядок расположения невыполняемых операторов по отношению друг к другу может быть произвольным.

Операторы фортрана обычно записывают на специальных бланках. Каждая строка такого бланка содержит 80 позиций и, следовательно, позволяет расположить 80 символов. Для записи операторов не обязательно использовать бланки, но необходимо помнить о том, что каждому символу (буква, цифра, знак, пробел и т. п.) должна соответствовать одна позиция на данной строке.

Такой способ записи операторов фортрана обусловлен тем, что текст каждого оператора будет пробит на стандартной 80-колонной перфокарте (см. рис. 1, 2). При этом каждому символу текста будет поставлена в соответствие определенная колонка перфокарты.

При пробивке перфокарт на устройствах с покомной кодировкой (типа ЕС-9015, ICL и др.) каждому символу ставится в соответствие определенная комбинация отверстий в данной колонке перфокарты. Сам символ при этом печатается над той же колонкой у верхнего края перфокарты.

Каждый оператор располагается, как правило, на отдельной перфокарте. При этом для размещения оператора отводятся колонки с 7-й по 72-ю включительно (так называемое *поле оператора*). Для записи «длинных» операторов может не хватить одной перфокарты. В этом случае запись оператора продолжается на последующих перфокартах, у которых в 6-й колонке должен быть указан любой символ, отличный от пробела. Этот символ принято называть *признаком продолжения*.

Перед началом записи оператора в колонках с 1-й по 5-ю может быть указана метка оператора. *Метка оператора* — это целое положительное число без знака, не превосходящее 32767. Метка служит для того, чтобы можно было сослаться на данный оператор (например, передать на него управление).

Колонки с 73-й по 80-ю служат для записи *комментария*. Обычно эти колонки используют для нумерации перфокарт. Комментарий при трансляции не учитывается, однако выдается на печать вместе с текстом подпрограммы.

Важно отметить, что пробелы в записи операторов и меток транслятором не принимаются во внимание. Они удобны для наглядности подпрограммы. С «точки зрения транслятора», записи

GO TO 20 GO TO20 GOT O 2 0 GOTO20
эквивалентны, но для чтения наиболее удобна первая из приведенных записей.

На одной перфокарте можно располагать несколько операторов. В этом случае они отделяются друг от друга знаком \$ (при пробивке на УПП используется знак ◇). После знака \$ нельзя записывать оператор с меткой. Знак \$ нельзя ставить непосредственно после оператора FORMAT. Отметим, что в некоторых версиях фортрана не разрешается располагать несколько операторов на одной перфокарте. Поэтому не рекомендуется без особой необходимости использовать эту возможность.

Кроме операторов, программа может содержать карты-комментарии. Признаком такой карты служит буква С в первой колонке. В колонках с 2-й по 80-ю карты-комментария может быть расположен произвольный текст, содержащий символы ISO (см. приложение 2). Каждая карта-комментарий должна иметь букву С в первой колонке.

Резюмируя сказанное, сформулируем основные

Правила записи операторов фортрана

1. Для размещения операторов фортрана используются колонки перфокарты с 7-й по 72-ю включительно. (В книге операторы фортрана всюду расположены, начиная с седьмой колонки.)

2. Метки операторов располагаются в колонках с 1-й по 5-ю.

3. Колонка 6-я служит для размещения признака продолжения.

4. Колонки с 73-й по 80-ю предназначены для размещения комментария.

5. На одной карте разрешается располагать несколько операторов, отделяемых друг от друга знаком \$ (или \diamond).

6. Карта-комментарий начинается с буквы С в первой колонке.

7. Для достижения большей наглядности текста программы при записи операторов и меток можно использовать пробелы.

Приведем примеры записи операторов фортрана. Колонка 6-я выделена вертикальными чертами.

15		A=1.
		CONTINUE
		S=X ** 2+2. * SIN (X)—
		* 15.3 * SQRT (X ** 2+1.)
10		B=A \$ X=A ** 2+4. \$ GO TO 20
С ЭТО	—	ЗАПИСЬ ОПЕРАТОРОВ ФОРТРАНА

§ 11. Типы величин, используемых в фортране

В фортране используются величины следующих типов:

- | | |
|------------------|-------------------------|
| 1) вещественные, | 4) с двойной точностью, |
| 2) целые, | 5) логические. |
| 3) комплексные, | |

Разрешается также использовать текстовые (холлери-товские) и восьмеричные величины. Текстовым и восьмеричным величинам должен быть приписан целый тип. Каждая из величин перечисленных типов имеет свое внутреннее представление в машине и строго определенный набор допустимых операций. Для размещения каждой величины вещественного, целого или логического типа отводится по одному машинному слову, а для размещения каждой величины комплексного типа или с двойной точностью — по два машинных слова.

Величина *вещественного* типа имеет машинное представление в виде нормализованного числа (см. § 2). Напомним, что диапазон нормализованных чисел (по модулю) на БЭСМ-6 лежит в пределах от 10^{-19} до 10^{19} , а точность машинного представления — до двенадцати десятичных знаков.

Величина *целого* типа представляется в БЭСМ-6 в ненормализованном виде (см. § 2), с порядком 40 и мантисой, расположенной в младших разрядах слова. Отсюда следует, что её модуль может изменяться от 0 до $2^{40}-1$.

Комплексная величина размещается в двух последовательных машинных словах. В первом из этих слов располагается действительная, а во втором — мнимая часть комплексной величины. Действительная и мнимая части комплексной величины могут рассматриваться отдельно друг от друга как вещественные величины.

Величина с *двойной точностью* располагается в двух последовательных машинных словах. Мантисса первого машинного слова содержит 40 старших разрядов мантиссы, знак ее соответствует знаку всей 80-разрядной мантиссы. Порядок первого машинного слова содержит 6 младших разрядов порядка. Старший разряд этого слова содержит знак порядка (1 — положительный, 0 — отрицательный). Второе машинное слово содержит 40 младших разрядов мантиссы (с 0 в 41-м разряде) и 6 старших разрядов порядка (с 0 в 48-м разряде). Таким образом, порядок величины с двойной точностью состоит из 12 двоичных разрядов, что соответствует диапазону от 10^{-1232} до 10^{1232} . Мантисса содержит 80 двоичных разрядов, что соответствует 24 десятичным знакам. Отметим, что в случае, когда величина с двойной точностью лежит в диапазоне,

допустимом для величин вещественного типа, первое слово ее машинного представления совпадает с величиной вещественного типа.

Для представления величин логического типа используется младший разряд машинного слова (1 — истина, 0 — ложь).

Текстовые величины располагаются в машинном слове посимвольно, начиная со старших его разрядов. Под каждый символ (включая и пробелы) отводится 8 двоичных разрядов (1 байт), т. е. в каждом машинном слове располагается по 6 символов. Отсюда следует, что текстовая величина может содержать не более 6 символов. Если число символов меньше шести, то при записи в машинном слове справа будет добавлено недостающее число пробелов. Каждый символ представляется в виде восьмиразрядной комбинации 0 или 1 в соответствии с кодом ISO (см. приложение 2).

Восьмеричная величина представляется в машине в виде последовательности из 16 восьмеричных цифр.

Над величинами вещественного, целого, комплексного типов и с двойной точностью можно производить арифметические операции сложения, вычитания, умножения, деления и возведения в степень. В результате получается величина одного из четырех перечисленных типов. Более подробно этот вопрос рассмотрен в § 15. Над величинами рассмотренных четырех типов можно производить также операции отношения (см. § 16), дающие в результате величину логического типа.

Величины логического типа допускают лишь логические операции, дающие в результате величину того же типа.

Над текстовыми величинами можно в принципе производить любые операции, допустимые для величин целого типа. Однако смысл имеют лишь операции сравнения (см. § 16), которые устанавливают, совпадают две данные текстовые величины или нет.

Операции над восьмеричными величинами производятся по тем же правилам, что и над величинами целого типа. Отметим, что в этом случае транслятор не проверяет правильности записи восьмеричной величины, как величины целого типа.

Каждая из величин может быть либо константой, либо переменной (без индексов или с индексами), либо функций.

Результатом вычислений всегда является некоторое конкретное значение переменной величины. Это значение записывается в определенные ячейки оперативной памяти машины (одну или две — в зависимости от типа величины) и может быть выдано на внешние носители информации: бумажную ленту АЦПУ, перфокарты, магнитную ленту и т. п.

§ 12. Запись констант на фортране

Часть величин, участвующих в вычислительном процессе, не меняет своего значения в ходе его выполнения. Такие величины на фортране называются *константами*.

Константа может принадлежать к одному из 5 типов, рассмотренных в § 11. В зависимости от типа она может быть представлена на фортране одним или несколькими способами. Отметим, что множество допустимых значений констант (кроме логических) зависит от типа машины.

Наиболее распространенным типом констант являются *вещественные* константы. Запись констант этого типа на фортране напоминает запись обычных десятичных чисел, содержащих целую и дробную часть. Однако в качестве разделителя целой и дробной части здесь принята не запятая, а точка. Например, десятичное число 3,14 на фортране будет представлено как 3.14.

Отметим, что десятичная точка является характерным признаком вещественной константы. Например, 1. (единица с точкой) является вещественной константой, а 1 (единица без точки) уже не является вещественной константой.

При записи обычных десятичных чисел часто используется степень числа 10, например, $3 \cdot 10^6$, $-2,5 \cdot 10^{-4}$. На фортране предусмотрена возможность аналогичной записи вещественных констант, а именно: в качестве основания степени вместо числа 10 используется буква E, а показатель степени указывается справа от нее. Рассмотренные выше числа на фортране запишутся так: 3.E6, $-2.5E-4$

Знак константы записывается перед самой константой, причем знак плюс можно опустить.

Суммируя сказанное, укажем общий вид записи вещественной константы на фортране:

$$\pm n.mE \pm S$$

Здесь $\pm n$ обозначает целую часть константы, m — дробную часть, $\pm S$ — показатель степени при 10. Знак $+$ перед n и S можно опустить. При отсутствии дробной части допускается запись вида $\pm nE \pm S$. В записи вещественной константы обязательна десятичная точка либо буква E и хотя бы одна цифра в целой или дробной части. Приведем примеры записи вещественных констант на фортране (см. табл. 1).

Т а б л и ц а 1

Обычная запись числа	Запись соответствующей вещественной константы на фортране
3,14159	3.14159
1975	1975.
0,6985	0.6985 или .6985
$-1,234 \cdot 10^5$	$-1.234 E5$
$0,00317 \cdot 10^{-11}$	$0.00317E-11$ или $.00317E-11$
$3 \cdot 10^6$	$3.E6$ или $3E6$

Модуль вещественной константы должен находиться в интервале от 10^{-19} до 10^{19} или равняться нулю. Константы, меньшие по модулю, чем 10^{-19} , преобразуются транслятором в нуль; если модуль константы превышает 10^{19} , то транслятор выдает сигнал об ошибке.

Количество значащих цифр вещественной константы не должно превышать 12. Если константа содержит более 12 десятичных цифр, то лишние цифры справа при трансляции будут заменены нулями.

Целая константа записывается в виде последовательности не более чем 12 десятичных цифр. Перед отрицательной константой ставится знак $-$. Например, 1975, -805 , 32767.

Комплексная константа представляет собой упорядоченную пару вещественных констант, разделенных запятой и заключенных в круглые скобки. Например,

комплексная константа (3., -12.) соответствует комплексному числу $3-12i$.

Константа с двойной точностью может состоять не более чем из 24 значащих цифр. Признаком двойной точности служит буква D, справа от которой записывается показатель степени при 10. Перед отрицательной константой ставится знак —.

П р и м е р ы констант с двойной точностью. Число e с 15 знаками после запятой можно записать так: 2.718281828459045D0.

Известное «шахматное число» ($2^{64}-1$) можно записать, например, так: 18446744073.709551615D9.

Логическая константа может принимать одно из двух значений: .TRUE. или .FALSE. (истина или ложь).

Текстовая константа может содержать любые символы ISO (см. приложение 2) в количестве не более шести, включая пробелы. Признаком текстовой константы служат стоящие слева символы nH , где n — количество символов в константе ($n \leq 6$). Например, 3HABC, 6H_CAMAC. (В последнем примере текстовая константа начинается с пробела.)

Применяется и другой способ записи текстовых констант. Он состоит в том, что вместо символов nH используются апострофы, в которые заключается текстовая константа. Например, приведенные выше текстовые константы можно записать так: 'ABC' и 'CAMAC'.

Восьмеричная константа может содержать не более 16 восьмеричных цифр, за которыми следует буква B. Перед отрицательной константой ставится знак —. Например, 377B, -17432B. Если в записи восьмеричной константы содержится более 16 цифр, то транслятором будут отброшены лишние цифры слева.

Отрицательные восьмеричные константы преобразуются транслятором в обратный код, т. е. двоичные единицы заменяются нулями, а нули единицами. Например, восьмеричная константа -15B будет представлена в машине так же, как и восьмеричная константа 7777 7777 7777 7762B

У п р а ж н е н и я 1. Записать на фортране следующие константы:

а) -1, 236; б) 0, 00015; в) 1974; г) $1+2i$; д) i ; е) 1, 35.
10⁻⁶; ж) $2,78 \cdot 10^{20}$; з) 3,141592653589793; и) $-1,16 \cdot 10^{-28}$.

2. Найти ошибки в записи следующих констант:

а) 1, 25; б) $-5*10**2$; в) (1, 0); г) 1.35E20

§ 13. Переменные величины

Величина, значение которой может изменяться в ходе выполнения подпрограммы, называется *переменной*. Для каждой переменной транслятор отводит одну или две ячейки памяти (в зависимости от типа переменной). В каждый конкретный момент времени в этих ячейках содержится текущее значение соответствующей переменной. Это текущее значение сохраняется до тех пор, пока в данную ячейку (или ячейки) не будет занесено новое значение переменной. Новое значение переменной при записи в память «стирает» старое ее значение.

Для того чтобы отличать одни переменные от других, каждой переменной дается определенное наименование, или идентификатор. *Идентификатор* — это последовательность букв и цифр, начинающаяся с буквы. Переменные могут быть двух видов: простые переменные и переменные с индексами. Обозначение простой переменной состоит только из идентификатора. Переменная с индексами кроме идентификатора имеет еще один, два или три индекса, разделенных запятыми и заключенных в круглые скобки. В качестве переменных с индексами могут использоваться только элементы массивов (см. § 21).

Отметим два важных обстоятельства, связанных с понятием идентификатора.

1. Кроме букв и цифр, идентификатор может содержать произвольное количество пробелов. При этом пробелы транслятором игнорируются, т. е. транслятор «выбрасывает» из идентификаторов все пробелы, оставляя лишь значащие символы. Таким образом, идентификаторы, состоящие из одних и тех же значащих символов (например, AB5 и AB5), считаются одинаковыми.

2. Общее количество значащих символов в идентификаторе не должно превышать 6. При наличии более 6 значащих символов транслятор «укорачивает» идентификатор до 6 первых значащих символов. В таких случаях выдается предупредительная диагностика и трансляция продолжается с учетом укороченных идентификаторов.

Поясним сказанное *п р и м е р а м и*. Обозначения X, A12, PSI, DELTA2, ALFAPRIM, FORTRAN, BESM_6, S_I_G_M_A являются идентификаторами. При трансляции пятый и шестой идентификаторы

будут «укорочены» до ALFAPR и FORTRA соответственно, а два последних идентификатора будут «сжаты» до BESM6 и SIGMA.

Приведем примеры обозначений, не являющихся идентификаторами фортрана: X+PTA, 5ABC. В первом случае в состав обозначения входит недопустимый символ +, во втором — обозначение начинается с цифры, а не с буквы.

Переменная величина может принадлежать одному из 5 типов, рассмотренных в § 11. Тип переменной — одна из ее важнейших характеристик, поэтому информация о типе переменной должна быть каким-то образом сообщена транслятору. Как мы увидим ниже (см. § 21), тип любой переменной может быть описан операторами описания типа. Однако, — и в этом состоит одна из важных особенностей фортрана, — если переменная имеет тип вещественный или целый, то нет необходимости его описывать. В этом случае тип будет определяться по первой букве идентификатора переменной.

Если идентификатор переменной начинается с одной из букв

I, J, K, L, M, N,

то, при отсутствии описания типа, переменной будет присвоен тип целый.

Если идентификатор переменной начинается с любой другой буквы, то, при отсутствии описания типа, переменной будет присвоен тип вещественный.

Поскольку тип, который надо присвоить некоторой переменной, чаще всего бывает известен заранее, то в случае, если он вещественный или целый, можно выбрать в качестве первой буквы идентификатора переменной ту, которая соответствует нужному типу. Практика показывает, что значительное большинство подпрограмм не требует иных типов переменных, кроме вещественных и целых. Так что операторы описания типа чаще всего отсутствуют.

У п р а ж н е н и е 3. Какие из следующих обозначений допустимы в качестве идентификаторов и какие недопустимы (объясните почему). а) DUBNA б) ALGOL60 в) PL/1 г) SIMULA д) HI ** 2 е) B ** 2—4. *A* C, ж) SQRT з) 5KSI и) OMEGA8

§ 14. Стандартные математические функции

Фортран позволяет производить вычисления по формулам. Формула же включает в себя числа, буквенные обозначения величин и математические функции.

Аналогами чисел, входящих в состав формул, на фортране являются константы. Аналогами буквенных обозначений служат идентификаторы переменных. Фортранные аналоги математических функций также называются функциями. Они имеют обозначения, очень сходные с обозначениями математических функций. Каждая такая функция имеет свой идентификатор и список аргументов, заключенный в круглые скобки. Например, $F(A,B)$, $FI(T, 0.5, X)$

Каждой функции соответствует определенное значение, полученное в результате ее вычисления при заданных значениях аргументов. Значение функции, как и всякая величина на фортране, принадлежит к одному из 5 типов, описанных в § 14.

Тип значения функции (или тип функции) определяется точно так же, как и тип переменной, т. е. либо по первой букве идентификатора функции, либо с помощью описания (см. § 21). Например, функции, рассмотренные выше, при отсутствии описания их типов будут иметь тип вещественный.

Таким образом, функция в фортране играет почти ту же роль, что и переменная величина, так как она имеет свой идентификатор и свой тип. Однако между переменной и функцией есть известная разница. Она состоит в том, что для хранения каждой переменной отводятся специальные ячейки памяти (одна или две), только для этого и предназначенные. Для хранения же значения каждой функции таких специальных ячеек памяти не отводится. Отсюда следует, что нельзя, например, непосредственно выдать на печать значение функции. Более подробно о функциях будет сказано в § 22 и 23. А пока запомним, что для обозначения функций служат идентификаторы и что каждой функции должен быть приписан определенный тип.

Для наиболее употребительных функций в фортране приняты стандартные обозначения, во многом сходные с их математическими обозначениями. См. табл. 2.

(Символ — обозначает отсутствие соответствующей функции.)

Т а б л и ц а 2

Математическая запись	x — вещественный	x — комплексный	x — с двойной точностью
\sqrt{x}	SQRT (X)	CSQRT (X)	DSQRT (X)
$ x $	ABS (X)	CABS (X)	DABS (X)
e^x	EXP (X)	CEXP (X)	DEXP (X)
$\ln x$	ALOG (X)	CLOG (X)	DLOG (X)
$\lg x$	ALOG10 (X)	CLOG10 (X)	DLOG10 (X)
$\sin x$	SIN (X)	CSIN (X)	DSIN (X)
$\cos x$	COS (X)	CCOS (X)	DCOS (X)
$\arcsin x$	ASIN (X)	—	—
$\arctg x$	ATAN (X)	—	DATAN (X)

Тип функции всюду совпадает с типом ее аргумента, кроме функции CABS(X), имеющей вещественный тип. Отметим, что тип стандартной функции жестко закреплен за нею и его нет необходимости описывать.

Наименования, присвоенные стандартным функциям, нельзя, вообще говоря, использовать для других целей.

§ 15. Арифметические операции и правила их выполнения.

Арифметические выражения

Аналогом математической формулы на фортране служит арифметическое выражение. Арифметическое выражение состоит из констант, переменных и функций, соединенных знаками арифметических действий. Для указания порядка выполнения этих действий можно использовать круглые скобки. Константы, переменные и функции, входящие в состав арифметического выражения, могут быть вещественного, целого, комплексного типов и с двойной точностью.

Знаками арифметических действий являются:

+ сложение
— вычитание

* умножение
/ деление

** возведение в степень

П р и м е р ы. Математические формулы

$$ax + 2 \sin^2 x - \frac{a}{x + 5,2} + abc^3,$$

$$x + \frac{1}{2x^2 + \frac{1}{3x^3 + \frac{1}{4x^4}}},$$

$$2 \cos^3 x + 3e^x + [ax + b(y + cz)]^3$$

записывают на фортране следующими арифметическими выражениями:

```
A * X + 2. * SIN (X) ** 2 - A / (X + 5.2) + A * B * C ** 3
X + 1. / (2. * X ** 2 + 1. / (3. * X ** 3 + 1. / (4. * X ** 4)))
2. * COS (X) ** 3 + 3. * EXP (X) + (A * X + B * (Y + C * Z)) ** 3
```

Сравнивая запись арифметического выражения на фортране с записью соответствующей математической формулы, можно заметить определенные различия между ними.

Все арифметические выражения записываются в один «этаж». Выражение, соответствующее формуле $\frac{a}{x + 5,2}$, на фортране записывается с помощью скобок, т. е. как $A / (X + 5.2)$, а не $A / X + 5.2$. Используются только круглые скобки.

В то же время можно увидеть и много общего между записью математической формулы и соответствующего арифметического выражения на фортране. В обоих случаях операция возведения в степень является старшей по отношению к остальным арифметическим операциям, а операции умножения и деления являются старшими по отношению к операциям сложения и вычитания. Скобки играют в фортранной записи ту же роль, что и в записи математической формулы, т. е. они служат для указания порядка, в котором должны вычисляться отдельные части арифметического выражения.

П р и м е р ы. Выражение $A/B \cdot C$ соответствует формуле $\frac{a}{b} \cdot c$, но не $\frac{a}{bc}$.

Выражение $A * X + B / C * X + D$ соответствует математической записи $ax + \frac{b}{c}x + d$, но не $\frac{ax+b}{cx+d}$.

Формулу $\frac{x}{-y}$ нельзя записывать как $X / -Y$, но можно как $X / (-Y)$, либо как $-X / Y$, либо как $(X) / (-Y)$.

Сформулируем теперь основные

Правила записи арифметических выражений

1. В состав арифметических выражений могут входить константы, переменные и функции вещественного, целого, комплексного типов и с двойной точностью. В арифметических выражениях запрещается использовать величины логического типа.

2. Константы, переменные и функции рассмотренных типов являются частными случаями арифметических выражений.

3. Если A — арифметическое выражение, то (A) , $((A))$ и т. д. также являются арифметическими выражениями.

4. В арифметических выражениях используются знаки арифметических действий $+$, $-$, $*$, $/$, $**$ и круглые скобки. Использовать иные знаки действий (например, знаки логических действий) запрещается.

5. Нельзя ставить подряд два знака арифметических действий.

6. После правой скобки, если она не является последней, должны стоять либо правая скобка, либо знак арифметического действия. Между константой и левой скобкой необходим знак арифметического действия.

Поясним последнее правило п р и м е р а м и.

В записи $(A * X + B)C$ после правой скобки пропущен знак арифметического действия. То же самое можно сказать про записи

$5.(A+B)$, $(A * T + 3.)$ $(\text{SIN}(X) + 1.)$, $(A - C) 3.2$.

Во всех указанных случаях транслятор вставит на место пропущенного знака действия знак умножения, отпечатает диагностический текст и продолжит дальнейшую трансляцию арифметического выражения. Нарушение правил 1, 4, 5 препятствует выходу задачи на счет.

Вычисление значения арифметического выражения производится слева направо с учетом старшинства арифметических операций и круглых скобок. При этом, как уже говорилось, самой старшей является операция возведения в степень, далее идут умножение и деление, имеющие одинаковое старшинство, и, наконец, сложение и вычитание, также одинакового старшинства.

Отметим редко встречающийся случай записи вида

$$A ** B ** C$$

выполняемой как

$$(A ** B) ** C$$

Рассмотрим теперь важный вопрос о том, величины каких типов (из числа допустимых) можно использовать в арифметических выражениях.

Для удобства дальнейшего изложения введем обозначения: R — вещественная величина, I — целая величина, C — комплексная величина, DP — величина с двойной точностью и знак —, обозначающий недопустимую комбинацию. Табл. 3 и 4 показывают, какие величины могут быть объединены с помощью знаков арифметических действий в допустимые арифметические выражения, а также тип получаемого результата (он указан на пересечении соответствующих строки и столбца).

Таблица 3

Для операций +, —, *, /.

	R	I	C	DP
R	R	R	C	DP
I	R	I	C	DP
C	C	C	C	—
DP	DP	DP	—	DP

Таблица 4

Для операции X ** Y

X \ Y				
	R	I	C	DP
R	R	R	—	DP
I	R	I	—	DP
C	—	C	—	—
DP	DP	DP	—	DP

Сформулируем некоторые полезные правила, вытекающие из приведенных таблиц.

1. В одном арифметическом выражении не могут участвовать величины комплексные и с двойной точностью.

2. Показатель степени не может быть комплексной величиной.

3. Комплексные величины можно возводить только в целую степень.

4. Тип результата арифметической операции определяется по принципу старшинства типов величин, участвующих в данной операции. При этом тип вещественный считается старше типа целого, а типы комплексный и с двойной точностью — старше типа вещественного.

5. При делении двух целых величин также получается величина целого типа. При этом дробная часть частного отбрасывается. Например, вычисляя выражение $5 * (19/4)$, получим в результате 20, потому что после деления целого числа 19 на целое число 4 получим целое число 4 (а не 4,75, как в обычной арифметике).

Дадим пояснения к выполнению операции возведения в степень. Если показатель степени является целой величиной, то возведение в степень выполняется как последовательность умножений (с переходом к обратной величине, если показатель отрицательный). Если же показатель степени является величиной вещественной или с двойной точностью, то операция $X ** Y$ выполняется как $EXP(Y * ALOG(X))$ для вещественного показателя или как $DEXP(Y * ALOG(X))$ для показателя с двойной точностью. Здесь предполагается, что X вещественный.

Отсюда следует, что возведение в целую степень может выполняться при любых значениях основания степени (но если показатель степени отрицательный, то основание степени не может равняться нулю). Возведение же в степень вещественного типа или с двойной точностью может выполняться лишь для положительных значений основания степени.

Таким образом, при возведении в целую (в математическом смысле) степень показатель ее лучше записывать как величину целого типа.

Отметим, что в тех случаях, когда возведение в целую или вещественную степень не проходит из-за превышения диапазона представимых чисел, необходимо основание или показатель степени представить как величину с двойной точностью. Например, операция

12.5 ** 20 на БЭСМ-6 приведет к переполнению. Однако, записав ее в виде 12.5D0 ** 20 или 12.5 ** 20.D0, получим правильный результат (первая из этих записей предпочтительнее).

Отметим также, что извлечение квадратного корня лучше записывать как SQRT(X), чем X ** 0.5, или X ** (1./2.). Извлечение корня с помощью функции SQRT на БЭСМ-6 выполняется примерно в 2,5 раза быстрее, чем возведение в степень 0.5, и первый способ обеспечивает большую точность.

У п р а ж н е н и я. 4. Записать в виде арифметических выражений следующие формулы:

а) $abc + 3 \sin x + e^{-x} - \frac{1}{ab + cd},$

б) $a + \frac{ad - bc}{a^2 + b^2 + c^2 + d^2},$

в) $|ax^2 + bx + c| - \frac{a^2}{|x + ax^2|},$

г) $3x \sin^2 x - 5y \cos^3 y.$

5. Указать математические формулы, соответствующие следующим фортранным арифметическим выражениям

- а) A * SIN (X) ** 2 + 5. / (2. + 3. * SQRT (X ** 2 + Y ** 2))
- б) 3. * EXP (X) + ALOG (2. * COS (X) ** 4 + 3. * X ** (1./3.))
- в) ABS (X ** 3 + 2. * SIN(X) - SQRT (X - 2.)).

6. Определить, являются ли допустимыми для арифметических выражений приведенные ниже записи. В случае допустимой записи определить тип результата. Тип переменной всюду определяется по первой букве ее идентификатора.

- а) AX1 * 2. 3E6 + (1., 2.5) ** 3
- б) -C + I / (J + 1) ** (2. + AFT)
- в) 2.3D - 5 + (1.5, -7.) ** R
- г) (ACT + PSI) ** (1.2 + (1.3, -2.))
- д) A + CSQRT (1.2, 3.5) * 2.7D0
- е) -1.E6 + 2.5D125 / I + 2.6 ** 12.D2
- ж) SIN (1.5, -2.) + DSQRT (2.)
- з) (1.5, 7.) ** T + 1.E6 / SIN (X)
- и) AC2 + 35. * DEXP (C + 1.D - 1) + ATAN (T)
- к) ALOG (2.76 / T) + C ** 2 / SQRT (F1) / I5
- л) I + 3 * (K + 1)

§ 16. Логические выражения и выражения отношения

Наряду с арифметическими выражениями в фортрапе используются логические выражения и выражения отношения.

Логические выражения по структуре аналогичны арифметическим выражениям, т. е. они включают в себя константы, переменные и функции, знаки действий и круглые скобки. Однако, в отличие от арифметических выражений, в логических выражениях допускаются лишь логические величины и знаки логических действий. Знаками логических действий являются:

.NOT. — логическое отрицание,

.AND. — логическое умножение,

.OR. — логическое сложение.

Значением логического выражения может быть лишь «истина» или «ложь». Логические выражения чаще всего используются в логических условных операторах IF (см. § 18).

Результаты логических операций определяются следующим образом:

.NOT.L есть «ложь», если L — «истина»,

L1.AND. L2 есть «истина», только если выражения L1 и L2 истинны,

L1.OR. L2 есть «ложь», только если L1 и L2 ложны. Иначе говоря, указанные логические операции эквивалентны операциям логического отрицания, логического умножения и логического сложения, принятым в математической логике.

Старшинство логических операций определяется следующим образом: .NOT., .AND., .OR., т. е. операция .NOT. является самой старшей, а операция .OR. — самой младшей.

Основные правила записи логических выражений

1. Логическая константа, логическая переменная, логическая функция и выражение отношения (см. ниже) являются логическими выражениями.

2. Если L — логическое выражение, то (L) , $((L))$
и т. д. также являются логическими выражениями.
3. Выражения вида

$$\begin{array}{ll} L1.AND..AND.L2, & L1.OR..AND.L2, \\ L1.AND..OR.L2, & L1.OR..OR.L2 \end{array}$$

недопустимы, т. е. не допускается двух и более подряд стоящих операций $.AND.$ или $.OR.$

4. Операция $.NOT.$ может встречаться в комбинациях с $.AND.$ или $.OR.$ только таким образом:

$$\begin{array}{ll} .AND..NOT. & .AND. (.NOT. L) \\ .OR..NOT. & .OR. (.NOT. L) \end{array}$$

где L — логическое выражение.

5. Многократное повторение операции $.NOT.$ допускается только в записи вида

$$.NOT. (.NOT. ...(.NOT. L)...) .$$

6. Вычисление значения логического выражения производится слева направо с учетом старшинства логических операций и круглых скобок.

Нарушение правил 3—5 диагностируется при трансляции.

Перейдем к описанию выражений отношения.

Выражение отношения имеет форму

$$q_1 \text{ оп } q_2,$$

где q_1 и q_2 — арифметические выражения, а оп — один из следующих знаков операций:

$$\begin{array}{ll} .EQ. & \text{— равно,} \\ .NE. & \text{— не равно,} \\ .GT. & \text{— больше,} \end{array} \quad \begin{array}{ll} .GE. & \text{— больше или равно,} \\ .LT. & \text{— меньше,} \\ .LE. & \text{— меньше или равно.} \end{array}$$

Значение выражения отношения есть «истина», если значения арифметических выражений q_1 и q_2 удовлетворяют условию, налагаемому данной операцией отношения.

Например, выражение $1.LE.2$ всегда истинно, а выражение $X ** 2 + Y ** 2.LT.0$ всегда ложно (X и Y — не комплексные величины).

Сформулируем

Правила записи и вычисления выражений отношения

1. Выражения вида

$$q_1 \text{ on } q_2 \text{ on } q_3$$

не допустимы.

2. Допускаются выражения вида

$$q_1 \text{ on } q_2 \text{ .AND. } q_3 \text{ on } q_4$$

$$q_1 \text{ on } q_2 \text{ .OR. } q_3 \text{ on } q_4$$

3. В состав выражений отношения не могут входить комплексные арифметические выражения.

4. Вычисление выражений отношения производится слева направо. При этом записи вида

$$q_1 \text{ on } q_2, q_1 \text{ on } (q_2), (q_1) \text{ on } q_2, (q_1) \text{ on } (q_2)$$

эквивалентны.

5. При вычислении выражений отношения операции выполняются в порядке старшинства таким образом: сначала арифметические операции, затем операции отношения и затем логические операции. Например, запись

$$A+2*B:GT.0..AND.X+Y.LT.5.$$

не требует круглых скобок.

П р и м е р ы. 1. Выражение отношения

$$(X.GE.0..AND.X.LE.1.).AND.(Y.GE.0..AND.Y.LE.1.)$$

истинно, если точка с координатами (X, Y) принадлежит единичному квадрату с вершинами в точках (0,0), (0,1), (1,0), (1,1), и ложно в противном случае.

2. Выражение отношения

$$(X ** 2 + Y ** 2).LT.4$$

истинно для всех точек (X, Y), лежащих внутри круга радиуса 2 с центром в начале координат.

Приведенные примеры показывают удобство использования выражений отношения для проверки различных условий, налагаемых на значения арифметических выражений. Выражения отношения чаще всего применяются в логических условных операторах IF (см. § 18), позволяющих производить разветвления в зависимости от истинности или ложности значений логических выражений.

Операции отношения заменяются транслятором на вычитание с последующим сравнением разности с нулем.

Например, при вычислении значения выражения

$$ABS(X - Y).GT.1.$$

производится вычисление величины

$$ABS(X - Y) - 1.$$

и если ее значение больше нуля, то исходному выражению приписывается значение «истина»,

Исключение составляют операции .EQ. и .NE. для целых величин. В этих случаях производится не вычитание, а поразрядное сравнение соответствующих машинных слов. Целые величины считаются равными (т. е. операция .EQ. дает значение «истина»), если сравниваемые машинные слова совпадают во всех 48 разрядах. Это позволяет сравнивать между собой текстовые и восьмеричные величины.

3. Выражение

$$3HEND.EQ.4HEND*$$

ложно, так как не все разряды машинных представлений текстовых констант, стоящих в левой и правой части, совпадают.

4. Выражение

$$1.EQ.1.$$

истинно (разность $1 - 1$. равна 0). Однако те же константы 1 и 1., записанные как восьмеричные, уже не совпадают, т. е. выражение

$$6400\ 0000\ 0000\ 0001B.EQ.\ 4050\ 0000\ 0000\ 0000\ B$$

ложно.

В заключение рассмотрим несколько записей, недопустимых в качестве выражений отношения.

5. Запись

$$A.LT.(B.AND.C)$$

недопустима, так как (B.AND.C) есть логическое выражение, которое не может сравниваться по операции отношения.

6. Запись

$$A.AND..NOT.B.GT.C$$

недопустима, так как .NOT. B есть логическое выражение.

У п р а ж н е н и е 7. Определить, являются ли допустимыми приведенные ниже записи. Тип переменной всюду определяется первой буквой ее идентификатора,

- а) A.EQ.B.AND.C.NE..TRUE,
- б) A.OR.1..GT.B
- в) A ** 2+B ** 2.GT.C+1,
- г) 3HABC.EQ.N1
- д) L.AND.M.GT.C
- е) ABS (C)+SIN (X+1.).GT.B15.AND.F
- ж) A.GT.5..AND. (B.LT.3.)
- з) (X+Y).GT. (AC ** 2+F).AND. (B.EQ.1.)

§ 17. Оператор присваивания

В процессе выполнения подпрограммы происходит вычисление значений различных арифметических и логических выражений, а также выражений отношения. Значение каждого такого выражения может быть записано в определенное место машинной памяти. Для записи значения выражения в память машины служит специальный оператор, именуемый *оператором присваивания*. Общий вид записи этого оператора

$$A = E$$

Здесь A — переменная (простая или с индексами)
 E — выражение, $=$ — знак присваивания.

В зависимости от типа выражения E различают арифметический и логический операторы присваивания. В правой части *арифметического оператора* присваивания может стоять любое арифметическое выражение и, в частности, константа, переменная или функция любого типа, кроме логического. Слева же может стоять лишь переменная любого типа, кроме логического.

Приведем примеры.

$$A=2.+X ** 2$$

$$Z2T=ABS (X ** 2-Y ** 2)+SIN (FI)$$

$$A=1.$$

$$C=A$$

$$A=A+1.$$

$$A=A ** 2+1./A$$

Запись $A = A+1.$ означает, что к текущему значению переменной A , хранящемуся в некоторой ячейке

памяти, прибавляется константа 1. и результат помещается в ту же ячейку памяти. Тем самым текущее значение переменной величины А увеличилось на 1.

Оператор $A = A ** 2 + 1./A$ присваивает переменной А новое значение, полученное из прежнего ее значения в результате вычисления арифметического выражения, написанного справа.

Таким образом, оператор присваивания, внешне напоминающий запись математического равенства, вызывает определенное действие, состоящее в том, что значение выражения, стоящего справа, записывается в определенное место машинной памяти, предназначенное для хранения текущего значения некоторой переменной величины, стоящей слева от знака $=$. При этом прежнее значение переменной величины «стирается».

Рассмотрим теперь вопрос о преобразовании типа результата в случаях, когда тип переменной в левой части арифметического оператора присваивания не совпадает с типом арифметического выражения в правой его части.

Таблица показывает, как происходит преобразование типа арифметического выражения в процессе выполнения арифметического оператора присваивания $A = E$.

$\begin{array}{c} \backslash \\ A \end{array} \begin{array}{c} E \end{array}$	R	I	C	DP
R	R	R	R	R
I	I	I	I	I
C	C	C	C	—
DP	DP	DP	—	DP

П р и м е р ы

1. $I = R$ Здесь производится вычисление целой части R с последующим преобразованием ее в величину целого типа (денормализация) и запись в ячейку памяти, отведенную для переменной I.

2. $R = DP$ Значение первого машинного слова переменной DP записывается в ячейку памяти, отведенную для R, при условии, что DP не выходит за диапазон

значений, допустимых для величин вещественного типа. В противном случае преобразование невозможно и выдается сигнал об ошибке. Например, присваивание $R = 1.D40$ невозможно, так как значение константы в правой части выходит за диапазон, допустимый для вещественных величин.

3. $C = R$ Значение R присваивается действительной части C , мнимая часть C полагается равной нулю.

Например, в результате выполнения оператора $C = 1.E3$ комплексная величина C получит значение, равное комплексной константе $(1.E3,0.)$.

4. $DP = R$ Значение R присваивается первому машинному слову DP (главной части). Второе машинное слово полагается равным нулю.

5. $R = I$ Производится преобразование целой величины в вещественную (нормализация) с последующей записью ее в ячейку памяти, отведенную для R .

В том случае, когда тип переменной в левой части совпадает с типом арифметического выражения, происходит лишь запись значения арифметического выражения в нужное место памяти.

Отметим, что многие начинающие пользователи не усматривают большого различия между записями вида

$$A=1. \text{ и } A=1$$

предпочитая вторую из них первой (из-за того, что не надо ставить «лишнюю» точку). В результате, конечно, все получается правильно, т. е. переменной A будет присвоено значение вещественной константы 1., однако на БЭСМ-6 время выполнения второго оператора будет примерно в 10 раз превышать время выполнения первого оператора. Привыкая к тому, что транслятор все равно «исправит» запись вида $A = 1$, такой пользователь почти наверняка напишет аналогичную вещь в операторе DATA (см. § 25), в котором транслятор уже не будет ничего «исправлять» (!).

Таким образом, если переменной присваивается значение константы, то тип константы лучше выбирать тот же самый, что и у переменной в левой части оператора присваивания.

Логический оператор присваивания используется значительно реже арифметического оператора присваи-

вания. Здесь не возникает вопроса о преобразовании типа результата, так как в левой части логического оператора присваивания допускаются лишь переменные логического типа, а в правой части могут быть лишь логическое выражение или выражение отношения, принимающие значения также логического типа.

У п р а ж н е н и е 8. Определить, какие из приведенных записей являются допустимыми для операторов присваивания. Во всех записях тип переменной определяется по первой букве ее идентификатора.

- а) $A = I * T + (\sin(A * X + B) + \text{TRUE.})$
- б) $F = 1.D7 / (A + I ** 0.3)$
- в) $FI ** 2 = \text{PSI} ** 2 + \text{THETA} ** 2$
- г) $\sin(X) = A * \cos(FI)$
- д) $1.E7 = A + C ** 2 / \sin(\text{ALFA} + \text{BETA})$
- е) $\text{PRINT} = \text{GO TO} * \text{IF}$

§ 18. Операторы условного перехода. Операторы GO TO и CONTINUE

Прежде чем приступить к описанию указанных в заголовке операторов, рассмотрим две задачи.

З а д а ч а 1. Написать программу вычисления величины

$$\text{sgn } x = \begin{cases} -1, & \text{если } x < 0, \\ 0, & \text{если } x = 0, \\ 1, & \text{если } x > 0. \end{cases}$$

Специфика данной задачи состоит в том, что в зависимости от значения x должен быть выполнен один и только один из следующих трех операторов присваивания:

$$\text{SGNX} = -1. \quad \text{SGNX} = 0. \quad \text{SGNX} = 1.$$

Таким образом, сначала необходимо, в зависимости от значения x , определить, который из этих трех операторов надо выполнить. Затем надо обойти выполнение двух остальных операторов присваивания. Напомним, что присваивание переменной величине нового значения «стирает» ее старое значение. Мы запишем

сначала одно из возможных решений задачи 1, а затем дадим пояснения *).

IF (X) 5, 8, 13	GO TO 15
5 SGNX=-1.	13 SGNX=1.
GO TO 15	15 CONTINUE
8 SGNX=0.	

Оператор IF производит условный переход, или условную передачу управления по следующему правилу:

1) если $X < 0$, то происходит переход к выполнению оператора, метка которого указана первой (в нашем случае это оператор с меткой 5);

2) если $X = 0$, то происходит переход к выполнению оператора, метка которого указана второй (в нашем случае это оператор с меткой 8);

3) если $X > 0$, то происходит переход к оператору с меткой, которая указана последней в списке меток (в нашем случае это оператор с меткой 13).

Итак, после выполнения оператора IF произойдет переход к выполнению одного из трех операторов, метки которых написаны в данном операторе IF. Этот переход будет зависеть от некоторого условия, а именно от значения величины X , написанной в скобках. Проследим за дальнейшим ходом вычислений. Пусть $X < 0$ и, следовательно, мы попали на оператор с меткой 5. После его выполнения переменной SGNX будет присвоено нужное значение, а именно -1 . Если теперь не предпринять никаких мер, то подпрограмма будет выполняться дальше и придет к оператору с меткой 8, написанному ниже. Но этот оператор не должен выполняться, так как он «испортит» верное значение переменной SGNX. Для того чтобы это предотвратить, и написан оператор

GO TO 15

который осуществляет безусловный переход на оператор с меткой 15, обходя тем самым те операторы, которые не должны быть выполнены. В случае, когда $X = 0$, будет выполнен оператор с меткой 8, после чего произойдет переход к оператору с меткой 15, обходя оператор с меткой 13. В случае $X > 0$ оператор IF

*) Здесь и далее операторы печатаются в два столбца.

передаст управление на оператор с меткой 13, после выполнения которого будет выполнен оператор с меткой 15.

Оператор с меткой 15 должен обозначать продолжение выполнения подпрограммы. Иначе говоря, это должен быть оператор, который не выполнял бы никаких действий, но к которому можно было бы приписать метку. Таким оператором, который не выполняет никаких действий, а служит лишь носителем метки, и является оператор

CONTINUE

Рассмотренный пример иллюстрирует применение оператора IF для организации разветвлений. При этом число ветвей в программе не превосходит трех.

Итак, в процессе решения задачи 1 мы познакомились с тремя новыми операторами фортрана: IF, GO TO, CONTINUE. С их помощью мы решим следующую задачу.

З а д а ч а 2. Вычислить сумму

$$S = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{1000\,000}.$$

На первый взгляд кажется, что для решения этой задачи не требуется никаких операторов, кроме оператора присваивания. Достаточно написать всего лишь один оператор

$$S = 1. + 1./2. + 1./3. + \dots + 1./1000000.$$

Здесь многоточие означает, что в записи оператора отсутствуют 999996 слагаемых, которые в реальной программе должны быть указаны. Несложный расчет показывает, что для записи этого оператора потребуется около 10^7 символов. Ясно, что такое «тривиальное» решение задачи не годится. Посмотрим, как можно представить процесс вычисления суммы S . Для этого надо вычислить очередное слагаемое и прибавить его к имеющейся неполной сумме. Затем надо вычислить следующее слагаемое и снова прибавить его к текущему значению суммы и т. д. Процесс вычисления суммы будет закончен после прибавления последнего, миллионного по счету слагаемого. Запишем рассмотренный

вычислительный процесс

$S=0.$

$A=0.$

1 $A=A+1.$

$S=S+1./A$

IF ($A-1000000.$) 1, 2, 2

2 CONTINUE

Как видим, сначала переменным S и A присваиваются исходные (в данном случае нулевые) значения. В дальнейшем S будет означать текущее значение суммы, а A — текущее значение знаменателя, равное номеру соответствующего слагаемого. В процессе вычисления суммы будет происходить увеличение текущего номера слагаемого на 1 с последующим добавлением соответствующего слагаемого к текущему значению суммы. После добавления очередного слагаемого производится проверка окончания процесса вычисления суммы с помощью оператора IF.

Обратим внимание на важную особенность написанной последовательности операторов. Здесь происходит возврат к выполнению одних и тех же операторов, пока значение переменной A остается меньше 1000000. Участок подпрограммы, состоящий из трех операторов, выполняется миллион раз. Такие многократно повторяющиеся последовательности операторов называются циклами. Итак, *цикл* — это группа операторов, которые выполняются многократно путем возврата от последнего оператора этой группы к ее первому оператору.

Дадим теперь более точное описание введенных в этом параграфе операторов.

1. Арифметический оператор IF имеет вид

IF (A) n_1, n_2, n_3

Здесь A — арифметическое выражение (не комплексное), а n_1, n_2, n_3 — метки операторов. Арифметический оператор IF вычисляет значение арифметического выражения A , а затем передает управление согласно следующим правилам:

если $A < 0$, то — оператору с меткой n_1 ;

если $A = 0$, то — оператору с меткой n_2 ;

если $A > 0$, то — оператору с меткой n_3 .

Среди меток n_1 , n_2 и n_3 могут быть одинаковые. В дальнейшем вместо «арифметический оператор IF» мы часто будем кратко говорить «оператор IF». Приведем примеры операторов IF.

а) IF (B ** 2 — 4. * A * C) 12, 7, 18

б) IF (3.D2+DSIN(X * 2. * 3.14159D0)) 101, 15, 15

в) IF ((1.+CABS(X + (1.3, 2.5)))/12.6D4) 9, 11, 12

Отметим, что на БЭСМ-6 в случае использования в операторе IF комплексного арифметического выражения происходит проверка его действительной части согласно правилам выполнения этого оператора. При этом никакой диагностики не выдается.

2. Оператор перехода GO TO. Этот оператор имеет вид

GO TO n

Здесь n — метка оператора, на который происходит переход (или передача управления). Другими словами, после выполнения оператора GO TO будет выполняться оператор с меткой n .

3. Оператор CONTINUE. Этот оператор не выполняет никаких действий и служит лишь носителем метки. Запрещается записывать CONTINUE без метки. Допускается запись подряд нескольких операторов CONTINUE с разными метками. В этом случае их метки считаются эквивалентными, т. е. переход на любую из них означает одно и то же. Исключение составляют те операторы CONTINUE, которыми оканчиваются циклы DO (см. § 20).

4. Логический оператор IF. Рассмотрим теперь другую разновидность оператора IF, так называемый логический оператор IF, или условный логический оператор перехода. Он имеет вид

IF (L) S

Здесь L — логическое выражение, а S — любой оператор, кроме DO (см. § 20) или другого логического IF. Выполнение данного оператора производится следующим образом:

если L истинно, то выполняется оператор S;

если L ложно, то выполняется оператор, следующий за этим логическим оператором IF.

Проиллюстрируем применение логического оператора IF на решении двух предыдущих задач.

Решение первой задачи:

SGNX = -1.		IF (X.EQ.0.) GO TO 15
IF (X.LT.0.) GO TO 15		SGNX = 1.
SGNX = 0.		15 CONTINUE

А вот более краткое решение этой задачи:

```
SGNX = -1.
IF (X.EQ.0.) SGNX = 0.
IF (X.GT.0.) SGNF = 1.
```

Решение второй задачи с помощью логического IF можно представить так:

```
S = 0.
A = 0.
1 A = A + 1.
  S = S + 1./A
  IF (A.LT.1.E6) GO TO 1
```

Как видим, запись решения обеих задач с помощью логического оператора IF получилась короче и проще, чем с помощью арифметического оператора IF. Однако такое решение не всегда самое экономное.

Приведем примеры логических операторов IF.

- а) IF (A.EQ.B.AND.X ** 2 + Y ** 2.GT.3) C = 1.
- б) IF (Z + SIN (X + 5.).GT.1..AND.T.LT.2.) GOTO10
- в) IF (A.GT.(1.5D20 - DEXP (1.7D3))) A = A + 1.

Как видно из примеров, в логическом операторе IF чаще всего используются выражения отношения.

У п р а ж н е н и я. Вычислить значения следующих вещественных величин, используя арифметический или логический оператор IF. Значения вещественных величин x и y предполагаются заданными.

$$9. z = \begin{cases} \sqrt{x-2}, & \text{если } x \geq 2, \\ 1/x + \sin(x+3), & \text{если } x < 2 \text{ и } x \neq 0, \\ 0, & \text{если } x = 0. \end{cases}$$

$$10. t = \begin{cases} \sqrt{x^2 - 2x + 3}, & \text{если } x^2 - 2x + 3 \geq 0, \\ \frac{1}{x^2 + 1}, & \text{если } x^2 - 2x + 3 < 0. \end{cases}$$

$$11. S = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots + \frac{1}{99999}.$$

$$12. w = 1 + \frac{1}{2} + \frac{1}{3} - \frac{1}{4} - \frac{1}{5} + \frac{1}{6} + \frac{1}{7} - \frac{1}{8} - \frac{1}{9} + \dots \\ \dots \pm \frac{1}{100000}.$$

Здесь вместо \pm надо поставить правильный знак.

$$13. f = \begin{cases} \max(x, y + 5), & \text{если } x > y, \\ \min(x + 1, y^2, 3), & \text{если } x \leq y. \end{cases}$$

§ 19. Операторы перехода: вычисляемый GO TO и GO TO по предписанию. Оператор ASSIGN

Мы уже познакомились с двумя видами условных операторов IF и простейшим оператором перехода GO TO. Эти операторы позволяют управлять порядком выполнения операторов в подпрограмме. В частности, они позволяют выполнять разветвления и организовывать циклы.

Теперь мы познакомимся еще с двумя разновидностями операторов перехода, которые предназначены в основном для организации разветвлений. Как мы уже знаем, разветвление в подпрограмме может быть выполнено посредством арифметического или логического оператора IF. В первом случае число ветвей не превосходит трех, во втором случае — двух. Но иногда возникает необходимость организации разветвления более чем на 3 ветви. Такое разветвление может быть организовано посредством нескольких операторов IF (арифметических или логических). Однако для организации таких разветвлений в фортране предусмотрен более удобный оператор, который называется вычисляемый GO TO.

Вычисляемый GO TO имеет вид

GO TO (n_1, n_2, \dots, n_m), i

Здесь n_1, n_2, \dots, n_m — метки операторов, i — простая целая переменная (переключатель). Вычисляемый GO TO осуществляет переход на оператор с меткой n_i . Таким образом, значение переключателя i определяет порядковый номер метки в списке, заключенном в скобки ($1 \leq i \leq m$).

П р и м е р.

GO TO (15, 18, 102, 7, 18, 3), ISW

Этот оператор осуществляет переход на оператор с меткой 15, если ISW=1, 18, если ISW=2, ..., 3, если ISW=6

Значение переключателя i должно быть определено до его использования в данном операторе GO TO. При этом должно выполняться условие

$$1 \leq i \leq m,$$

где m — число меток в списке оператора GO TO. В противном случае правильная работа программы не гарантируется, хотя диагностика и не выдается.

Как показывает практика, наиболее частая ошибка при использовании вычисляемого GO TO состоит в том, что значение переключателя i оказывается равным 0. Это чаще всего обусловлено тем, что отсутствует оператор, определяющий значение переключателя i . В этом случае происходит заикливание программы, т. е. программа не может выйти из цикла. Отметим, что заикливание не может быть обнаружено средствами машины.

Итак, вычисляемый GO TO обычно применяется для организации разветвлений в случаях, когда число ветвей более 3. Пусть, например, требуется вычислить значение величины f , определяемой по следующему правилу:

$$f = \begin{cases} 1, & \text{если } n=1, \\ n+5, & \text{если } n=2, \\ n^2+2n+4, & \text{если } n=3, \\ -1, & \text{если } n=4, \\ 1, & \text{если } n=5. \end{cases}$$

Требуемое вычисление можно осуществить, например, следующим образом:

21 GO TO (21, 2, 35, 17, 21), N F=1. GO TO 30 2 F=N+5. GO TO 30	35 F=N ** 2+ * 2. * N+4. GO TO 30 17 F=-1. 30 CONTINUE
---	--

Предполагается, что значение N уже определено и что условие $1 \leq N \leq 5$ выполнено.

Как видно из последнего примера, вычисляемый GO TO производит разветвление, а каждая ветвь оканчивается оператором безусловного перехода GO TO, производящим выход на основную ветвь программы.

Перейдем к рассмотрению оператора GO TO по предписанию. Этот оператор имеет вид

GO TO $m, (n_1, n_2, \dots, n_k)$

Здесь n_1, n_2, \dots, n_k — метки операторов, а m — простая целая переменная, являющаяся переключателем. Данный оператор осуществляет передачу управления оператору с меткой $n_i = m$. Таким образом, здесь переключатель m в явном виде определяет метку того оператора, на который происходит передача управления.

Для определения значения переключателя m служит специальный оператор, иногда называемый *оператором присваивания метки*. Этот оператор имеет вид

ASSIGN s TO m

Здесь s — одна из меток, на которую будет сделан переход в операторе GO TO по предписанию; m — переключатель, являющийся простой целой переменной.

П р и м е р.

GO TO MET, (16, 36, 7, 55)

Данный оператор осуществляет переход по одной из меток, указанных в списке. Значение этой метки должно быть заранее определено оператором ASSIGN, например,

ASSIGN 7 TO MET

В этом случае будет осуществлен переход на оператор с меткой 7.

Обращаем внимание читателя на то, что оператор ASSIGN служит лишь для определения значений целых переменных, предназначенных исключительно для использования в операторах GO TO по предписанию.

Оператор GO TO по предписанию можно использовать для организации разветвлений подобно вычисляемому GO TO. Этот оператор удобно применять для организации возврата на нужный участок подпрограммы после выполнения одной из ветвей подпрограммы. Такая ситуация возникает, например, в случаях, когда некоторый участок подпрограммы желательно использовать в нескольких местах этой подпрограммы.

Пусть, например, нам необходимо вычислить величину $\operatorname{sgn} x$ (см. § 18) в трех разных местах подпрограммы. Это можно сделать, например, так:

```
      ASSIGN 15 TO L
      GO TO 7
35 CONTINUE
      ASSIGN 83 TO L
      GO TO 7
83 CONTINUE
      ASSIGN 42 TO L
      GO TO 7
42 CONTINUE
7 CONTINUE
      SGNX = -1.
      IF (X.EQ.0.) SGNX = 0.
      IF (X.GT.0.) SGNX = 1.
      GO TO L, (15, 83, 42)
```

Здесь многоточия означают, что на этом месте могут быть расположены некоторые операторы. Обратим внимание на организацию вычислительного процесса в рассмотренном примере. Сначала с помощью опера-

тора ASSIGN определяется метка, на которую нужно будет возвратиться. Затем происходит безусловный переход на нужный участок подпрограммы, который заканчивается оператором GO TO по предписанию. Таким образом, сначала обеспечивается возврат на нужную метку, а затем производится безусловный переход на участок подпрограммы, возврат из которого уже заранее предписан.

Отметим важное правило, состоящее в том, что переключатель m в операторе GO TO по предписанию не может быть формальным параметром (см. § 26).

Если переключатель m будет определен не оператором ASSIGN, а, например, оператором присваивания, то никакой диагностики при трансляции выдано не будет. Однако при счете это обычно приводит к потере управления с «порчей» памяти и в конце концов к аварийному останову.

У п р а ж н е н и е 14. Определить ошибки, допущенные в приводимых ниже примерах. Указать наиболее вероятные последствия этих ошибок. Предполагается, что все нужные метки в подпрограмме имеются.

- а) $N=6$
GO TO (15, 18, 15, 7), N
- б) $LM=20$
GO TO LM, (20, 18, 36, 1)
- в) ASSIGN 16 TO ISW
GO TO (16, 15, 38, 22), ISW
- г) ASSIGN 7 TO JT
GO TO JT, (8, 11, 17, 35, 17)

§ 20. Оператор DO

В § 18 было рассмотрено понятие цикла и выяснено его важное значение при составлении подпрограмм. Было выяснено также, что для организации циклов могут быть использованы операторы IF и оператор GO TO. Однако в фортране предусмотрен более удобный способ организации циклов с помощью специального оператора DO.

Оператор DO имеет вид

$$DO\ n\ i = m_1, m_2, m_3$$

Здесь n — метка последнего оператора цикла; i — простая целая переменная, называемая *переменной* (или *параметром*) *цикла*; m_1 — начальное значение, присваиваемое i ; m_2 — наибольшее значение, которое может принимать переменная i ; m_3 — величина, прибавляемая к i после каждого «оборота» цикла (*шаг*). Величины m_1 , m_2 и m_3 могут быть целыми константами без знака или простыми целыми переменными и принимать значения, большие 0.

Если m_3 опущено, то оно считается равным 1, т. е. запись вида

$$\text{DO } n \ i = m_1, \ m_2$$

эквивалентна записи

$$\text{DO } n \ i = m_1, \ m_2, \ 1$$

Оператор DO, оператор с меткой n и все операторы между ними составляют цикл DO. В некоторых случаях используется форма записи циклов без DO (см. §§ 25 и 28).

Операторы цикла DO выполняются сначала при значении i , равном m_1 , затем при $i = m_1 + m_3$, затем при $i = m_1 + 2m_3$ и т. д. до тех пор, пока $i \leq m_2$.

Операторы цикла должны выполняться хотя бы один раз, т. е. должно быть $m_1 \leq m_2$. После выполнения цикла DO значение переменной i не определено.

Пример 1. Написать программу вычисления суммы

$$S = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{1000000}$$

с помощью цикла DO.

Решение.

```

S=0.
N=1 000 000
DO 1 I=1, N
1 S=S+1./I

```

Здесь цикл состоит всего из двух операторов. Сначала цикл выполнится при $I=1$, затем при $I=2$, $I=3$ и т. д. до тех пор, пока значение I не достигнет $N=1000000$. По окончании цикла управление будет передано опера-

тору, написанному после оператора с меткой 1. Обратим внимание читателя на то, что запись вида

DO 1 I=1, 1000 000

на БЭСМ-6 не прошла бы, так как константы, участвующие в операторе DO, на БЭСМ-6 не могут превосходить $2^{15}=1$, т. е. 32767. На других ЭВМ также имеются подобные ограничения, свои для каждой машины.

Пример 2. Написать программу вычисления суммы

$$S = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{1000} + \frac{1}{2001} + \frac{1}{2002} + \dots + \\ + \frac{1}{3000} + \frac{1}{4001} + \frac{1}{4002} + \dots + \frac{1}{5000} + \dots + \frac{1}{100001} + \\ + \frac{1}{100002} + \dots + \frac{1}{101000}.$$

Решение. Эта сумма состоит из 51 группы слагаемых, по 1000 слагаемых в каждой группе. Каждая группа начинается со слагаемого вида $1/[2(n-1) \cdot 1000 + 1]$, а оканчивается слагаемым вида $1/[(2n-1) \cdot 1000]$, где $n=1, 2, \dots, 51$. Для решения надо, очевидно, повторить 51 раз цикл вычисления суммы из 1000 слагаемых. Таким образом, придется написать два цикла: один — внешний — на 51 оборот и другой — внутренний — на 1000 оборотов. Запишем сначала решение задачи, а затем дадим пояснения.

S=0.

DO 10 N=1, 51

IMIN=2 * (N-1) * 1000+1

IMAX=IMIN+999

DO 1 I=IMIN, IMAX

1 S=S+1./I

10 CONTINUE

В нашем решении внешний цикл управляется переменной N, принимающей значения от 1 до 51 с шагом 1. Внутренний цикл с шагом 1 организован по переменной I, пределы изменения которой зависят от N и вычисляются перед входом во внутренний цикл. После выполнения внутреннего цикла, прибавляющего к имеющейся сумме очередную группу из 1000 слагаемых, происходит выход во внешний цикл. В ре-

зультате выполнения всего цикла к начальному значению суммы, равному 0, будет последовательно прибавлено 51000 слагаемых, что даст в итоге значение искомой суммы.

Обратим внимание на последний оператор внешнего цикла — CONTINUE. Здесь он понадобился для окончания записи внешнего цикла, так как после выхода из внутреннего цикла во внешнем цикле ничего делать не надо, а требуется лишь закончить его запись. Отметим, что в этом случае оператор CONTINUE уже не является «пустым» оператором, так как ему соответствуют определенные машинные команды, осуществляющие контроль окончания выполнения цикла. Отметим также, что можно было бы совместить последние операторы внешнего и внутреннего циклов. Для этого надо вместо

DO 10 N=1, 51

написать

DO 1 N=1, 51

и при этом убрать оператор CONTINUE.

Итак, одни циклы DO могут быть расположены внутри других, внешних, циклов DO. В таких случаях говорят, что внутренний цикл DO *вложен* во внешний цикл DO. А внешний цикл DO, содержащий внутренние циклы DO, называется *гнездом* DO. На БЭСМ-6 максимально допустимое число вложений циклов DO друг в друга равно 50.

При написании вложенных циклов DO (или гнезд DO) должно соблюдаться следующее *правило*.

Последний оператор внутреннего цикла DO должен либо предшествовать последнему оператору внешнего цикла DO, либо в крайнем случае совпадать с ним. Если при этом последние операторы внутреннего и внешнего циклов совпадают, то передача управления в конец цикла означает передачу управления в конец внутреннего цикла DO.

Таким образом, вложенные циклы DO напоминают «матрешек», вложенных друг в друга. При этом внутренние и внешняя матрешки могут иметь общее донышко.

Пр и м е р 3 многократного вложения циклов DO. Рассмотрим популярную задачу о «счастливых» билетах

на городском транспорте. Шестизначный номер такого билета считается «счастливым», если сумма трех первых его цифр равна сумме трех последних цифр. Например, номер 486594 — «счастливый». Итак решим следующую задачу.

Найти число «счастливых» билетов с номерами от 000000 до 999999 включительно.

Решение. Для решения задачи надо перебрать все шестизначные номера, т. е. написать один внешний цикл DO и 5 вложенных друг в друга внутренних DO. Число оборотов каждого цикла равно числу возможных цифр, т. е. 10. Общее число комбинаций цифр будет равно 10^6 , причем все комбинации будут различны.

По правилам записи операторов DO нельзя изменять значение параметра цикла от 0 до 9, как требуется по условию задачи. Поэтому значение параметра каждого цикла изменяется от 1 до 10.

Итак, получаем решение поставленной задачи:

NC=0		DO 1 M=1, 10
DO 1 I=1, 10		DO 1 N=1, 10
DO 1 J=1, 10		IF (I+J+K.EQ.L+M+N)
DO 1 K=1, 10		*NC=NC+1
DO 1 L=1, 10		1 CONTINUE

Здесь I, J, K, L, M, N обозначают увеличенные на единицу цифры номеров билетов. Признаком «счастливого» билета является равенство

$$I+J+K=L+M+N$$

при выполнении которого счетчик числа таких билетов NC увеличивается каждый раз на единицу.

Полное решение этой задачи (т. е. оформленное в виде программы с распечаткой результата) приведено в [29] (стр. 53, задача 47).

У читателя, быть может, возник вопрос: а нельзя ли в последнем примере окончить все циклы логическим оператором IF, т. е. приписать метку 1 этому оператору и заодно убрать оператор CONTINUE? Такая возможность разрешается, т. е. цикл может оканчиваться логическим оператором IF.

В нашем примере в случае истинности логического выражения, написанного внутри скобок, будет выпол-

нен оператор $NC=NC+1$, а в случае ложности этого выражения будет продолжено выполнение цикла.

Однако существуют ограничения на последний оператор цикла DO. Цикл DO не может оканчиваться арифметическим оператором IF, операторами GO TO, другим оператором DO, операторами RETURN или STOP (см. § 27). Во всех указанных случаях цикл DO надо оканчивать оператором CONTINUE.

Как видим, оператор DO налагает определенные ограничения на порядок расположения операторов внутри цикла. Это практически единственный случай, когда порядок расположения выполняемых операторов является существенным с точки зрения самого языка фортран. Вне цикла DO правила языка фортран не налагают ограничений на порядок следования выполняемых операторов.

Рассмотрим теперь некоторые правила, связанные с входом в цикл DO и с выходом из него.

Первый вход в цикл DO должен быть выполнен обязательно через оператор DO или, как говорят, через «голову». Например, запись

```

GO TO 10
DO 5 I=1, 120, 3
10 CONTINUE . . .
5 CONTINUE . .

```

недопустима, так как происходит передача управления внутрь цикла DO, минуя его «голову».

Однако изнутри цикла DO разрешается выход на любой внешний участок подпрограммы. При необходимости оттуда можно возвратиться внутрь данного цикла DO или другого цикла DO, содержащего данный. Например, запись

DO 5 I=1, 100	3 CONTINUE
DO 3 J=1, 100	5 CONTINUE .
GO TO 15 . .	15 CONTINUE .
20 CONTINUE .	GO TO 20 . .

вполне допустима.

Для правильного выполнения цикла DO необходимо также, чтобы параметр этого цикла не перепределялся внутри цикла. В частности, внутри данного цикла DO не может содержаться другой цикл DO, использующий тот же самый параметр цикла.

Например, недопустимой является запись вида

DO 1 I=1, 10		2 CONTINUE
DO 2 I=5, 130, 6		1 CONTINUE
.....		

В заключение кратко сформулируем

Основные правила записи циклов DO

1. Цикл DO начинается оператором

$$DO\ n\ i = m_1, m_2, m_3$$

где n — метка последнего оператора цикла; i — простая целая переменная; m_1 , m_2 и m_3 — целые константы без знака или простые целые переменные. Если $m_3=1$, то в записи оператора $DO\ m_3$ можно опустить (вместе с предшествующей запятой).

2. Цикл DO может оканчиваться любым оператором, кроме арифметического оператора IF, операторов GO TO, DO, RETURN и STOP. В этих случаях удобно оканчивать цикл оператором CONTINUE с меткой.

3. Внутренний цикл DO не может оканчиваться оператором, расположенным после оператора, которым оканчивается внешний цикл DO. Допускается окончание нескольких вложенных друг в друга циклов DO одним и тем же оператором.

4. Внутри цикла DO не допускается переопределения величин i , m_1 , m_2 и m_3 . В частности, цикл DO, расположенный внутри данного, не может использовать i в качестве переменной цикла.

5. Не разрешается входить внутрь цикла DO, минуя оператор DO. Однако можно выйти изнутри цикла DO и при необходимости возвратиться либо внутрь того же цикла, либо внутрь другого цикла DO, содержащего данный.

У п р а ж н е н и я. 15. Найти ошибки в записях циклов DO, приводимых ниже. Предполагается, что внутри циклов могут быть расположены некоторые выполняемые операторы

a) DO 1 I=1, N+1, 2. 6) DO 2 I=5, N, b) DO 3 I=1, 100, 5
1 CONTINUE 2 A=1. 3 GO TO 5

r) DO 4 I=1, 100	д) DO 6 I=5, 500, 6	е) GO TO 100;
DO 5 K=1, 200	DO 6 I=2, 150	DO 7 I=1, N
4 CONTINUE	6 CONTINUE	100 CONTINUE
5 CONTINUE		7 CONTINUE

```

ж) DO 8 J=1, K      8) DO 9 I=1, 10
   J=J+1              9 DO 10 J=1, 20
8 CONTINUE           10 CONTINUE

```

Записать программы вычисления сумм. Знак \pm означает, что надо определить, который из знаков $+$ или $-$ следует поставить.

16. $S = 1 + \frac{1}{2} - \frac{1}{3} + \frac{1}{4} - \frac{1}{5} + \dots + \frac{1}{10,000}.$

17. $S = 1 + \frac{1}{2} - \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \frac{1}{6} - \frac{1}{7} + \dots \pm \frac{1}{10,000}.$

Здесь чередуется одно слагаемое со знаком $+$ и два слагаемых со знаком $-$.

$$18. S = 1 + \frac{1}{2} - \frac{1}{3} - \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} - \frac{1}{8} - \frac{1}{9} - \frac{1}{10} - \frac{1}{11} + \dots \pm \frac{1}{10,000}.$$

Здесь сумма состоит из групп слагаемых: одно слагаемое с $+$, два слагаемых с $-$, три слагаемых с $+$, четыре слагаемых с $-$ и т. д. Последняя группа слагаемых может быть неполной.

19. $S = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \frac{1}{8} + \dots + \frac{1}{n}$.

Здесь каждый знаменатель, начиная с третьего, равен сумме двух предыдущих знаменателей (последовательность Фибоначчи). Суммирование производится до тех пор, пока $n \leq 10^9$.

20. Составить программу вычисления члена последовательности a_n , $n=1000$, если

$$a_1 = 1, \quad a_{n+1} = a_n + \frac{1}{a_n^2} \text{ при } n \geq 1.$$

21. Составить программу вычисления числа e , используя разложение в ряд

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \dots + \frac{1}{n!} + \dots$$

Ряд оборвать на слагаемом, которое не превосходит 10^{-10} .

§ 21. Массивы переменных.

Операторы описания типа

До сих пор мы имели дело с простыми переменными, т. е. с такими переменными, каждая из которых обозначалась собственным идентификатором, отличным от идентификаторов других переменных.

Теперь мы рассмотрим переменные другого вида. Начнем с задачи, которая позволит понять смысл и назначение нового вида переменных.

Пример 1. Дано 1000 чисел $a_1, a_2, \dots, a_{1000}$. Составить программу вычисления их среднего арифметического

$$S = \frac{a_1 + a_2 + \dots + a_{1000}}{1000}.$$

Решение. Если решать эту задачу теми средствами фортрана, которые уже нами рассмотрены, то надо каждое из чисел $a_1, a_2, \dots, a_{1000}$ обозначить собственным идентификатором, например, A1, A2, ..., A1000. После этого придется написать громадных размеров оператор фортрана, вычисляющий нужную величину. Такой способ не приемлем. Мы сделаем так, что каждая величина будет иметь свой номер, который в процессе счета можно увеличивать на 1, перебирая тем самым все числа $a_1, a_2, \dots, a_{1000}$. Это реализуется в фортране путем введения массива переменных.

Массив переменных — это упорядоченная совокупность переменных, имеющих один и тот же идентификатор и отличающихся друг от друга номером, или индексом, написанным в скобках. Тип массива определяется аналогично типу простой переменной. Применительно к нашей задаче мы можем объединить переменные A1, A2, ..., A1000 в массив, который назовем, например, A. Тогда каждая из указанных переменных

будет обозначаться как $A(1)$, $A(2)$, . . . , $A(1000)$. После этого задача решается просто:

$$\begin{aligned} S &= 0. \\ \text{DO } 1 \text{ } I &= 1, 1000 \\ 1 \text{ } S &= S + A(I) \\ S &= S/1000. \end{aligned}$$

Здесь внутри цикла производится прибавление к текущему значению суммы очередного слагаемого, являющегося элементом массива A с индексом I . Этот индекс изменяется от 1 до 1000, позволяя тем самым просуммировать все величины $A(1)$, $A(2)$, . . . , $A(1000)$, а затем вычислить их среднее арифметическое.

Каждой переменной величине соответствуют, как мы знаем, одна или две ячейки памяти машины. Всякий раз, когда в подпрограмме встречается новый идентификатор переменной, транслятор отводит для размещения соответствующей переменной (в зависимости от ее типа) одну или две ячейки памяти. Если производится объединение определенного количества переменных в массив, то для размещения его в памяти транслятор должен отвести надлежащее число ячеек, расположенных подряд. Для этого необходимо сообщить транслятору длину массива, т. е. количество переменных в нем. Это делается с помощью оператора описания массивов **DIMENSION**. Например, чтобы описать массив A , состоящий из 1000 переменных величин, надо написать оператор

DIMENSION A (1000)

Оператор **DIMENSION**, описывающий массив, является невыполняемым оператором (см. § 10). Он должен предшествовать любому выполняемому оператору подпрограммы.

Итак, описание одномерного массива имеет вид

DIMENSION A (N)

где N — целая константа без знака или простая целая переменная. В одном операторе **DIMENSION** можно описать несколько массивов, разделенных запятыми.

Величины, указанные в скобках после наименований массивов, определяют максимальные значения

индексов для элементов соответствующих массивов. Эти величины называются *размерностями* массивов.

Замечание. Массивы, размерности которых являются переменными, допускаются только в подпрограммах типа FUNCTION или SUBROUTINE (см. § 22 и 24), и притом лишь в качестве формальных параметров.

Пример 2. Дано 1000 чисел $a_1, a_2, \dots, a_{1000}$. Составить программу вычисления величин

$$S_1 = \frac{a_1 + \dots + a_{100}}{100}, \quad S_2 = \frac{a_{101} + \dots + a_{200}}{100}, \quad \dots$$

$$\dots, S_{10} = \frac{a_{901} + \dots + a_{1000}}{100},$$

т. е. надо вычислить средние арифметические для каждой группы из 100 чисел.

Решение. Сначала необходимо описать два массива: один длины 1000 для размещения переменных A, другой длины 10 для размещения переменных S. Затем надо написать два цикла: внутренний, вычисляющий значение среднего арифметического очередных ста слагаемых, и внешний, повторяющий эту процедуру 10 раз. Решение примера 2 можно записать, например, так:

DIMENSION A (1000), S (10) DO 10 I=1, 10 S (I)=0. J1=100 * (I - 1) + 1		J2=J1+99 DO 1 J=J1, J2 1 S (I)=S (I)+A (J) 10 S (I)=S (I)/100.
---	--	---

В процессе вычисления средних арифметических нам пришлось перед входом во внутренний цикл каждый раз вычислять значения нижнего и верхнего пределов изменения его индекса J. Было бы значительно удобнее, если бы можно было представить массив A как 10 подмассивов по 100 величин в каждом. Тогда индекс I внешнего цикла указывал бы номер подмассива, а индекс J внутреннего цикла — номер элемента в данном подмассиве.

Такая организация массива в фортране допустима и реализуется путем описания этого массива как двумерного.

Массив А в примере 2 можно описать так:

DIMENSION A (100, 10)

Тогда решение задачи примет вид

DIMENSION A (100, 10),	DO 1 J=1, 100
* S (10)	1 S (I)=S (I)+
DO 10 I=1, 10	+ A (J, I)
S (I)=0.	10 S (I)=S (I)/100.

Массив А (100, 10) состоит из 100 строк и 10 столбцов. При сквозной нумерации элементов от 1 до 1000 элементы двумерного массива А располагаются *по столбцам* (а не по строкам). Элементы первого столбца будут иметь номера с 1 по 100, элементы второго столбца — с 101 по 200 и т. д.

Итак, на фортране можно описывать двумерные массивы. Описание каждого двумерного массива имеет вид

DIMENSION A (M, N)

Здесь М и N — целые константы без знака или простые целые переменные (см. замечание на стр. 88), причем М обозначает число строк, а N — число столбцов в этом массиве. Элементы двумерного массива располагаются в нем по столбцам. Например, элементы массива А (2, 3) будут расположены в памяти машины в таком порядке:

А (1, 1), А (2, 1), А (1, 2), А (2, 2), А (1, 3), А (2, 3).

В версии фортрана для БЭСМ-6 разрешается описывать только одномерные, двумерные и трехмерные массивы. С описаниями одномерных и двумерных массивов мы уже познакомились. Описание трехмерного массива имеет вид

DIMENSION A(M, N, K)

Здесь М, N и К — целые константы без знака или простые целые переменные (см. замечание на стр. 88). При этом М обозначает число строк, N — число столбцов и К — число «этажей». Трехмерный массив можно наглядно представить себе как прямоугольный параллелепипед, состоящий из $M \cdot N \cdot K$ кубиков. Эти кубики образуют К «этажей», на каждом из которых они уложены в виде прямоугольника, имеющего М строк и

N столбцов. Нумерация «этажей» идет сверху вниз. Элементы трехмерного массива располагаются в нем следующим образом. Элементы первого «этажа» образуют двумерный массив размером (M, N), расположенный по столбцам. Этим элементам соответствуют номера

$$1, 2, \dots, M; M+1, \dots, 2M; \dots \\ \dots, M(N-1)+1, \dots, MN.$$

Далее располагаются элементы второго, третьего и т. д. этажей, — также по столбцам. Последними располагаются элементы K-го этажа, имеющие номера

$$MN(K-1)+1, \dots, MN(K-1)+M \dots \\ \dots, MN(K-1)+M(N-1)+1, \dots, MNK.$$

Мы рассмотрели описание одномерных, двумерных и трехмерных массивов. Каждому элементу такого массива соответствуют один, два или три индекса, однозначно определяющие положение этого элемента в массиве. Например, если описан массив

$$\text{DIMENSION X (10, 7, 3),}$$

то элемент X (K, N, L) расположен в K-й строке и N-м столбце L-го этажа. При этом должны, конечно, выполняться неравенства $1 \leq K \leq 10$, $1 \leq N \leq 7$, $1 \leq L \leq 3$.

Значения индексов у элементов массива могут быть не только целыми константами или простыми целыми переменными, но и арифметическими выражениями, принимающими значения целого типа.

Стандартной формой записи индекса является одна из следующих форм:

$$c * I \pm d, I \pm d, c * I, I, c$$

Здесь c и d — целые константы без знака, I — простая целая переменная, * — знак умножения.

В некоторых версиях фортрана допускается только стандартная форма записи индексов. В фортране на БЭСМ-6 разрешается любая форма индексов, лишь бы значение индекса было величиной целого типа. Например, можно написать оператор

$$A(K(I)+I)=0.$$

конечно, при условии, что описаны массивы А и К. Однако в неявных циклах (см. § 28) допускается только стандартная форма записи индексов.

Отметим важное обстоятельство, связанное с использованием дву- и трехмерных массивов. Мы можем обозначить элементы этих массивов как переменные с меньшим числом индексов. Можно, например, для обозначения элемента двумерного массива использовать один индекс, соответствующий порядковому номеру данного элемента в массиве.

Пусть, например, нам требуется просуммировать все элементы двумерного массива А (10, 20). Это можно сделать так:

DIMENSION A (10, 20) S=0. DO 1 I=1, 10	DO 1 J=1, 20 1 S=S+A (I, J)
--	--------------------------------

Однако можно предложить более простое решение, которое записывается короче и выполняется быстрее:

```

      DIMENSION A (10, 20)
      S=0.
      DO 1 I=1, 200
1  S=S+A (I)
  
```

Аналогично, элементы трехмерного массива, описанного оператором

```

      DIMENSION A (10, 6, 15)
  
```

можно использовать как элементы двумерного массива А(60, 15) или одномерного — А (900).

Сформулируем кратко основные

*Правила описания массивов и использования
их в подпрограмме*

1. В версии фортрана для БЭСМ-6 разрешается использовать одномерные, двумерные и трехмерные массивы.

2. Каждый массив должен быть описан. Описание массива может быть сделано оператором вида

```

      DIMENSION A1, A2, . . . , An
  
```

Здесь идентификаторы переменных А_і (і=1, 2, . . . , n) могут иметь одну, две или три размерности, заключенные в скобки и разделенные запятыми. Размерности

могут быть целыми константами без знака или простыми целыми переменными (см. замечание на стр. 88). Оператор DIMENSION является невыполняемым оператором. Он располагается в подпрограмме до первого выполняемого оператора.

3. Тип массива определяется аналогично типу простой переменной. Тип каждого элемента массива должен совпадать с типом массива.

4. Разрешается использовать элементы трехмерного массива как переменные с двумя индексами или с одним индексом, а элементы двумерного массива — как переменные с одним индексом. В фортране IV это не так.

5. В подпрограмме может находиться любое число операторов DIMENSION. Порядок их взаимного расположения может быть произвольным.

6. Длина массива равна произведению размерностей, написанных в скобках после идентификатора этого массива. Длина массива — это количество его элементов (но не машинных слов!).

7. Если элемент массива указан без индексов, то считается, что указан первый элемент массива. Это верно для всех операторов, кроме операторов ввода-вывода (см. § 28).

Перейдем к *операторам описания типа*. Эти операторы предназначены для того, чтобы указать тип переменной, массива или функции. В § 11 было выяснено, что каждой величине должен быть приписан один из 5 типов: вещественный, целый, комплексный, с двойной точностью, логический. В случае, если величина является вещественной или целой, тип ее описывать не обязательно: при отсутствии описания он будет определяться по первой букве ее идентификатора. Однако таким способом нельзя определить типы комплексный, с двойной точностью и логический. Кроме того, может случиться, что идентификатор, обозначающий величину вещественного или целого типа, начинается не с той буквы, которая нужна для правильного указания типа. Во всех таких случаях возникает необходимость описания типа с помощью соответствующего оператора. Описание типа переменной, массива или функции производится одним из опера-

торов вида

```
REAL R1, R2, ..., Rn  
INTEGER I1, I2, ..., In  
COMPLEX C1, C2, ..., Cn  
DOUBLE PRECISION DP1, DP2, ..., DPn  
LOGICAL L1, L2, ..., Ln
```

Здесь в списках указываются идентификаторы описываемых переменных, массивов или функций. При этом идентификаторы массивов могут быть указаны с размерностями или без них. Если идентификатор массива указан без размерностей, то массив должен быть описан одним из операторов DIMENSION или COMMON (см. § 25). Если же массив описан вместе с размерностями, то он уже не может быть описан оператором DIMENSION, а в операторе COMMON можно указать лишь его наименование без размерностей.

Рассмотрим несколько п р и м е р о в.

1. REAL A, K12(7, 6), PSI(18), NOT
DIMENSION NOT (3, 5, 8)

Здесь оператором REAL описаны простая вещественная переменная A и вещественные массивы K12, PSI, NOT. При этом размерности массивов K12 и PSI указаны непосредственно в операторе REAL, а размерности массива NOT описаны в операторе DIMENSION. Порядок расположения операторов REAL и DIMENSION не существен.

2. Операторы

```
DIMENSION ARG (7)  
COMPLEX NT (18, 2), ARG, FIT, ELLA
```

описывают комплексные массивы NT и ARG и простые комплексные переменные FIT и ELLA.

3. Операторы

```
DOUBLE PRECISION AS (39), GM (52), AR  
LOGICAL NEFE (42), RTS, TBH, HART
```

описывают массивы и простые переменные с двойной точностью и логические.

В приведенных примерах идентификаторы без размерностей могут быть и наименованиями функций.

В заключение сформулируем основные

Правила записи операторов описания типа

1. Оператор описания типа является невыполняемым оператором. Он должен находиться среди невыполняемых операторов, предшествующих первому выполняемому оператору данной подпрограммы.

2. Каждый идентификатор может быть описан не более чем в одном операторе описания типа.

3. Идентификатор, не включенный в оператор описания типа, будет определять целую величину, если первая его буква есть I, J, K, L, M или N. В противном случае он будет определять вещественную величину.

4. Если операторы описания типа и операторы DIMENSION, COMMON или EQUIVALENCE (см. § 25) появляются вместе, то порядок их следования не существен.

У п р а ж н е н и я. 22. Описать целые величины R15, KSI (8), TRP, ANNA (18, 9, 6) и комплексные величины FI, ALLA (3, 6), NIK (12, 3, 7).

23. Найти ошибки в следующих группах операторов:

- а) REAL A (15), N, PSI, LT
DIMENSION A (15), LT (70)
- б) COMPLEX ARG, CTA (18), IVP (37)
REAL ARG (7), NT, PSI, L
- в) DIMENSION NP (16, 5), TOP (19)
COMPLEX TOP (19), ASA, .NS
- г) DOUBLE PRECISION DIV (120), P5
DIMENSION P5 (7), CK (15)
INTEGER AP, CK (15), P5

§ 22. Подпрограмма-функция

В процессе составления подпрограммы на фортране может возникнуть необходимость в вычислении значения некоторой величины, зависящей от других величин. Например, это может быть функция

$$f(x) = 2 \sin x + 4,5xe^{-x^{1/2}}$$

или, скажем, сумма диагональных элементов двумерного массива A, имеющего N строк и N столбцов. В первом случае величина f однозначно зависит от переменной x , во втором случае искомая сумма однозначно определяется массивом A и значением переменной N.

Такие величины в фортране, как и в математике, называются *функциями*.

Вычисление значения функции можно реализовать в виде некоторого блока внутри подпрограммы. Однако часто бывает удобнее оформить эту вычислительную процедуру в виде самостоятельной подпрограммы, называемой *подпрограммой-функцией*. Например, вычисление функции $f(x)$ можно оформить так:

```
FUNCTION F (X)
F=2. * SIN (X)+4.5 * X * EXP (-X ** 2/2.)
RETURN
END
```

Здесь FUNCTION — заголовок вычислительной процедуры, реализующей вычисление значения функции F, зависящей от аргумента X. Далее следует оператор, присваивающий наименованию функции ее значение, являющееся арифметическим выражением, содержащим X. Затем написан оператор RETURN, обозначающий возврат в ту подпрограмму, из которой данная подпрограмма-функция F была вызвана. Последний оператор END обозначает конец записи подпрограммы-функции.

Покажем теперь, как можно использовать подпрограмму-функцию F. Пусть, например, требуется вычислить величину Y, равную значению функции F при X, равном

$$3.+2.*A(I)+ABS(A(I+1)+C/A(2*I))$$

Тогда можно присвоить значение указанного арифметического выражения некоторой переменной вещественного типа, скажем, T и затем написать

$$Y=F(T)$$

Можно, однако, непосредственно указать арифметическое выражение в скобках у функции F:

$$Y=F(3.+2.*A(I)+ABS(A(I+1)+C/A(2*I)))$$

Пусть теперь мы хотим узнать, будет ли значение F при X, равном $5.+D/SQRT(C+EXP(-A/4.))$ меньше, равно или более 3. и в зависимости от этого

разветвиться на операторы с метками 15, 37 и 21. Тогда мы можем написать

IF (F (5.+D/SQRT (C+EXP (—A/4.)))—3.) 15, 37, 21

Рассмотренные примеры показывают, что функцию можно использовать в арифметических выражениях наравне с константами и переменными. При этом в качестве значений аргументов (в скобках после наименования функции) можно указывать любые арифметические выражения. Это верно для функций вещественного, целого и комплексного типов и функций с двойной точностью. Если функция имеет логический тип, то ее можно использовать в логических выражениях и в выражениях отношения наравне с константами и переменными логического типа.

Величины, от которых зависит значение функции, на фортране называются *параметрами* данной функции. Параметры, указанные в самой подпрограмме-функции, называются *формальными* параметрами. В нашем примере функция F имеет один формальный параметр X. Параметры, указанные при вызове (т. е. использовании) данной подпрограммы-функции, называются *фактическими* параметрами.

Покажем теперь, как можно оформить вычисление значения суммы диагональных элементов массива A, имеющего N строк и N столбцов. Вот решение этой задачи:

FUNCTION SUM (A, N)	1 S=S+A (I, I)
DIMENSION A (N, N)	SUM=S
S=0.	RETURN
DO 1 I=1, N	END

Здесь формальными параметрами являются массив A и переменная N, обозначающая число строк и столбцов этого массива. Обратим внимание на то, что размерности массива A здесь заданы не в виде целых констант (как всюду было ранее), а в виде простых целых переменных. Для размещения элементов массива A, как мы знаем, должен быть отведен участок памяти нужного размера. Здесь размер этого участка памяти заранее не известен, так как значение N может меняться в процессе счета. Таким образом, подпрограмма

функция SUM не может внутри себя отвести под массив A соответствующий участок памяти. Это может быть сделано в подпрограмме, вызывающей данную подпрограмму-функцию SUM. Это утверждение справедливо для всех формальных параметров, а не только для тех из них, которые являются массивами с переменными размерностями (см. § 26).

Пусть нам надо вычислить значение той же функции F, но для комплексных значений параметра X. Тогда и значение функции F будет, вообще говоря, комплексной величиной. Это обстоятельство должно быть как-то указано в записи подпрограммы-функции F. В фортране предусмотрено два способа определения типа подпрограммы-функции. Первый из них состоит в том, что указатель типа подпрограммы-функции (REAL, INTEGER и т. д.) ставится перед заголовком соответствующей подпрограммы-функции. В нашем случае это означает, что заголовок подпрограммы-функции F примет вид

COMPLEX FUNCTION F(X)

Другой способ состоит в том, что тип подпрограммы функции описывается аналогично типу переменной. В случае комплексной подпрограммы-функции F это можно сделать так:

```
FUNCTION F(X)  
COMPLEX F
```

С учетом сказанного решение задачи можно записать, например, так:

```
COMPLEX FUNCTION F (X)  
COMPLEX X  
F=2. * CSIN (X)+4.5 * X * CEXP (-X ** 2/2.)  
RETURN  
END
```

В подпрограмме, вызывающей данную подпрограмму-функцию, необходимо ее описать оператором COMPLEX F. Кроме того, необходимо предусмотреть, чтобы значение фактического параметра (также описанного оператором COMPLEX) каждый раз было комплексной величиной.

Рассмотрим еще один пример. Даны массивы $A(M)$ и $B(N)$. Вычислить сумму их максимальных элементов.

Решение. Составим сначала подпрограмму-функцию, вычисляющую максимальный элемент вещественного одномерного массива. Затем составим другую подпрограмму-функцию, вычисляющую сумму максимальных элементов массивов A и B .

<pre> FUNCTION AMAX (A, N) DIMENSION A (N) AM = A (1) DO 1 I = 2, N IF (A (I).GT.AM) * AM = A (I) 1 CONTINUE AMAX = AM RETURN END </pre>	<pre> FUNCTION SUMMAX = (A, M, B, N) DIMENSION A (M), = B (N) SUMMAX = AMAX = (A, M) + AMAX (B, N) RETURN END </pre>
--	--

Здесь вычислительный процесс организован таким образом, что одна подпрограмма-функция вызывает другую, передавая ей часть своих параметров. Например, массивы A и B передаются подпрограмме-функции $SUMMAX$ извне и затем каждый из них поочередно передается далее, т. е. подпрограмме-функции $AMAX$. Таким образом, формальный параметр подпрограммы-функции может служить фактическим параметром при вызове другой подпрограммы-функции.

Рассмотренные примеры позволяют понять механизм передачи информации от одной подпрограммы-функции (или подпрограммы) к другой посредством параметров.

Каждая подпрограмма-функция объявляет список своих формальных параметров. Формальный параметр может быть простой переменной или массивом любого типа, а также наименованием подпрограммы или подпрограммы-функции (см. § 26).

Другая подпрограмма, вызывающая данную подпрограмму-функцию, обязана при вызове ее указать фактические параметры, соответствующие формальным параметрам. При этом должны совпадать количество формальных и фактических параметров, их

типы и порядок их следования. Если формальный параметр есть массив, то ему должен соответствовать фактический параметр, также являющийся массивом. Если формальный параметр является простой переменной, то в качестве фактического параметра ему может соответствовать любое выражение (арифметическое, логическое или выражение отношения) того же типа.

Сформулируем в заключение основные

Правила записи подпрограмм-функций

1. Всякая подпрограмма-функция должна начинаться одним из операторов вида

```
FUNCTION F (P1, P2, . . . , Pn)  
REAL FUNCTION F (P1, P2, . . . , Pn)  
INTEGER FUNCTION F (P1, P2, . . . , Pn)  
COMPLEX FUNCTION F (P1, P2, . . . , Pn)  
DOUBLE PRECISION FUNCTION F (P1, P2, . . . , Pn)  
LOGICAL FUNCTION F (P1, P2, . . . , Pn)
```

Здесь F — наименование подпрограммы-функции, P₁, P₂, . . . , P_n — ее формальные параметры ($1 \leq n \leq 63$).

2. Наименование подпрограммы-функции не может упоминаться в невыполняемых операторах внутри нее, за исключением оператора описания типа.

3. Наименование подпрограммы-функции должно упоминаться внутри нее хотя бы один раз: либо как левая часть оператора присваивания, либо как фактический параметр при вызове другой подпрограммы-функции или подпрограммы. При этом хотя бы один из указанных операторов должен выполняться.

4. Ни один элемент списка формальных параметров не должен упоминаться в операторах COMMON, EQUIVALENCE или DATA (см. §§ 25 и 26) внутри данной подпрограммы-функции.

5. Если формальный параметр — идентификатор массива, то он должен быть описан в операторе DIMENSION или операторе описания типа внутри подпрограммы-функции.

6. Подпрограмма-функция должна иметь хотя бы один формальный параметр.

7. Внутри подпрограммы-функции должен быть хотя бы один оператор RETURN.

У п р а ж н е н и я. Даны вещественные массивы $A(N, N)$ для упр. 24, 25 и $A(M, N)$ для упр. 26—29. Составить подпрограммы-функции, вычисляющие следующие величины.

24. Сумму элементов его строки с номером I .

25. Сумму элементов его диагонали, идущей от элемента $A(1, 1)$ к элементу $A(N, N)$.

26. Номер строки, имеющей максимальную сумму элементов.

27. Сумму элементов строки и столбца, на пересечении которых расположен максимальный элемент массива A .

28. Сумму попарных произведений соответственных элементов строк с номерами I и J (скалярное произведение строк).

29. Сумму квадратов элементов массива A , удовлетворяющих условию $X \leq A(K, L) \leq Y$, где X и Y — заданные числа.

§ 23. Функции-операторы

Для вычисления значения величины, однозначно зависящей от параметров, можно, как мы видели, составить соответствующую подпрограмму-функцию. Однако это не всегда удобно, так как наименования подпрограммы-функции и ее тип жестко закреплены за ней. Список формальных параметров подпрограммы-функции также не может быть изменен извне. Было бы желательно иметь возможность при определенных условиях менять наименование функции, ее тип и список ее формальных параметров, не изменяя ее содержания. Такая возможность реализуется в фортране посредством функций-операторов.

Функция-оператор аналогична подпрограмме-функции, с той лишь разницей, что сама функция-оператор располагается внутри подпрограммы. При этом необходимо выполнение условия, что запись функции-оператора должна состоять из одного оператора. Покажем на примере функции

$$f(x) = 2\sin x + 4, 5xe^{-x^2/2},$$

уже рассмотренной в § 22, как оформляется запись функции-оператора. Эта запись является фортранной копией математической записи данной функции:

$$F(X) = 2. * \text{SIN}(X) + 4.5 * X * \text{EXP}(-X ** 2/2.)$$

Таким образом, запись функции-оператора представляет собой оператор присваивания, в левой части которого указывается наименование функции вместе со списком ее формальных параметров, а в правой

части — выражение, определяющее значение данной функции. Это выражение может быть арифметическим или логическим выражением, либо выражением отношения. Формальными параметрами функции-оператора могут быть лишь простые переменные. В правой части функции-оператора не должно быть переменных с индексами. Важно отметить, что наименования формальных параметров функции-оператора могут использоваться в подпрограмме и для обозначения других объектов.

Функции-операторы должны быть расположены в подпрограмме до первого выполняемого оператора, но после всех невыполняемых операторов типа COMMON, DIMENSION и др. В правой части функции-оператора могут быть указаны любые допустимые константы и простые переменные, а также подпрограммы-функции и ранее определенные функции-операторы.

Вызов функции-оператора аналогичен вызову подпрограммы-функции. Фактические параметры могут содержать любые выражения, соответствующие по типу формальным параметрам данной функции-оператора.

Функции-операторы, в отличие от подпрограмм-функций, являются внутренними объектами данной подпрограммы. Их иногда называют внутренними функциями, в отличие от внешних функций, представляющих собой подпрограммы-функции. Это означает, что данная функция-оператор может быть использована лишь в той подпрограмме, в которой она указана. Это означает также, что в случае совпадения наименования функции-оператора с наименованием подпрограммы-функции будет вызвана функция-оператор.

Правила записи функции-оператора

1. Функция-оператор имеет вид

$$F(P_1, P_2, \dots, P_n) = E$$

Здесь F — наименование функции-оператора, P_i — формальные параметры, E — выражение (арифметическое, логическое или выражение отношения). При этом $1 \leq n \leq 63$.

2. Наименование функции-оператора не должно упоминаться в операторах COMMON, DIMENSION,

EQUIVALENCE, DATA и EXTERNAL. Тип функции-оператора может быть описан в операторе описания типа аналогично описанию типа переменной.

3. Формальные параметры должны быть простыми переменными.

4. Выражение E не должно содержать переменных с индексами, но может содержать обращения к любым подпрограммам-функциям и ранее определенным функциям-операторам.

5. Все функции-операторы должны быть расположены перед первым выполняемым оператором подпрограммы, но после всех невыполняемых операторов (COMMON, DIMENSION и др.).

6. При совпадении наименования функции-оператора с наименованием подпрограммы-функции будет вызвана функция-оператор.

У п р а ж н е н и я. Определить, можно ли оформить в виде функции-оператора следующие вычислительные процедуры. Если да, то записать соответствующие функции-операторы.

30. Вычисление значения функции

$$a) f(x, y) = 3 \sin(x + y) + 2,5e^{-\sqrt{x^2 + y^2}};$$

$$б) f(x) = \begin{cases} 3\sqrt{x^2 - 1}, & \text{если } |x| \geq 1, \\ 0, & \text{если } |x| < 1; \end{cases}$$

в) $F(X) = X * \sin(X) + A(I, J)$, где A — двумерный массив.

31. Вычисление максимального элемента массива A (100).

32. Вычисление суммы элементов массива X (10, 20).

§ 24. Подпрограмма (SUBROUTINE)

В § 22 мы рассмотрели один из возможных способов оформления вычислительной процедуры в виде подпрограммы-функции. Здесь мы рассмотрим другой, более универсальный, способ оформления вычислительной процедуры в виде подпрограммы (SUBROUTINE). В этом параграфе слово «подпрограмма» будет означать подпрограмму типа SUBROUTINE.

П р и м е р. Вычислить значение функции

$$f(x) = 5 \sin x + 2e^{-x^2}.$$

Р е ш е н и е. Вычисление $f(x)$ в виде подпрограммы-функции можно представить так:

```
FUNCTION F (X)
F=5. * SIN (X)+2. * EXP (-X ** 2)
RETURN
END
```

Оформим теперь вычисление $f(x)$ в виде подпрограммы

```
SUBROUTINE SF (X, F)
F=5. * SIN (X)+2. * EXP (-X ** 2)
RETURN
END
```

Отличие подпрограммы от подпрограммы-функции состоит в заголовке, а также в том, что значение вычисляемой величины не присваивается наименованию подпрограммы. В данном случае для обозначения вычисляемой величины потребовался дополнительный формальный параметр F.

Можно, однако, вместо формальных параметров использовать общие блоки (см. § 25), описав в этих блоках любую из переменных X или F, или обе эти величины. Например, нашу подпрограмму можно записать таким образом:

```
SUBROUTINE SF
COMMON/T/X, F
F=5. * SIN (X)+2. * EXP (-X ** 2)
RETURN
END
```

Здесь оператор COMMON (см. § 25) описывает общий блок T, содержащий переменные X и F, являющиеся общими для нескольких подпрограмм.

Как видно из рассмотренных примеров, в подпрограмме по сравнению с подпрограммой-функцией отсутствуют многие ограничения на форму представления величин в виде формальных параметров или элементов общих блоков. В частности, допускается подпрограмма без параметров.

Для того чтобы использовать некоторую подпрограмму, необходимо написать *оператор вызова* этой

подпрограммы, имеющий вид

CALL S или CALL S (P_1, P_2, \dots, P_n)

Здесь S — наименование подпрограммы, P_i — фактические параметры, причем $1 \leq i \leq 63$.

Оператор CALL передает управление подпрограмме, которая будет выполняться до тех пор, пока в ней не встретится оператор RETURN. По окончании выполнения подпрограммы управление будет передано первому оператору вызывающей подпрограммы, написанному вслед за оператором CALL. Однако, если оператор CALL является последним оператором в цикле DO, то после возврата из подпрограммы продолжится выполнение цикла.

Правила записи подпрограмм

во многом аналогичны правилам записи подпрограмм-функций. Ниже мы приведем лишь те правила, которые отличны от соответствующих правил для подпрограмм-функций.

1. Запись подпрограммы начинается с оператора вида

SUBROUTINE S или SUBROUTINES (P_1, P_2, \dots, P_n)

где S — наименование подпрограммы, а P_i — ее формальные параметры ($1 \leq i \leq 63$).

2. Наименование подпрограммы не может появиться внутри нее, в операторе присваивания, в операторе ввода или вывода, в невыполняемом операторе или в качестве фактического параметра в операторе CALL.

3. Вызов подпрограммы производится оператором вида

CALL S или CALL S (P_1, P_2, \dots, P_n)

где S — наименование подпрограммы, P_i — фактические параметры ($1 \leq i \leq 63$).

У п р а ж н е н и я. Составить подпрограммы, реализующие следующие вычислительные процедуры.

33. В двумерном массиве A (M, N) переставить строку, содержащую максимальный элемент массива, со строкой, содержащей минимальный его элемент.

34. В двумерном массиве $X(10, 35)$ переставить строки в обратном порядке, сохраняя порядок расположения элементов в каждой строке.

35. В двумерном массиве $A(N, N)$ переставить в обратном порядке элементы диагонали, идущей от элемента $A(1, 1)$ к элементу $A(N, N)$.

36. Умножить матрицу $A(N, N)$ на матрицу $B(N, N)$, получив в результате матрицу $C(N, N)$.

37. В двумерном массиве $A(N, N)$ вычислить суммы элементов каждой диагонали, идущей параллельно главной диагонали, т. е. диагонали, соединяющей элемент $A(1, 1)$ с элементом $A(N, N)$. Результаты поместить в одномерный массив S в таком порядке: $A(N, 1)$, затем — сумма элементов диагонали, состоящей из двух элементов $A(N-1, 1)$ и $A(N, 2)$, и т. д. Сумма с номером N содержит элементы главной диагонали. Последняя сумма массива S должна совпадать с элементом $A(1, N)$.

§ 25. Общие блоки. Эквивалентности. Данные

1. Оператор **COMMON**. Программа на фортране, как уже говорилось, может состоять из отдельных подпрограмм. Эти подпрограммы составляются и транслируются независимо друг от друга. Независимость подпрограмм друг от друга означает, в частности, что все обозначения (т. е. идентификаторы и метки) каждой подпрограммы, кроме наименований подпрограмм и общих блоков (см. ниже), выбираются независимо от обозначений, принятых в других подпрограммах. Это означает также, что каждая подпрограмма будет расположена на своем участке машинной памяти, недоступном для других подпрограмм. Таким образом, подпрограммы как бы изолированы друг от друга.

Однако для совместной работы нескольких подпрограмм необходимо, чтобы определенная информация могла передаваться от одной подпрограммы к другой. В фортране предусмотрено два способа передачи информации от одной подпрограммы к другой: через общие блоки и через аппарат формальных и фактических параметров (см. § 26). Здесь мы опишем первый способ, состоящий во введении так называемых общих блоков, или **COMMON-блоков**. Рассмотрим конкретный пример, который позволит понять смысл и назначение этих блоков.

Пример 1. Дан массив вещественных чисел a_1, a_2, \dots, a_{100} . Пусть a_{\max} — максимальное, и a_{\min} —

минимальное из этих чисел. Вычислить значение функции

$$f(x, y) = \sin x + (3xy + 5y^2)e^{-(x+y)} + \sqrt{x^2 + y^2}$$

при $x = a_{\max}$, $y = a_{\min}$.

Решение. Составим две подпрограммы, первая из которых вычисляет значения a_{\max} и a_{\min} , а вторая — значение функции $f(a_{\max}, a_{\min})$. Будем предполагать, что массив чисел передается первой подпрограмме из какой-то другой подпрограммы. Кроме того, предположим, что значение функции, вычисленное второй подпрограммой, может понадобится некоторой третьей подпрограмме для дальнейших вычислений. Таким образом, первая подпрограмма должна получить извне массив чисел a_1, a_2, \dots, a_{100} , отыскать в этом массиве a_{\max} и a_{\min} и передать второй подпрограмме их значения. Вторая подпрограмма должна вычислить $f(a_{\max}, a_{\min})$ и передать значение этой функции некоторой третьей подпрограмме.

```
SUBROUTINE SUB1 $ DIMENSION A (100)
COMMON /INPUT/ A
COMMON /RESULT/ AMAX, AMIN
AMAX=A (1) $ AMIN=A (1)
DO 2 I=2, 100
IF (A (I).GT.AMAX) AMAX=A (I)
IF (A (I).LT.AMIN) AMIN=A (I)
2 CONTINUE $ RETURN $ END
SUBROUTINE SUB2
COMMON /RESULT/ X, Y /EXY/ F
F=SIN (X)+(3. * X * Y+5. * Y ** 2) * EXP
* (- (X+Y))+SQRT (X ** 2+Y ** 2)
RETURN $ END
```

Поясним, для чего служат общие блоки в обеих подпрограммах. Начнем с общего блока COMMON /RESULT/.

COMMON — это обозначение оператора фортрана, предназначенного для описания общих блоков.

/RESULT/ — это идентификатор общего блока, заключенный в косые черточки (те же самые, что приняты в фортране для обозначения знака операции деления).

В подпрограмме SUB1 в общем блоке /RESULT/ описаны переменные AMAX и AMIN, а в подпрограмме SUB2 в том же общем блоке указаны переменные X и Y.

По смыслу задачи подпрограмма SUB2 должна вычислить значение функции $f(a_{\max}, a_{\min})$. Таким образом, переменная, обозначенная в подпрограмме SUB1 идентификатором AMAX, и переменная, обозначенная в подпрограмме SUB2 через X, должны располагаться в одной и той же ячейке памяти. Иначе говоря, переменным AMAX и X соответствует одна и та же ячейка памяти, обозначенная в разных подпрограммах по-разному. Аналогичное соответствие должно быть и между переменными AMIN и Y.

Таким образом, между величинами, расположенными в одном и том же общем блоке, но описанными в разных подпрограммах, устанавливается взаимно однозначное соответствие. Это соответствие реализуется одним и тем же порядковым номером машинного слова в данном общем блоке.

Например, если первое машинное слово общего блока /RESULT/ в подпрограмме SUB1 соответствует переменной AMAX, а в подпрограмме SUB2 — переменной X, то тем самым эти переменные обозначают одно и то же машинное слово.

Если бы в подпрограмме SUB2 было описание

COMMON/RESULT/Y, X

то установилось бы соответствие: AMAX \leftrightarrow Y, AMIN \leftrightarrow X. В случае же описания в SUB 2

COMMON/RESULT/ T (2)

соответствие выглядело бы так: AMAX \leftrightarrow T(1), AMIN \leftrightarrow T(2).

В подпрограмме SUB1 есть общий блок /INPUT/, отсутствующий в подпрограмме SUB2. Это означает, что информация из этого общего блока недоступна подпрограмме SUB2. Однако, если в SUB2 или любой другой подпрограмме будет описан общий блок /INPUT/, то информацией из этого блока можно будет воспользоваться и в этой подпрограмме.

Иначе обстоит дело с индексной переменной I из подпрограммы SUB1. Эта переменная не описана

ни в каком общем блоке, поэтому она недоступна никакой другой подпрограмме.

Такие величины называются внутренними или собственными величинами данной подпрограммы.

Таким образом, каждая подпрограмма может содержать внутренние и общие величины.

Внутренняя величина — это величина, которая может быть использована, вообще говоря, только в данной подпрограмме. Передача внутренних величин другим подпрограммам возможна только через фактические параметры при вызове этих подпрограмм из данной подпрограммы (см. § 26).

Общая величина — это величина, описанная в некотором общем блоке и доступная всем подпрограммам, имеющим описание этого общего блока.

Для описания общих блоков служит оператор `COMMON /c1/ Список1 /c2/ Список2 . . . /cn/ Списокn`

Здесь c_i — идентификатор общего блока. Идентификатор может быть и пустым — в этом случае ему соответствует непомеченный общий блок. Непомеченный блок обозначается либо опусканием идентификатора блока вместе с наклонными черточками, либо двумя наклонными черточками // перед списком общего блока. В подпрограмме разрешается описывать не более одного непомеченного общего блока.

Список общего блока составляется из идентификаторов простых переменных и идентификаторов массивов (с размерностями или без них). Если в списке появился идентификатор массива без размерностей, то значения их определяются оператором `DIMENSION` или оператором описания типа в той же подпрограмме. Если в списке указан массив с размерностями, то в этом случае массив не может быть описан оператором `DIMENSION`.

П р и м е р ы записи общих блоков.

```
COMMON /ABC/ A, B (15), T (7, 6), ABC
COMMON /A/ A /B/ B (8, 6) /TETA/ FI, R, S
COMMON T (7), PSI (8, 8), ALFA
COMMON // T (7), PSI (8, 8), ALFA
```

Записи двух последних общих блоков эквивалентны. Как видно из примеров, идентификаторы общих бло-

ков могут совпадать с идентификаторами некоторых описываемых в них величин.

Операторы

COMMON /AB/ A (15), B (7, 6), FI, RT
COMPLEX A, FI, T, C (20)
INTEGER B, PSI (15), TETA
DIMENSION FI (10), X (20)

описывают общий блок, состоящий из комплексного массива A (15), целого массива B (7, 6), комплексного массива FI (10) и простой вещественной переменной RT. Кроме того, эти операторы описывают внутренние величины: комплексную переменную T, комплексный массив C (20), целый массив PSI (15), целую переменную TETA и вещественный массив X (20). Рассмотренные операторы можно располагать в произвольном порядке.

Сформулируем теперь

Основные правила записи оператора COMMON

1. COMMON — невыполняемый оператор. Он должен находиться среди невыполняемых операторов, предшествующих первому выполняемому оператору. В подпрограмме может быть написано любое количество операторов COMMON.

2. Порядок следования операторов COMMON, DIMENSION, EQUIVALENCE (см. ниже) и операторов описания типа не существен.

3. Идентификаторы общих блоков используются только для опознавания блоков в процессе трансляции и загрузки. Поэтому они могут быть использованы в подпрограмме для обозначения других объектов.

4. Идентификатор из одного общего блока не может быть описан в другом общем блоке.

5. Описание массива вместе с размерностями может быть сделано не более чем в одном из операторов COMMON, DIMENSION или операторов описания типа.

6. В подпрограмме идентификатор блока может встречаться не более одного раза. Непомеченный блок может быть только один.

Перейдем теперь к рассмотрению важного вопроса о длине общего блока. Длина общего блока — это

количество машинных слов в нем. Она определяется количеством и типом величин, перечисленных в списке данного блока. Определим длину общего блока /AB/ в последнем из рассмотренных примеров. Этот блок состоит из комплексного массива A (15), целого массива B (7, 6), комплексного массива FI (10) и вещественной переменной RT. Следовательно, длина его составляет $2 \cdot 15 + 7 \cdot 6 + 2 \cdot 10 + 1 = 93$ машинных слова. Пусть указанный блок описан в некоторой подпрограмме S1. Пусть, далее, другой подпрограмме S2 нужен лишь целый массив B (7, 6). Тогда в этой подпрограмме можно сделать, например, такое описание:

COMMON /AB/ DUMMY (30), N (7, 6)

Здесь вещественный массив DUMMY (30), соответствующий комплексному массиву A (15), описан для того, чтобы начало целого массива B совпало с началом массива N. Массив FI и переменная RT подпрограмме S2 не нужны, и поэтому соответствующие им машинные слова в данном блоке не указаны. Тем самым длина блока, описанного в подпрограмме S2, равна не 93, а 72 машинным словам. Такое несоответствие длин общих блоков, описанных в разных подпрограммах, допустимо не всегда (см. § 34). Отметим, что поскольку подпрограммы транслируются независимо друг от друга, транслятор не может обнаружить никаких несоответствий в длинах общих блоков.

Важно отметить, что введение общих блоков позволяет разным подпрограммам использовать один и тот же участок памяти. Поэтому иногда в целях экономии памяти приходится вводить общие блоки, хотя это и не связано с необходимостью обмена информацией между подпрограммами. Более подробно этот вопрос рассмотрен в § 58.

2. Оператор EQUIVALENCE. При составлении подпрограмм иногда может возникнуть ситуация, когда два идентификатора, скажем, A и B, должны обозначать одну и ту же величину. Иначе говоря, идентификаторам A и B должна соответствовать одна и та же ячейка памяти. В этом случае надо указать, что идентификаторы A и B эквивалентны. Это делается

оператором

EQUIVALENCE (A, B)

Если надо, чтобы идентификатор C был также эквивалентен A и B, то это можно сделать так:

EQUIVALENCE (A, B), (A, C)

или так:

EQUIVALENCE (A, B), (B, C)

или, проще, так:

EQUIVALENCE (A, B, C)

Запишем теперь общий вид оператора эквивалентности:

EQUIVALENCE (A₁, B₁, . . .),
(A₂, B₂, . . .), . . ., (A_n, B_n, . . .)

Здесь (A_i, B_i, . . .) — группа эквивалентности двух или более простых переменных, или переменных с одним индексом.

Если необходимо включить в оператор EQUIVALENCE переменную с несколькими индексами, то ее следует представить как переменную с одним индексом. При этом переменной A (I, J, K) из массива A с размерностями L, M и N соответствует переменная с одним индексом, равным

$$I + (J - 1) * L + (K - 1) * L * M$$

Например, переменная A (2, 1, 3) из массива A (6, 3, 5) будет представлена как A (38).

П р и м е р ы. Операторы

DIMENSION A (8, 9), B (15)

EQUIVALENCE (A (2), B (5))

означают, что в памяти машины совмещены элементы A (2) и B (5). Тем самым A (1) стал эквивалентным B (4), A (3) — эквивалентным B (6) и т. д. Если бы не было оператора эквивалентности, то массивы A и B расположились бы в памяти последовательно и заняли бы вместе 87 машинных слов (в предположении, что оба массива не комплексные и не с двойной точностью). Теперь, при наличии эквивалентности, элементы массива B, начиная с B (4), совместились с элементами

более длинного массива А. Элементы же В(1), В(2) и В(3) оказались за пределами массива А. Таким образом, теперь для размещения обоих массивов потребуется не 87, а всего 75 ячеек памяти. Из сказанного следует, что с помощью эквивалентности можно экономить память, требуемую данной подпрограмме. Более подробно этот вопрос рассмотрен в § 58.

Оператор

EQUIVALENCE (A(2), B(6)), (A(5), B(1))

является ошибочным, так как если А(2) эквивалентно В(6), то отсюда следует, что А(5) эквивалентно В(9), а не В(1).

Рассмотрим теперь вопрос о том, как можно использовать в операторе эквивалентности элементы из общих блоков. При рассмотрении этого вопроса мы должны исходить из того, что каждый общий блок занимает в памяти машины жестко зафиксированное место и «двигать» его уже нельзя. Отсюда следует, что эквивалентность (прямая или косвенная) между элементами общих блоков запрещена.

Если же эквивалентность затрагивает, с одной стороны, элементы из общего блока и, с другой стороны, внутренние величины некоторой подпрограммы, то такая эквивалентность возможна с некоторыми ограничениями. Поясним это примером.

Пусть написаны операторы

```
DIMENSION A (8, 9), B (15)
EQUIVALENCE (A (2), B (5))
COMMON /A/ A
```

Как уже было установлено, элементы В(1), В(2) и В(3) массива В расположатся в памяти машины перед началом массива А. Но так как начало массива А совпадает с началом общего блока, то массив В не может быть размещен без изменения начала этого блока. А так как начало общего блока менять нельзя, то такая эквивалентность запрещена.

Пусть теперь заданы операторы

```
DIMENSION A (8, 9), B (15)
EQUIVALENCE (A (2), B (5))
COMMON /B/ B
```

т. е. теперь массив В находится в общем блоке, а массив А не в общем блоке. Массив А, как более длинный, будет выходить за пределы массива В, но с конца его, а не с начала. В этом случае за счет эквивалентности длина общего блока увеличится на 60 машинных слов. Таким образом, хотя формально описан общий блок длиной 15 слов, фактическая его длина будет 75 машинных слов.

Рассмотрим вопрос об эквивалентности величин различных типов. Здесь никаких дополнительных ограничений не налагается, т. е. в операторе EQUIVALENCE можно записывать величины любых типов. Однако необходимо следить за тем, чтобы в процессе счета всякий раз использовалась величина именно того типа, который был присвоен последним.

Например, если написаны операторы

```
EQUIVALENCE (A, N)
N=1
B=1./A
```

то на БЭСМ-6 будет выдана ошибка по делению на ненормализованное число (см. § 31), так как в ячейке, соответствующей переменной А, оказалась величина целого типа. Сформулируем теперь основные

Правила записи оператора EQUIVALENCE

1. Оператор EQUIVALENCE является невыполняемым оператором. Он должен находиться среди невыполняемых операторов, предшествующих первому выполняемому оператору в подпрограмме.

2. Порядок следования операторов EQUIVALENCE, COMMON, DIMENSION и операторов описания типа не существен.

3. Эквивалентными могут быть массивы и переменные любых типов (без индексов или с одним индексом).

4. Массивы, элементы которых описаны как эквивалентные в операторе EQUIVALENCE, могут удлинить общий блок, но начало блока не должно меняться.

5. В операторе EQUIVALENCE не допускается установление эквивалентности (прямо или косвенно) между элементами общих блоков.

3. Оператор DATA. В ряде случаев возникает необходимость до выполнения подпрограммы занести

в некоторые места памяти определенную информацию. Пусть, например, необходимо, чтобы значение переменной *A* при работе подпрограммы всегда равнялось 1. Это можно сделать, например, оператором присваивания *A*=1. Однако указанный оператор будет выполняться при каждом входе в подпрограмму, отнимая лишнее время и занимая лишнюю память.

Для занесения в определенные ячейки памяти начальной информации служит оператор DATA. Этот оператор позволяет присваивать переменным (в том числе и переменным из помеченных общих блоков) начальные значения до выполнения программы.

Оператор DATA имеет вид
DATA (*A*₁=список), (*A*₂=список), . . . , (*A*_{*n*}=список)
Здесь *A*_{*i*} может быть простой переменной, идентификатором массива или переменной с индексом, заданным в виде целой константы. Допускается занесение информации с помощью неявного цикла (см. § 28).

В списке оператора DATA могут быть указаны константы всех 5 типов, а также строки текстовых символов и восьмеричные константы. Строка текстовых символов имеет вид текстовой константы (см. § 12) с той лишь разницей, что здесь допускается запись более 6 символов. Каждая строка заносится в несколько последовательных машинных слов из расчета 6 символов на машинное слово (неполное последнее слово дополняется пробелами). В списке могут быть подсписки, которые можно повторить несколько раз. Каждый такой подсписок заключается в скобки с указанием перед скобками числа его повторений в виде целой константы.

В качестве примера рассмотрим операторы

DIMENSION A (20), B (30)

DATA (A=1.5,4 (32,5.),7 (1.),

*14H FORTRAN_ BESM—6, 2), (B=4 (1.)),

*(B (8)=2 (1.5,—1.,3.))

В массив A заносятся последовательно 20 констант: 1.5, 32, 5., 32., 5., 32, 5., 32, 5., 1., 1., 1., 1., 1., 1., 1., 6HFORTRA, 6HN_BESM, 2H—6, 2. Массив B заполняется так: сначала заносят единицы в элементы B(1), B(2), B(3), B(4), затем в элементы, начиная с B(8), заносятся 6 констант: 1.5, —1., 3., 1.5,

—1., 3. Остальные элементы массива В (т. е. с В(5) по В(7) и с В(14) по В(30) включительно) не заполняются и их состояние считается неопределенным.

Как видно из примера, оператором DATA можно заносить информацию не во все элементы массива, оставляя часть массива незаполненной. Однако в списке оператора DATA не может быть указано больше констант, чем требуется для заполнения массива.

Отметим одно важное свойство оператора DATA. В приведенном примере в вещественный массив А заносятся вещественные, целые и текстовые (т. е. тоже целого типа) константы. Тем самым не все элементы массива А будут иметь тип, соответствующий типу массива.

Однако операции над элементами массива А будут выполняться в расчете на то, что все эти элементы имеют вещественный тип. Например, при выполнении оператора $C=1./A(2)$ будет предполагаться, что в А(2) находится вещественное число. А так как оператором DATA туда занесено целое число 32, то на БЭСМ-6 будет выдана ошибка по делению на нуль (см. § 34).

Таким образом, запись вида

DATA (A=1)

приведет к тому, что значение переменной А будет иметь не тот тип, который требуется.

В операторе присваивания вида $A=1$ можно не опасаться за правильность типа значения переменной А: транслятор вставит нужные команды, преобразующие целую константу 1 в вещественную 1. В невыполняемом операторе DATA такого преобразования типа не производится.

Приведем пример использования неявного цикла (§ 28) в операторе DATA

DIMENSION A (30)

DATA ((A (I), I=1, 30, 2)=1., 6 (3.14), 4 (1.3, 2.7))

Здесь производится занесение информации в элементы массива, индексы которых меняются от 1 до 30 с шагом 2, т. е. заполнение элементов массива А с нечетными индексами.

Кроме рассмотренной формы записи оператора DATA, на БЭСМ-6 допускается запись этого оператора,

принятая в версии фортран-IV. Эта запись имеет вид

DATA $A_1, \dots, A_n/I_1 * D_1, \dots, I_k * D_k/, A_{n+1}, \dots,$
 $A_m/I_{k+1} * D_{k+1}, \dots, I_p * D_p/.$

Здесь A_1, \dots, A_m — простые переменные, переменные с индексами (в этом случае индексы должны быть целыми константами) или идентификаторы массивов; D_1, \dots, D_p — константы любых типов; I_1, \dots, I_p — целые константы без знака, указывающие число повторений данной константы при занесении информации. Многоточие справа означает, что список оператора DATA может быть продолжен. Например, запись

DIMENSION X (20), Y (10), Z (15)
DATA X, Y/5*2., 25*1.E3/, Z/7*2.5, 8*1.E4/

эквивалентна записи

DIMENSION X (20), Y (10), Z (15)
DATA (X=5 (2.), 15 (1.E3)), (Y=10 (1.E3)),
*(Z=7 (2.5), 8 (1.E4))

Занесение информации, указанной в операторе DATA, в соответствующие ячейки памяти производится на этапе загрузки подпрограмм в память машины. Это означает, что если несколько подпрограмм заносят информацию в один и тот же элемент помеченного общего блока, то туда будет записана информация, указанная в той подпрограмме, которая будет загружена позже других. Отметим, что процессом загрузки можно управлять с помощью оператора EXTERNAL (см. § 32).

В фортране на БЭСМ-6, как и в фортране-IV, имеется возможность составлять подпрограммы (BLOCK DATA), состоящие только из невыполняемых операторов и предназначенные для занесения информации в общие блоки (см. § 27). Сформулируем

Основные правила записи оператора DATA

1. DATA — невыполняемый оператор. Он должен находиться среди невыполняемых операторов, предшествующих первому выполняемому оператору в подпрограмме.

2. Порядок следования операторов DATA и других невыполняемых операторов не существен.

3. В операторе DATA не могут содержаться массивы переменных размерностей, являющиеся формальными параметрами подпрограммы (см. § 26).

4. Тип засылаемых оператором DATA констант может не соответствовать типу переменных, так как константы засылаются без преобразования их транслятором.

У п р а ж н е н и я. 38. Описать общие блоки, содержащие следующие массивы и переменные. Определить длины этих блоков.

а) Комплексный массив C (15), логическую переменную L, массив с двойной точностью D (10, 6), вещественный массив A (10, 7, 3).

б) Целый массив R (15, 10), массив с двойной точностью F (7, 6), логический массив LT (16).

в) Вещественный массив A (7, 5), логический массив T (10), целый массив TN (10, 6, 2), целые переменные N, PSI и D.

39. Описать следующие эквивалентности, выяснив при этом их допустимость. Определить суммарную длину массивов с учетом эквивалентности.

а) Элемент вещественного массива A (15, 6) с индексами (2, 4) эквивалентен элементу комплексного массива C (10, 8) с индексами (8, 5). Оба массива не в общем блоке.

б) Элемент массива с двойной точностью D (16, 2, 5) с индексами (5, 2, 3) эквивалентен элементу целого массива N (6, 70) с индексами (5, 23). Массив D — в общем блоке, массив N — не в общем блоке.

в) Переменная с двойной точностью D эквивалентна элементу комплексного массива CT (10, 6) с индексами (3, 2) и элементу целого массива N (5, 5) с индексами (2, 4). Массив CT — в общем блоке; массив N и переменная D — не в общем блоке.

40. Найти ошибки в следующих записях:

а) COMMON /ABC/ C (12)

DIMENSION R (25)

EQUIVALENCE (R (8), C (3))

б) DIMENSION A (18, 3), B (16, 7)

EQUIVALENCE (A (8, 2), B (3, 5))

в) DIMENSION A (25), B (18), C (15)

EQUIVALENCE (A (5), B (7)), (B (10), C (13)), (A, C)

г) COMMON /A/ B (8) /D/ N (15)

EQUIVALENCE (B (13), N (8))

д) COMMON /T/ X (18)

DIMENSION R (15), S (28)

EQUIVALENCE (R (6), S), (S (3), X (4))

41. Занести начальные значения с помощью оператора DATA в следующие массивы и переменные:

а) В вещественный массив A (8, 3), начиная с элемента с индексами (2, 2), — пять раз константу 1.5 и два раза константу 77B.

б) В массив с двойной точностью D (18, 2) — три раза константу 2.5D8 и пять раз константу 1.65 D4.

в) В целую переменную T — константу 15, в вещественный массив NX (10) — пять раз константу 1.5, два раза константу 3.8, три раза константу — 5.6.

§ 26. Формальные и фактические параметры.

Оператор EXTERNAL

При составлении фортранных подпрограмм формальные и фактические параметры играют очень важную роль. Рассмотрим эти понятия более подробно на конкретных примерах.

Пример 1. Пусть имеются три подпрограммы

SUBROUTINE	SUBROUTINE S2	SUBROUT-
*S1	COMMON /AA/ A	INE S3 (A)
A=1.	A=1.	A=1.
RETURN	RETURN	RETURN
END	END	END

На первый взгляд кажется, что все три подпрограммы делают одно и то же — присваивают переменной A значение вещественной константы 1. Однако имеются большие различия в характере переменной A, используемой в каждой из трех подпрограмм.

В первой подпрограмме переменная A является внутренней, или собственной переменной данной подпрограммы. Это означает, что внутри подпрограммы S1 для переменной A отведена ячейка памяти. Эта ячейка является «собственностью» подпрограммы S1. Никакая другая подпрограмма без «согласия» подпрограммы S1 не может изменить содержимого этой ячейки.

Во второй подпрограмме переменная A является элементом общего блока. Это означает, что ячейка памяти, отведенная для размещения переменной A, расположена вне подпрограммы S2. Это означает также, что данная ячейка уже не является «собственностью» подпрограммы S2 и может быть использована любой другой подпрограммой, в которой описан общий блок /AA/.

В третьей подпрограмме переменная А является формальным параметром. Это означает, что адрес ячейки памяти, куда будет занесено значение вещественной константы 1., заранее не известен подпрограмме S3 и будет зависеть от способа вызова этой подпрограммы.

Пример 2.

```
SUBROUTINE S
CALL S3 (B)
T=2. * B ** 2
RETURN
END
```

В результате в ячейку памяти, отведенную для переменной В, будет занесено значение вещественной константы 1. Иначе говоря, в этом случае формальному параметру А в подпрограмме S3 будет соответствовать фактический параметр В, являющийся собственной переменной подпрограммы S.

Пример 3.

<pre>SUBROUTINE P1 DIMENSION A (10, 3) CALL P2 (A (7, 2)) RETURN END SUBROUTINE P2 (X) DIMENSION X (10)</pre>	<pre>CALL P3 (X(3)) RETURN END SUBROUTINE P3 (T) T=1. RETURN END</pre>
---	--

Выясним, куда будет занесено значение константы 1. Это будет адрес третьего элемента массива X, первый элемент которого соответствует элементу A(7, 2) из массива A(10, 3). Иначе говоря, искомый адрес равен адресу элемента A(7, 2) плюс 2, т. е. адресу элемента A(9, 2).

Таким образом, при вызове одной подпрограммы из другой в качестве фактического параметра передается адрес ячейки памяти, соответствующий этому фактическому параметру. Такой способ передачи информации, когда при передаче ее указывается не адрес самой информации, а адрес ячейки, в которой указан адрес расположения информации, называется *косвенной адресацией*.

Если некоторая величина является внутренней величиной данной подпрограммы или элементом общего блока, то адресация этой величины будет прямой, т. е. подпрограмме будет известен адрес ячейки памяти, в которой расположена данная величина. Если же величина является формальным параметром, то подпрограмме будет известен только адрес ячейки памяти, в которой находится адрес данной величины. Отсюда следует, что формальный параметр не может быть элементом общего блока.

Рассмотрим еще несколько примеров передачи информации через посредство формальных и фактических параметров.

Пример 4. Пусть имеется подпрограмма

```
SUBROUTINE INVERT (X, Y)
Y=1./X
RETURN
END
```

Эта подпрограмма вычисляет значение вещественной переменной Y, равное $1./X$, где X — также значение некоторой вещественной переменной. В данном случае указывается адрес некоторой ячейки памяти, содержащей значение исходной величины, и адрес другой ячейки памяти, куда заносится результат. Конкретное содержание этих ячеек памяти определяется в подпрограмме, вызывающей данную подпрограмму. Например, если написан оператор

CALL INVERT (2.3+5. * SIN (X), A (1, 6))

то тем самым будет вычислена обратная величина для арифметического выражения, указанного в качестве первого фактического параметра. Значение этой величины будет присвоено элементу A (1,6).

Пусть теперь мы написали оператор

CALL INVERT (2, A)

т. е. хотим вычислить величину, обратную целой константе 2. В данном случае нарушено соответствие между типами формального и фактического параметров. В результате на БЭСМ-6 мы получим сообщение об ошибке при делении на ненормализованное число (см. § 31).

Пусть теперь написан оператор

CALL INVERT (X, 1.)

В результате его выполнения будет «испорчена» ячейка, в которой находилась константа 1.

Итак, формальный параметр — это адрес некоторой ячейки памяти, конкретное значение которого определяется в вызывающей подпрограмме.

Важным обстоятельством при использовании формальных параметров является то, что в качестве фактического параметра может быть указано наименование подпрограммы-функции или подпрограммы.

Пример 5. Пусть составлена подпрограмма-функция

FUNCTION AMID (A, B, N, F)		S=S+F (X)
H=(B-A)/(N-1)		1 X=X+H
S=0.		AMID=S/N
X=A		RETURN
DO 1 I=1, N		END

Эта подпрограмма-функция производит вычисление среднего арифметического N значений функции $F(X)$, где X принимает значения из отрезка A, B с постоянным шагом $H = (B - A)/(N - 1)$. Подпрограмма-функция $F(X)$ может быть любой, причем наименование ее нам заранее не известно. Требуется только, чтобы это была вещественная функция от одного вещественного параметра. Пусть ее наименование есть POLY и нам надо найти среднее арифметическое 100 ее значений, заданных на отрезке $[0.5, 2.7]$ с постоянным шагом. Тогда, казалось бы, достаточно написать оператор

$Y = \text{AMID}(0.5, 2.7, 100, \text{POLY})$

Однако здесь не хватает информации о том, что POLY есть наименование подпрограммы-функции или подпрограммы. А если нет этой дополнительной информации, то транслятор отведет для величины POLY одну ячейку памяти и будет считать ее простой вещественной

переменной. При вызове подпрограммы-функции AMID адрес этой переменной будет передан в качестве адреса фактического параметра. При выполнении подпрограммы-функции AMID произойдет передача управления по этому адресу, а там вместо начала нужной подпрограммы-функции будет находиться некоторая «посторонняя» величина. Произойдет потеря управления с возможной диагностикой: «контроль команды» (см. § 31).

Для того чтобы отличить наименования подпрограмм-функций и подпрограмм, являющихся фактическими параметрами, от иных типов фактических параметров, необходимо описать эти наименования в операторе EXTERNAL.

Общий вид этого оператора:

$$\text{EXTERNAL } N_1, N_2, \dots, N_k$$

где N_i — наименования подпрограмм-функций и подпрограмм. EXTERNAL — невыполняемый оператор. Он должен располагаться среди других невыполняемых операторов, т. е. до первого выполняемого оператора. Его расположение среди невыполняемых операторов может быть произвольным.

Отметим, что в операторе EXTERNAL можно описывать любые подпрограммы-функции и подпрограммы, а не только те из них, которые будут использоваться в качестве фактических параметров. Отметим также, что если формальный параметр есть подпрограмма-функция, то тип этой подпрограммы-функции определяется по обычному правилу, т. е. либо по первой букве ее наименования (как формального параметра), либо путем описания в операторе описания типа.

Согласование между типом фактически заданной подпрограммы-функции и типом формального параметра, ей соответствующего, должно быть обеспечено пользователем. Точно так же формальному параметру, являющемуся FUNCTION, должен соответствовать фактический параметр FUNCTION, а не SUBROUTINE.

В заключение рассмотрим еще один пример.

Пример 6. Составить подпрограмму-функцию для вычисления значений многочлена степени N с коэффициентами, заданными в массиве A , начиная со старшего.

Р е ш е н и е. Вот одно из возможных решений:

FUNCTION POLYNO (A, N, X)	1 P=P*X+A (I)
DIMENSION A (1)	POLYNO=P
P=A (1)	RETURN
N1=N+1	END
DO 1 I=2, N1	

Поясним описание массива A, содержащего коэффициенты многочлена. Размерность этого массива равна N+1. Однако описание

DIMENSION A (N+1)

не годится, так как размерность массива не может быть арифметическим выражением. Описание вида

DIMENSION A (M),

где $M = N+1$ требует задания еще одного формального параметра. Можно, конечно, потребовать, чтобы вместо степени многочлена в вызывающей подпрограмме задавалось число его коэффициентов. Однако это не всегда удобно. Поскольку массив A является формальным параметром, то назначение оператора DIMENSION в данном случае сводится к тому, чтобы указать, что A есть массив. Размерность массива можно указать, любой, например, равной 1.

У п р а ж н е н и я. 42. Составить подпрограмму-функцию, вычисляющую сумму минимального и максимального элементов вещественного одномерного массива A, содержащего N элементов.

43. С помощью подпрограммы-функции предыдущего упражнения вычислить:

а) Сумму минимального и максимального элементов массива X (30)

б) Сумму минимального и максимального элементов массива R (6, 8)

44. Составить подпрограмму (SUBROUTINE либо FUNCTION), вычисляющую значение интеграла на отрезке [A, B] по формуле трапеций с делением отрезка на N равных частей, $N \geq 2$. Подпрограмма-функция с одним параметром, вычисляющая значение подынтегральной функции, должна служить ей формальным параметром.

45. С помощью подпрограммы предыдущего упражнения вычислить значение интеграла от функции e^{-x^2} на отрезке [0, 1], разделенном на 50 равных частей.

46. Составить подпрограмму вычисления числа сочетаний из n элементов по k :

$$C_n^k = \frac{n(n-1) \dots (n-k+1)}{k!}$$

при заданных n и k . Учесть при этом, что C_n^k — целое число (в обычном смысле).

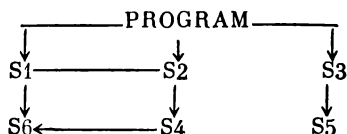
§ 27. Структура фортранной программы. Операторы ENTRY и RETURN

До сих пор мы говорили об отдельных подпрограммах фортранной программы. Теперь рассмотрим программу в целом.

Для того чтобы фортранная программа могла начать работу, необходимо указать ту подпрограмму, с которой начинается работа всей программы. Эта подпрограмма, называемая *главной* (или *головной*) подпрограммой, имеет особый заголовок PROGRAM, вслед за которым указывается ее наименование. PROGRAM не может иметь формальных параметров. Любая другая подпрограмма (SUBROUTINE или FUNCTION) может быть вызвана из PROGRAM. Однако никакая SUBROUTINE или FUNCTION не может вызывать PROGRAM.

С помощью управляющей карты *MAIN (см. § 8) можно объявить главной любую подпрограмму, не имеющую формальных параметров.

Изобразим схематически конкретный пример взаимодействия подпрограмм фортранной программы



Здесь стрелками указаны вызовы подпрограмм (SUBROUTINE или FUNCTION). Из схемы видно, что одна и та же подпрограмма может быть вызвана из разных подпрограмм. Схема фортранной программы должна удовлетворять, однако, двум требованиям:

- 1) в схеме должна быть главная подпрограмма;
- 2) в схеме не должно быть циклов, т. е. при любом движении по стрелкам мы должны в итоге попасть на подпрограмму, из которой стрелки уже не выходят.

Сборка программы из подпрограмм производится не на стадии трансляции, а на стадии загрузки. Каждая подпрограмма, которая либо описана в операторе EXTERNAL, либо вызывается из других подпрограмм, будет размещаться в памяти машины с помощью специальной системной программы-загрузчика (см. § 7). При этом разные подпрограммы могут находиться на разных носителях информации (перфокартах, магнитных барабанах, лентах, дисках) и могут быть написаны не только на фортране, но и на других входных языках системы «Дубна» (см. § 8).

Кроме главной подпрограммы и произвольного количества подпрограмм с заголовками SUBROUTINE или FUNCTION, в состав программы могут входить так называемые блоки данных.

Каждый блок данных начинается заголовком подпрограммы

BLOCK DATA

и заканчивается оператором END. Наименование подпрограммы BLOCK DATA не указывается. В качестве наименования блока данных транслятор выбирает наименование первого COMMON-блока, указанного в этом BLOCK DATA. Внутри блока данных могут быть описаны общие блоки, в которые с помощью операторов DATA занесена начальная информация. Например,

```
BLOCK DATA
COMMON /DIM/ A (15), I (10, 2)
DATA A/3 * 1.E3, 2 * 1., 8 * 10.2/, I/10 * 1, 10 * 2/
END
```

BLOCK DATA в отличие от SUBROUTINE или FUNCTION, не занимает места в памяти машины. При обработке подпрограммы BLOCK DATA загрузчик производит рассылку данных в соответствии с описаниями, в ней указанными.

Опишем еще три вида операторов, используемых в подпрограммах PROGRAM, SUBROUTINE и FUNCTION.

Оператор ENTRY позволяет организовать дополнительный вход в подпрограмму. Он имеет вид

ENTRY S

где S — наименование входа в данную подпрограмму.

Оператор ENTRY может находиться в любом месте подпрограммы, но только не внутри цикла DO. Он не должен иметь метки. Наименование ENTRY должно быть согласовано с наименованием основного входа по типу заголовка и по списку формальных параметров. Максимальное число входов в подпрограмму (включая основной, задаваемый заголовком) не должно превосходить 20. Обращение к наименованию входа делается по тем же правилам, что и обращение к FUNCTION или SUBROUTINE.

Пример. Пусть имеется подпрограмма-функция

```
FUNCTION F (X)
F=SQRT (X+1.)+EXP (-1./X)
RETURN
END
```

Здесь вычисляется значение функции, которая не определена при $X < -1$ и при $X = 0$. Мы хотим доопределить функцию F для указанных значений X, положив ее значение равным нулю. Это можно сделать путем составления отдельной подпрограммы-функции. Однако удобнее оформить это в качестве дополнительного входа в подпрограмму-функцию F следующим образом:

```
FUNCTION F (X)
1 F=SQRT (X+1.)+EXP (-1./X)
RETURN
ENTRY FD
F=0.
IF (X. GE.-1.AND.X.NE.0.) GO TO 1
RETURN
END
```

Такая составная подпрограмма-функция занимает меньше места в памяти, чем две отдельные подпрограммы-функции.

Использование входа FD производится так же, как и основного входа F. Например, можно написать оператор

$$Z = FD(2. + 5. * \sin(T) - \text{ABS}(T + 2.))$$

Каждый вход должен заканчиваться выходом в вызывающую подпрограмму. Выход производится с помощью оператора RETURN, уже рассмотренного ранее.

Если начало выполнения программы — это первый выполняемый оператор в PROGRAM, то окончанием работы программы служит либо оператор END главной подпрограммы, либо оператор STOP, расположенный в любой из подпрограмм.

В случае, если задача использует магнитную ленту или диск, работа программы должна обязательно заканчиваться оператором STOP (см. также § 6).

§ 28. Операторы ввода-вывода

Среди операторов фортрана особое место занимают операторы ввода-вывода. Они предназначены для организации обмена информацией между оперативной памятью машины (МОЗУ) и внешними носителями информации.

Ввод информации в память машины может быть произведен либо с перфокарт (или с перфоленты), либо с внешних запоминающих устройств: магнитных лент, барабанов или дисков. Вывод информации можно произвести на широкоформатную печать (АЦПУ), на перфокарты (или на перфоленту), а также на внешние запоминающие устройства. Некоторые современные ЭВМ оборудованы графопостроителями (плоттерами) и экранами типа телевизионных кинескопов (дисплеями), позволяющими выводить информацию в наглядной форме.

Рассмотрим сначала оператор вывода информации на печать, который встречается в подавляющем большинстве задач.

1. Оператор вывода PRINT. Печатающее устройство БЭСМ-6 (АЦПУ-128) производит печать на специальной бумажной ленте, имеющей отверстия по краям. Через каждые 305 мм лента имеет поперечную насечку, разделяющую ее на листы, складывающиеся «гармошкой». На одном листе можно напечатать 66 строк по 128 символов в каждой строке. Переход на новый лист производится либо автоматически после каждых 66 строк, либо программно, по мере необходимости. При желании можно заблокировать листование и печатать сплошным массивом строк (см. § 6).

Чтобы вывести на печать нужные величины, надо указать, что выдать на печать и как печатаемый материал расположить на бумаге.

П р и м е р. Пусть надо выдать на печать значение вещественной переменной *A* и целой переменной *N*. Тогда мы можем написать

```
PRINT 10, A, N
10 FORMAT (F20.9, I9)
```

Здесь PRINT — наименование оператора вывода информации на печать, 10 — метка оператора FORMAT, определяющего расположение печатаемой информации на бумаге. (Оператор FORMAT рассматривается в § 29).

Выдача информации на печать производится следующим образом. Просматривается список величин в операторе PRINT и одновременно просматривается содержимое соответствующего оператора FORMAT. При этом устанавливается взаимно однозначное соответствие между величинами, выдаваемыми на печать, и так называемыми спецификациями формата.

Рассмотрим более подробно оператор PRINT. Общий вид этого оператора таков:

```
PRINT n, L
```

Здесь *n* — метка оператора FORMAT, соответствующего данному оператору PRINT, *L* — список величин, выдаваемых на печать. Элементы списка разделяются запятыми. В качестве элементов списка могут быть:

- 1) простые переменные,
- 2) переменные с индексами,
- 3) массивы переменных,
- 4) элементы массивов, определенные с помощью неявного цикла.

Если запись вида

```
DIMENSION A (100)
A=1.
```

означает, что значение константы 1. присваивается первому элементу массива *A*, то запись вида

```
DIMENSION A (100)
PRINT 20, A
```

означает, что выдается на печать весь массив *A*. Для выдачи только первого элемента массива *A* надо написать

```
PRINT 20, A (1)
```

Индексы в списке оператора PRINT могут иметь одну из следующих форм:

$$c * I \pm d, I \pm d, c * I, I, c. \quad (*)$$

Здесь c и d — целые константы без знака, I — простая целая переменная, предварительно определенная или определяемая в самом списке оператора PRINT.

Примеры.

PRINT 115, A (I), C, T (3, J+1)

PRINT 22, I (K), A (J+1, K+5, 7), C (K, 1)

Рассмотрим теперь элементы списка оператора PRINT, заданные с помощью неявного цикла. *Неявный цикл* является удобным способом задания определенной последовательности элементов массива. Пусть, например, мы хотим выдать на печать значения простых переменных C, D, K и элементов массива A с нечетными номерами, не превосходящими N. Тогда мы можем написать

PRINT 10, C, D, K, (A (I), I=1, N, 2)

Здесь в скобках указано, что печатаются элементы массива A, номера которых изменяются от 1 до N (или до N-1, если N — четное) с шагом 2. Эта запись во многом напоминает запись цикла DO (потому она и называется неявным циклом).

Общая форма записи одномерного неявного цикла

$$(A (M), I=m_1, m_2, m_3)$$

Здесь M — индекс элемента массива A, который может иметь одну из форм (*), I — переменная неявного цикла, m_1, m_2, m_3 — целые константы без знака или предварительно определенные целые переменные, принимающие положительные значения. Если m_3 опущено, то оно полагается равным 1.

Общая форма записи двумерного неявного цикла имеет вид

$$((A (M_i, M_j), I=m_1, m_2, m_3), J=n_1, n_2, n_3)$$

или

$$((A (M_i, M_j), J=n_1, n_2, n_3), I=m_1, m_2, m_3)$$

Здесь M_i, M_j — индексы, зависящие от I и J соответственно и принимающие одну из форм (*); I и J —

переменные двумерного неявного цикла; $m_1, m_2, m_3, n_1, n_2, n_3$ — целые константы без знака или целые переменные, принимающие положительное значение.

Выборка элементов из массива A производится так. Сначала выбирается элемент, соответствующий начальным значениям переменных I и J . Затем переменная во внутренних скобках изменяется на величину своего шага, пробегая все значения, согласно записи цикла во внутренних скобках. Затем переменная во внешних скобках изменяется на величину своего шага и снова происходит изменение переменной во внутренних скобках от начального ее значения до конечного. Выполнение двумерного неявного цикла аналогично выполнению двух циклов DO, вложенных друг в друга.

Пусть, например, имеется массив $A(M, N)$, и мы хотим распечатать его по строкам. Тогда мы можем написать

PRINT 15, ((A (I,J), J = 1, N), I = 1, M)

Для выдачи на печать элементов массива A по столбцам можно написать

PRINT 15, ((A (I, J), I = 1, M), J = 1, N)

Так как элементы массива A расположены по столбцам, то в этом случае достаточно и более простой записи

PRINT 15, A

В случае трехмерного неявного цикла в зависимости от порядка следования переменных этого цикла получаются 6 возможных вариантов его записи. Мы дадим самую общую форму записи такого неявного цикла:

((A(M_i, M_j, M_k), $r_1 = m_1, m_2, m_3$),
 $r_2 = n_1, n_2, n_3$), $r_3 = p_1, p_2, p_3$)

Здесь M_i, M_j, M_k — индексы, зависящие от переменных цикла I, J, K ; r_1, r_2, r_3 — некоторая перестановка индексов I, J, K . Величины $m_1, m_2, m_3, n_1, n_2, n_3, p_1, p_2, p_3$ могут быть целыми константами без знака или целыми переменными, принимающими положительные значения. Если m_3, n_3 или p_3 опущены, то они полагаются равными 1. Например, запись вида

((A(I, J, K), K = 1, NK), J = 1, NJ), I = 1, NI)

применительно к массиву A (NI, NJ, NK) означает, что выборка его из массива A производится так:

A (1, 1, 1), A (1, 1, 2), . . . , A (1, 1, NK),
A (1, 2, 1), A (1, 2, 2), ...,
A (1, 2, NK), . . . , A (1, NJ, 1), A (1, NJ, 2), . . .

Самым внутренним циклом здесь является цикл по номеру этажа, расположенный в самых внутренних скобках, затем идет цикл по номеру столбца и самым внешним является цикл по номеру строки.

2. Оператор ввода READ. Оператор READ, предназначенный для ввода информации с перфокарт, имеет вид

READ *n*, L

Здесь *n* — метка оператора FORMAT, определяющего расположение вводимой информации на перфокартах; L — список величин, определяющих ячейки памяти, куда вводится информация. Список оператора READ устроен аналогично списку оператора PRINT. Например, запись

READ 1, A (I), C, T (8, 15), J
1 FORMAT (3F 12.4, I 10)

означает, что информация вводится в ячейки памяти, отведенные для размещения величин A (I), C, T (8, 15), J. Оператор FORMAT в данном случае определяет способ размещения вводимых чисел на перфокартах.

3. Оператор PUNCH. Оператор PUNCH, используемый сравнительно редко, позволяет выводить информацию на перфокарты с помощью итогового перфоратора. Карты перфорируются в кодировке CDC (см. рис. 1, стр. 43). Расположение информации на перфокартах задается оператором FORMAT. Например,

PUNCH 35, A (5), L
35 FORMAT (F 15.4, I 6)

4. Операторы обмена информацией с внешними запоминающими устройствами. Эти операторы разделяются на две группы: операторы ввода (чтения) и операторы вывода (записи). Для каждого оператора ввода или вывода возможны два варианта — форматный и бесформатный.

Форматный оператор ввода или *вывода* позволяет при обмене преобразовывать информацию в соответствии со спецификациями формата. *Бесформатный оператор ввода* или *вывода* означает обмен информацией в ее машинном (двоичном) представлении. Если информация, записываемая на магнитную ленту, предназначена для использования лишь на той же (или однотипной) машине, то предпочтительнее бесформатный обмен, который выполняется быстрее, чем форматный, и требует меньше места во внешней памяти.

Форматный обмен обычно необходим тогда, когда информация, записанная во внешнюю память на одной ЭВМ, должна быть использована на другой (не однотипной) ЭВМ.

Поскольку магнитофоны имеются на ЭВМ всех типов, а магнитные барабаны или диски далеко не на всех, то рассматриваемые операторы ввода-вывода рассчитаны на работу с магнитной лентой. При наличии на данной ЭВМ магнитных барабанов или дисков можно работать и с этими устройствами. Однако в этом случае структура оператора ввода-вывода не меняется, т. е. пользователь работает как бы с лентой.

Рассмотрим структуру записи информации на магнитную ленту. Каждый массив информации, записанный с помощью одного оператора вывода, образует одну или несколько логических единиц записи. В случае бесформатного оператора вывода образуется только одна логическая единица записи. Она подразделяется на массивы длиной по 24 машинных слова (физические рекорды). При записи к каждому рекорду приформируются по два служебных слова. В каждую зону МЛ (МД) или в каждый тракт МБ записывается по 39 таких рекордов, состоящих из 26 машинных слов. При этом емкость одной МЛ составляет 512 зон, а емкость одного МБ — 32 тракта.

В процессе записи магнитная лента перематывается в прямом направлении, останавливаясь перед началом участка, предназначенного для размещения следующей логической единицы записи. При чтении (вводе) лента также перематывается в прямом направлении. Возврат ленты производится специальными операторами либо на начало записи, т. е. на начало первой логической

единицы записи, либо на одну логическую единицу записи назад.

При чтении должно соблюдаться правило: количество читаемых машинных слов не должно превышать длины логической единицы записи, т. е. количества машинных слов, в нее записанных. Если читается не вся единица записи, а только ее часть, то лента все равно перематывается на начало следующей единицы записи.

Таким образом, при работе операторов ввода-вывода магнитная лента используется как устройство последовательного доступа. Для считывания или записи информации необходимо предварительно установить ленту в нужное положение, т. е. на начало того участка, на котором расположена считываемая информация или на который производится ее запись. Отметим, что для каждой магнитной ленты может быть установлен один из двух режимов использования: запись и чтение или только чтение (см. § 8).

Каждому устройству, с которым производится обмен информацией, присвоен определенный, так называемый логический номер. На БЭСМ-6 магнитофонам (накопителям на магнитных дисках) присвоены логические номера с 1 по 15, магнитному барабану — 16, читающему устройству — 50, печатающему устройству — 51, итоговому перфоратору — 52. На машинах других типов распределение логических номеров устройств может быть иным.

Логический номер означает, что реальное устройство, имеющее данный логический номер, может быть любым из однотипных устройств (например, магнитофонов). Соответствие логических номеров реальным устройствам устанавливается диспетчером, распределяющим эти устройства между параллельно решаемыми задачами (см. § 7). Пользователь должен только знать, что его задача (написанная на фортране) может использовать не более одного АЦПУ, не более одного выводного перфоратора, не более 15 магнитофонов и не более одного магнитного барабана.

Использование накопителей на магнитных дисках вместо магнитофонов организуется путем запроса их в управляющих картах пакета задачи (см. § 8).

5. Бесформатный оператор вывода. Этот оператор имеет вид

WRITE (*i*) L

Здесь *i* — логический номер устройства вывода, заданный в виде целой константы или простой целой переменной; L — список выводимых величин, имеющий ту же структуру, что и для оператора PRINT. Данный оператор передает информацию устройству вывода с номером *i* из тех мест оперативной памяти, которые определяются элементами списка L.

П р и м е р. Оператор

WRITE (5) A, R(7,6), (C(I), I = 1, 20)

означает запись на магнитную ленту, установленную на магнитофоне с логическим номером 5 переменной A, элемента массива R(7,6) и 20 первых элементов массива C. Всего будет записано 22 машинных слова (в предположении, что все элементы списка не типа COMPLEX или DOUBLE PRECISION), которые образуют одну логическую единицу записи на ленте.

6. Форматный оператор вывода. Этот оператор имеет вид

WRITE (*i*, *n*) L

Здесь *i* — логический номер устройства вывода; *n* — метка оператора FORMAT, по которому производится преобразование выводимой информации; L — список элементов, подлежащих выводу. Например, операторы

WRITE (NT, 20) X, Z, (T(I), I=1, 3), A (18)
20 FORMAT (6E15.8)

означают вывод информации на устройство вывода с заранее определенным номером NT с преобразованием выводимой информации посредством оператора FORMAT с меткой 20.

В данном случае произойдет вывод одной физической единицы записи, которая содержит 144 символа (из них 90 заданы форматом) и соответствует одной логической единице записи. Отметим, что для магнитной ленты длина физической единицы записи равна 144 символам, т. е. 24 машинным словам. Если $1 \leq NT \leq 15$, то произойдет запись на МЛ (или в файл на МД). При NT = 16 информация запишется на магнитный бара-

бан. При $NT = 51$ произойдет вывод информации на печать, т. е. указанный оператор будет эквивалентен оператору PRINT. Если $NT = 52$, то произойдет вывод на перфокарты, т. е. то же, что и по оператору PUNCH. Заметим, что в некоторых версиях фортрана операторы PRINT и PUNCH отсутствуют и вывод на печать или на перфорацию производится оператором WRITE с определенными логическими номерами устройств вывода (отличными, вообще говоря, от 51 и 52).

7. Бесформатный оператор ввода. Общий вид этого оператора

$$\text{READ } (i) \text{ L}$$

где i — номер устройства ввода; L — список, определяющий места в памяти машины, куда производится ввод информации. Ввод информации производится в пределах одной логической единицы записи, полученной ранее путем вывода по бесформатному оператору WRITE. Список L имеет структуру, аналогичную списку оператора PRINT.

8. Форматный оператор ввода. Этот оператор имеет вид

$$\text{READ } (i, n) \text{ L,}$$

где i — номер устройства ввода; n — метка оператора FORMAT, в соответствии с которым преобразуется вводимая информация; L — список величин, определяющих ячейки памяти, куда вводится информация. При $i = 50$ будет произведен ввод с перфокарт.

9. Операторы управления магнитной лентой. В этом пункте i означает номер магнитофона (или накопителя на МД, заказанного пользователем вместо этого магнитофона).

Оператор REWIND i устанавливает магнитную ленту на устройство с номером i к началу участка (файла). Если магнитная лента (или диск) была уже установлена в начало, то данный оператор не производит никакого действия.

Оператор BACKSPACE i возвращает ленту на i -м магнитофоне на одну логическую единицу записи. Если лента установлена в начало, то данный оператор не производит никакого действия. То же самое происходит с диском.

Оператор END FILE *i* записывает признак конца файла, т. е. участка магнитной ленты (или диска), на который произведена запись информации. Такая запись позволяет контролировать процесс чтения, не допуская выхода за пределы участка, ограниченного признаком конца.

Отметим, что для пропуска единиц записи в прямом направлении движения ленты можно использовать оператор READ (*i*) с пустым списком. В этом случае можно использовать также некоторые библиотечные программы (см. [3], программа K405).

Для формирования очередного файла и для чтения информации со следующего файла необходимо найти признак конца предыдущего файла. Для поиска признака конца файла на устройстве с логическим номером *i* можно написать обращение

CALL SCHEOF (*i*)

Поиск конца файла производится в прямом направлении. По окончании поиска лента (диск) устанавливается в начало следующего файла, если таковой имеется.

Во многих случаях вместо операторов ввода-вывода удобнее использовать комплекс программ, позволяющих работать с внешними накопителями информации на БЭСМ-6 как устройствами прямого доступа (см. [5]).

§ 29. Оператор FORMAT

Оператор FORMAT позволяет производить преобразование информации из одной формы ее представления в другую. Такое преобразование обычно необходимо при вводе информации в машину с внешних носителей и при выводе информации из машины на внешние носители. Преобразование информации с помощью оператора FORMAT можно производить и внутри оперативной памяти посредством операторов ENCODE и DECODE (см. § 30).

Здесь будут рассмотрены структура и функции оператора FORMAT применительно к операторам ввода-вывода. Однако все, что здесь сказано, в равной степени относится и к операторам ENCODE и DECODE.

Оператор FORMAT содержит информацию, необходимую для работы операторов ввода-вывода. Один

и тот же оператор FORMAT может быть использован в нескольких операторах ввода-вывода. Оператор FORMAT является невыполняемым оператором. Он может располагаться в любом месте подпрограммы, но после всех декларативных операторов (COMMON, DIMENSION и др.) и после всех функций-операторов. На БЭСМ-6, кроме того, необходимо, чтобы первому из операторов FORMAT предшествовал хотя бы один выполняемый оператор.

Каждый оператор FORMAT обязан иметь метку. На БЭСМ-6 транслятор отличает метки операторов FORMAT от одноименных меток выполняемых операторов. Например, допускается запись вида

```
4 A=0.  
4 FORMAT (F15.2)
```

Однако указанное свойство меток имеет место не для всех версий фортрана.

В процессе трансляции на БЭСМ-6 все операторы FORMAT выносятся в конец подпрограммы.

Информация, содержащаяся в операторе FORMAT, представляет собой список спецификаций, разделенных запятыми. Перед спецификациями или группами спецификаций может быть указано число повторений в виде целой константы. Список спецификаций оператора FORMAT может быть сформирован в процессе счета. Такой оператор называется *переменным оператором* FORMAT (см. п. 9).

Спецификации в операторе FORMAT могут быть двух видов: спецификации преобразования и редакционные спецификации. Каждая *спецификация преобразования* вызывает перекодировку информации для соответствующего ей элемента списка оператора ввода-вывода. *Редакционные спецификации* вызывают некоторые дополнительные действия, например, пропуск нескольких позиций на бумажной ленте, печать текста, заданного непосредственно в операторе FORMAT, переход на новую строку и т. д.

Каждая спецификация преобразования определяет форму представления информации на внешних носителях. Запись каждой такой спецификации состоит из одной буквы и нескольких цифр, внутри которых

может находиться десятичная точка. Например, F15.2, I6, A4.

Цифры, указанные после наименования спецификации, определяют ширину поля формата, отведенного на внешнем носителе информации для размещения данного элемента информации. Применительно к оператору PRINT ширина поля формата — это количество позиций в строке бумажной ленты (листинга), отведенных для размещения соответствующего элемента списка оператора PRINT. При вводе с перфокарт по оператору READ под шириной поля формата понимается количество колонок перфокарты, в которых пробито вводимое число или вводимый текст.

В некоторых спецификациях преобразования после записи ширины поля формата ставится десятичная точка, справа от которой указывается ширина части поля формата, отводимого для размещения дробной части числа.

П р и м е р ы использования спецификаций преобразования. Операторы

PRINT 20, A, K, NT
20 FORMAT (F12.3, I9, A6)

означают, что значение вещественной переменной A будет преобразовано в форму десятичного числа с тремя знаками в дробной части и расположено в первых 12 позициях строки листинга. Значение целой переменной K будет преобразовано в целое десятичное число и расположено в следующих девяти позициях строки. Значение целой переменной NT будет рассматриваться как последовательность текстовых символов и располагаться в следующих шести позициях строки.

Оператор

READ 20, T(18), LP, KX(J)

с тем же оператором FORMAT, что и выше, указывает, что значение переменной T(18) пробито в первых 12 колонках перфокарты в виде десятичного числа с плавающей запятой; значение переменной LP занимает на перфокарте следующие 9 колонок и пробито в виде целой константы; значение переменной KX(J) расположено в последующих 6 колонках перфокарты в виде последовательности текстовых символов, которые бу-

дут представлены в машине в кодировке ISO (см. приложение 2).

В дальнейшем ширина поля формата будет обозначаться буквой *w*, а ширина части поля формата, отведенной для размещения дробной части числа, — буквой *d*. Рассмотренные выше спецификации будут иметь общий вид *Fw.d*, *Iw*, *Aw*. Иногда мы будем называть эти спецификации просто *F*, *I*, *A*.

В фортране на БЭСМ-6 используются следующие спецификации преобразования:

Ew.d — десятичное число с плавающей запятой (без двойной точности) с порядком,

Fw.d — десятичное число с плавающей запятой (без двойной точности) без порядка,

Dw.d — десятичное число с плавающей запятой с двойной точностью с порядком,

Iw — десятичное целое число,

Lw — логическая величина,

Aw — текстовая величина,

Ow — восьмеричная величина.

В процессе ввода или вывода информации каждой из указанных спецификаций ставится в соответствие один из элементов списка оператора ввода-вывода. При этом должно соблюдаться определенное соответствие между типом элемента списка оператора ввода-вывода и той спецификацией, по которой производится преобразование этого элемента.

Спецификациям *E* и *F* при вводе и выводе и спецификации *I* только при выводе могут соответствовать элементы списка оператора ввода-вывода, имеющие вещественный, целый или комплексный тип; спецификации *D* при вводе и выводе — только элементы типа с двойной точностью; спецификации *I* при вводе — только элементы целого типа; спецификации *L* при вводе и выводе — только элементы логического типа.

По спецификации *A* можно преобразовывать любые целые величины. Однако они должны иметь вид текстовых величин. При преобразовании по спецификации *O* тип преобразуемой величины может быть любым, кроме типа **DOUBLE PRECISION**.

1. Спецификации преобразования при выводе. Рассмотрим на конкретных примерах использование спецификаций преобразования при выводе.

Пр и м е р 1. Пусть написаны операторы

```
COMPLEX Z (2)  
DOUBLE PRECISION DP  
PRINT 25, A, Z, DP, KT  
25 FORMAT (F8.4, F10.6, F10.6, E8.3, E9.1, D25.15, I5)
```

Здесь выводятся на печать величины четырех типов, имеющих машинное представление в виде чисел. При этом величины типов COMPLEX и DOUBLE PRECISION, занимающие каждая по два машинных слова, выводятся по-разному. Величина с двойной точностью выводится как единое целое по спецификации *Dw.d*. Величина комплексного типа выводится покомпонентно, т. е. отдельно действительная и отдельно мнимая часть, каждая по своей спецификации вида *Fw.d* или *Ew.d* с разными, вообще говоря, значениями *w* и *d*.

Рассмотрим каждую спецификацию в отдельности. Но прежде сформулируем несколько общих правил, справедливых для всех спецификаций формата при выводе.

При выводе по каждой спецификации в процессе преобразования выводимой величины из внутреннего (машинного) представления во «внешнее» происходит заполнение поля формата, отведенного для размещения преобразованной величины. Это заполнение производится справа налево для любой спецификации преобразования. В случае, когда ширина поля превышает число символов, необходимых для представления преобразованной величины, избыточные левые позиции поля заполняются пробелами.

Пусть для определенности в примере 1 выводимые величины имеют следующие значения:

```
A : 3.14159,  
Z : -2.7182818, 1374.1598374, 75431.1985, -0.000474,  
DP : 31.4159 26535 89793 2D0  
KT : 1975
```

При выводе величины *A* две левых позиции поля заполнятся пробелами, а младшая цифра результата будет округлена, так что получится — 3.1416

При выводе первой величины из комплексного массива *Z* получится —2.718282

Для вывода второго числа с 6 знаками в дробной части потребуется 11 позиций. Но ширина поля равна 10 позициям. В этом случае правые позиции поля будут заполняться выводимыми символами, а когда заполнение дойдет до самой левой позиции поля и выяснится, что ширины поля не хватает, то в эту позицию будет помещен символ *. Таким образом, второе число из массива Z будет представлено в виде

* 74.159837

Рассмотренные примеры показывают, что при выводе по спецификации F необходимо знать хотя бы приблизительный порядок выводимых чисел. Иначе есть риск, что поля формата может не хватить.

Следующее число выводится по спецификации E8.3. При использовании спецификации E восьми позиций поля будет достаточно для вывода числа 75431.1985. Отметим, что при выводе этого числа по спецификации F с тремя знаками после точки потребовалось бы девять позиций (знак + не выводится) и ширины поля не хватило бы.

При выводе по спецификации $Ew.d$ позиции поля формата распределяются следующим образом: две правых позиции отводятся под десятичный порядок, следующая позиция, — при счете справа налево, — под знак порядка, затем d позиций под дробную часть, одна позиция под десятичную точку и одна позиция для цифры слева от точки. Еще левее располагается знак числа, если выводимое число отрицательное. Таким образом, при выводе по спецификации $Ew.d$ ширина поля w должна быть не менее $d+5$ для вывода положительных чисел и не менее $d+6$ для вывода отрицательных чисел. Наше число будет выдано в виде 7.543+04

Итак, при выводе по спецификации $Ew.d$ число позиций, необходимое для размещения выводимого числа, не превосходит $d+6$ независимо от величины выводимого числа. Здесь производится вывод $d+1$ старшей цифры числа, причем самая старшая цифра выводится слева от точки. Десятичный порядок (со знаком + или —) печатается справа от мантиссы и занимает вместе со знаком три позиции.

При выводе следующего числа по спецификации E9.1 потребуется семь позиций, так что левые две

позиции будут пробелами, а выведенное число будет иметь вид `___-4.7-04`.

Если бы здесь вместо спецификации E9.1 была использована спецификация F9.1, то при выводе мы не получили бы ни одной значащей цифры числа, которое было бы представлено в виде: `_____ -0.0`

Вывод по спецификации D аналогичен выводу по спецификации E с той разницей, что под порядок отводится три позиции вместо двух. Выводимое здесь число будет представлено в виде `_____ 3.141592653589793+001`. Последнее число из списка оператора PRINT будет выведено так: `_ 1975`

Отметим, что при выводе по спецификации I при нехватке ширины поля избыточные левые цифры выводимого числа пропадают и при этом символ * не выдается. Отметим также, что если при выводе по спецификации Fw.d указано $d = 0$ или d отсутствует, то при выводе на печать десятичная точка не печатается. По такой спецификации иногда удобно выводить целые величины, так как в этом случае при нехватке поля в левой позиции будет печататься *.

Рассмотрим теперь спецификации L, A и O. Обращаем внимание читателя на то, что спецификация L используется очень редко, а при выводе по спецификациям A и O формат результата зависит от типа машины.

При выводе по спецификации Lw логическое значение .TRUE. или .FALSE. преобразуется в последовательность $w-1$ пробелов, за которыми следует буква T или F соответственно.

Спецификация Aw используется для вывода текстовых величин. На БЭСМ-6 в каждое слово записывается по 6 символов. Если в спецификации Aw указано $w \geq 6$, то будут выведены все 6 символов, а избыточные левые позиции будут заполнены пробелами. Если же $w < 6$, то будут выведены только левые w символов соответствующего машинного слова. Выводимая величина должна иметь целый тип.

Пр и м е р 2. Пусть написаны операторы

```
DATA NA, NB/'BESM-6', '_DUBNA'/  
PRINT 20, NA, NB  
20 FORMAT (A6, A6)
```


После вывода напечатается текст
BESM-6_DUBNA

Спецификация O используется в основном при отладке программ, когда необходимо знать точное машинное представление некоторых величин.

На БЭСМ-6 слово содержит 16 восьмеричных цифр. При $w < 16$ выводятся w младших восьмеричных разрядов, при $w \geq 16$ выводится все слово, причем избыточные левые позиции заполняются пробелами.

По данной спецификации можно выводить величины любых типов, кроме типа DOUBLE PRECISION. Для вывода величин с двойной точностью необходимо описать каждую часть этой величины отдельно, что делается путем эквивалентности. Например,

```
DOUBLE PRECISION D
DIMENSION R (2)
EQUIVALENCE (D, R)
PRINT 30, R
30 FORMAT (O18, O 18)
```

Сформулируем еще раз

*Основные правила использования спецификаций
преобразования при выводе*

1. При выводе по спецификациям $Ew.d$, $Fw.d$, $Dw.d$, Iw , Lw , Aw , Ow поле формата шириной w позиций заполняется справа налево. При этом избыточные левые позиции поля заполняются пробелами.

2. При выводе по спецификации $Fw.d$ ширина поля w , требуемая для размещения числа, зависит от величины целой части выводимого числа. Если модуль выводимого числа велик, то может не хватить ширины поля w . Если выводимое число мало, т. е. порядка 10^{-n} ($n \geq 1$), то для получения хотя бы одной значащей цифры необходимо задать $d \geq n$.

3. При выводе по спецификациям $Ew.d$ и $Dw.d$ ширина поля, требуемая для размещения числа, не зависит от его величины. При выводе по спецификации $Ew.d$ достаточно ширины поля $w = d+6$, а при использовании спецификации $Dw.d$ достаточно поля шириной $w = d+7$. В обоих случаях выводятся $d+1$ старших цифр числа. На БЭСМ-6 не имеет смысла указывать

$d > 11$ в спецификации $Ew.d$ и $d > 23$ в спецификации $Dw.d$, так как точность машинного представления соответствующих величин не может быть более 12 и 24 десятичных знаков.

4. При нехватке поля формата в спецификациях $Ew.d$, $Fw.d$ и $Dw.d$ $w-1$ правых позиций поля заполняются выводимыми символами, а в левой позиции поля печатается символ *.

5. При выводе по спецификациям Iw и Ow выводятся w младших цифр выводимой величины, а при выводе по спецификации Aw выводятся w левых символов текстовой величины. На БЭСМ-6 число символов, выводимых по спецификациям Iw , Ow и Aw , не превосходит 13, 16 и 6 соответственно (сюда не входят пробелы в левых позициях поля).

6. При выводе по спецификациям E , F и D производится округление младшей цифры выводимого числа.

2. Спецификации преобразования при вводе. Рассмотрим теперь спецификации преобразования при вводе информации с перфокарт. Прежде чем перейти к рассмотрению конкретных примеров, опишем общие правила, справедливые для всех спецификаций преобразования при вводе.

При вводе с перфокарт играет роль лишь ширина поля w , равная числу колонок перфокарты, в которых размещена вводимая величина. Поэтому при вводе по спецификациям $Ew.d$, $Fw.d$ и $Dw.d$ можно указывать любое d такое, что $0 \leq d \leq w$.

Информация, пробитая на перфокарте, может содержать различные символы: цифры, буквы, знаки и т. д. Для каждой спецификации преобразования существует набор допустимых символов, которые могут появиться в w колонках поля формата. В случае обнаружения недопустимых символов производится выдача диагностики (см. § 31, п. 6).

При вводе по спецификациям E , F , D , I , O пробелы, обнаруженные в поле формата, на машинах многих типов интерпретируются как нули. Это означает, что если вводимое число заполняет не все поле, то оно должно располагаться в правых колонках этого поля. Такое ограничение не всегда удобно.

На БЭСМ-6 при вводе по указанным спецификациям пробелы в поле формата игнорируются. Это означает,

что вводимые символы могут располагаться в поле формата произвольным образом.

При пробивке перфокарт на устройствах с покомпонентной кодировкой удобно располагать символы в левых колонках поля формата. Это обусловлено тем, что на большинстве таких устройств имеется возможность указать разметку полей на перфокарте (чаще всего с помощью специальной карты-шаблона). При пробивке данных очередные символы перфорируются в начальных колонках отведенного для них поля формата. По окончании пробивки всех символов нажимается клавиша пропуска (SKIP), в результате чего пропускаются неиспользованные правые колонки поля. То обстоятельство, что на БЭСМ-6 при вводе чисел пробелы в поле ввода игнорируются, позволяет в полной мере использовать указанную возможность.

Пример 3. Пусть написаны операторы

```
COMPLEX Z (2)
DOUBLE PRECISION DP
READ 25, A, Z, DP, KT
25 FORMAT (F8.4, F10.6, F10.6, E8.3, E9.1, D25.15, I5)
```

Эти операторы аналогичны рассмотренным в примере 1 с той разницей, что здесь производится ввод, а не вывод. Первое из вводимых чисел может располагаться произвольным образом в колонках 1÷8 перфокарты. При этом в записи числа допускаются десятичные цифры, буква E, не более одной точки и не более двух знаков + или —. Например, в качестве A может выступать одно из чисел

—123.76, +78.15E—2, 115, 321.

Таким образом, при вводе по спецификации Fw.d вводимое число может содержать любые символы, допустимые для вещественной или целой константы. При этом все символы должны быть расположены в пределах w колонок поля формата. Если при этом десятичная точка отсутствует, то при вводе производится умножение вводимого числа на 10^{-d} , так что число 115 по спецификации F8.4 будет введено в машину как 115.E—4.

В следующих двух полях по 10 колонок в каждом будут содержаться два числа, каждое из которых

будет преобразовано по спецификации F10.6 и записано в качестве значения действительной и мнимой части комплексной величины Z(1).

Следующее вводимое число преобразуется по спецификации E8.3. Здесь эта спецификация аналогична спецификации F8.3 с той разницей, что при отсутствии десятичной точки в записи вводимого числа оно не будет умножаться на 10^{-3} . То же самое справедливо и для спецификации E9.1.

При вводе по спецификации D25.15 будет проверяться наличие буквы D в записи вводимого числа, т. е. вводимое число должно иметь вид константы с двойной точностью.

При вводе по спецификации I5 у вводимого числа допускаются только десятичные цифры и не более одного знака + или — перед старшей цифрой.

Рассмотрим кратко остальные спецификации при вводе.

При вводе по спецификации Aw вводимая текстовая величина может содержать любые символы, допустимые в фортране (см. приложение 2). Здесь пробелы уже не игнорируются, а учитываются наравне с остальными символами. На БЭСМ-6 при $w > 6$ вводятся последние 6 символов поля ввода, а при $w \leq 6$ вводятся все символы, расположенные в w колонках поля. Ячейка памяти, соответствующая вводимой текстовой величине, заполняется слева направо. При $w < 6$ в младшие разряды этой ячейки будет записано недостающее число пробелов.

При вводе по спецификации Ow в поле ввода могут быть лишь восьмеричные цифры, которым может предшествовать знак + или —. На БЭСМ-6 при $w \geq 16$ вводятся последние 16 цифр поля ввода. При $w \leq 16$ вводимая величина заполняет w младших восьмеричных разрядов машинного слова, старшие разряды полагаются равными нулю. Отрицательные числа представляются в машине в обратном коде. Например, вводимое число —376В будет представлено в машине как

7777 7777 7777 7401В

Отметим, что данная спецификация обычно используется при отладке и при работе автокодных подпрограмм.

Спецификация *Lw* используется при вводе достаточно редко. В поле ввода в качестве первого, отличного от пробела символа должна быть буква T или F, которым соответствуют значения вводимых величин .TRUE. и .FALSE. соответственно. Поле ввода, состоящее из одних пробелов, интерпретируется как .FALSE.

Отметим одно важное обстоятельство, связанное с вводом информации. Информация, пробитая на перфокартах, может оказаться расположенной не в тех колонках, которые соответствуют полям формата, указанным в операторе FORMAT. В случае такого несоответствия ввод будет произведен неверно, причем диагностика ошибок будет выдана не всегда. Пусть, например, ввод производится согласно оператору

20 FORMAT (F9.3, E7.2, F10.3),

а на перфокарте пробито

6.7415E1/236.15 /6318.E—7

Здесь знак / отделяет числа, которые пользователь собирается ввести по указанному формату. В результате несоответствия полей первое число введется как 6.7415E12, второе — как 36.156 и третье — как 318.E—7. Если на карте пробито

6.7415E1/—236.15/+6318.E—7/

то при вводе по указанному формату будет выдана диагностика ошибки (см. § 31, п. 6), поскольку числа будут вводиться как

6.7415E1—, 236.15+, 6318.E—7

При вводе с МЛ и МБ происходит аналогичный процесс с той лишь разницей, что вводимые числа были ранее записаны туда по правилам вывода информации по спецификациям *Fw.d* и *Ew.d*. Несоответствие форматов при записи и чтении вызывает ситуации, аналогичные описанным. Сформулируем

Основные правила преобразования информации при вводе

1. При вводе информации учитывается лишь ширина поля формата *w*. В спецификациях *Ew.d*, *Fw.d* и *Dw.d* значение *d* не существенно.

2. Для каждой спецификации преобразования в поле ввода могут присутствовать лишь символы, допустимые для данной спецификации.

3. На БЭСМ-6 при вводе по спецификациям E, F, D, I, O пробелы в поле ввода игнорируются. Это позволяет располагать вводимую информацию в левых колонках поля ввода, что удобно при пробивке перфокарт (на устройствах с поколонной кодировкой).

3. Редакционные спецификации. Рассмотрим теперь редакционные спецификации. Этим спецификациям нет соответствующих элементов в списке оператора ввода-вывода. Редакционные спецификации позволяют располагать выводимую информацию в виде, наиболее удобном для чтения, а при вводе информации — выполнять дополнительные действия, связанные с выборкой этой информации.

Спецификация *wX* позволяет при выводе информации расположить между двумя соседними полями *w* пробелов. Пусть, например, написаны операторы

```
PRINT 20, A, I, C
20 FORMAT (F8.3, 5X, I4, 2X, E10.2)
```

и пусть $A = -136.1987$, $I = 121$, $C = 286.107$

В результате будет напечатано

```
-136.199 _____ 121 _____ 2.86+02
```

Здесь между первым и вторым числом получилось 6 пробелов: из них 5 за счет спецификации 5X и один за счет избыточной левой позиции в спецификации I4. Между вторым и третьим числом оказалось 5 пробелов: 2 за счет спецификации 2X и 3 за счет избыточных левых позиций в спецификации E10.2.

При вводе спецификация *wX* вызывает пропуск очередных *w* колонок перфокарты. Пусть, например, написаны операторы

```
READ 11, K, P, T
11 FORMAT (I2, 2X, F5.1, 5X, F6.2)
```

и вводимая перфокарта, начиная с первой колонки, содержит символы

```
15__-39.6SINF165.82
```

После ввода в К, Р и Т занесутся числа

15, —39.6 и 165.82

а символы SINFI не вводятся.

При выводе и вводе заголовков часто используется спецификация *w*N. Эта спецификация позволяет вывести любой набор символов, в том числе и пробелов, заданных непосредственно в операторе FORMAT. Буква N служит признаком текстового поля, при этом *w* означает количество символов, следующих за буквой N. Запятая после спецификации *w*N не обязательна.

Примеры.

```
PRINT 10
10 FORMAT (22N ТАБЛИЦА ЗНАЧЕНИЙ F (X))
PRINT 20, A
20 FORMAT (10X, 2NA=F6.1)
```

Спецификация *w*N при вводе заносит на свое поле в операторе FORMAT, т. е. в следующие *w* позиций за буквой N, вводимую текстовую информацию из двух символов A=. Пусть, например, написаны операторы

```
READ 20
20 FORMAT (17N <17 пробелов>)
```

и вводимая перфокарта содержит следующие 17 символов:

FORTRAN — AT — BESM-6

При вводе в оператор FORMAT после буквы N занесутся 17 символов с перфокарты. При последующем использовании данного оператора FORMAT в операторе вывода эти символы будут выведены. Например, оператор

```
PRINT 20
```

отпечатает строку

FORTRAN — AT — BESM-6

На БЭСМ-6 взамен спецификации *w*N часто используется редакционная спецификация, которая имеет вид '*<n позиций>*'. Данная спецификация является видоизменением спецификации *w*N. При использовании ее в операторе вывода можно вывести любые символы (кроме апострофов), заключенные между апострофами.

Первый апостроф означает начало текстового поля, второй — окончание текстового поля. Эта спецификация удобнее *wH* при выводе длинных текстов, так как не требует подсчета числа выводимых символов. Однако эта спецификация, в отличие от *wH*, может использоваться не на всех типах машин.

Н а п р и м е р, после выполнения операторов

```
PRINT 10  
10 FORMAT (30X 'ТАБЛИЦА _ F (X)')
```

будет отпечатан текст, заключенный в апострофы. В операторах

```
PRINT 20  
20 FORMAT ('DUBNA' BESM-6')
```

внутри текста оказался апостроф, который ограничит текстовое поле. Будет напечатано: DUBNA. Затем программа управления форматом попытается интерпретировать последующие символы как спецификацию, что приведет к сообщению об ошибке.

Эта же спецификация может быть использована как видоизменение спецификации *wH* при вводе нового заголовка на поле, отведенное внутри оператора FORMAT. В заголовок заносятся символы из вводимой перфокарты до тех пор, пока не заполнятся все позиции поля или во вводимой информации не встретится апостроф. В последнем случае в поле ввода будут занесены символы, расположенные левее апострофа, а избыточные правые позиции поля заполняются пробелами.

Пусть, н а п р и м е р, написаны операторы

```
READ 10  
10 FORMAT ('____')
```

и вводимая перфокарта, начиная с первой колонки, содержит текст: BESM-6

При последующем использовании данного оператора FORMAT в операторе вывода, например, PRINT 10 будет напечатано: BESM-6

Рассмотрим следующую редакционную спецификацию, которая обозначается знаком / (как и операция деления).

4. Новая единица записи. Знак / обозначает конец единицы записи. Он может появиться где угодно в спи-

ске спецификаций оператора FORMAT и нет необходимости отделять его запятыми от других элементов в списке этого оператора. Допускается написание нескольких знаков / подряд.

Единицей записи на БЭСМ-6 является:

- 1) одна полная строка бумажной ленты на печатающем устройстве (128 позиций),
- 2) одна перфокарта (80 колонок),
- 3) один физический рекорд (144 символа),
- 4) один или несколько физических рекордов, составляющих одну логическую единицу записи (см. § 28, п. 4),
- 5) массив ячеек памяти, содержащий определенное число символов, указанное в операторе ENCODE или DECODE (см. § 30).

При выводе на печать знак / используется для перехода к следующей строке, а при вводе с перфокарт — для перехода к следующей карте. Например, после выполнения операторов

```
PRINT 10, A, B, C
10 FORMAT (2HA=F15.3/2HB=F15.3/2HC=F15.3)
```

каждая из величин будет напечатана с новой строки с соответствующим заголовком. А например, пара операторов

```
READ 20, A, B, C
20 FORMAT (E12.3/F10.2, F15.5)
```

означает, что сначала вводится величина A, пробитая на первой перфокарте по спецификации E12.3, затем переходим ко второй карте, на которой пробиты значения величин B и C по спецификациям соответственно F10.2 и F15.5.

5. Масштабный коэффициент. Спецификациям F, E и D может предшествовать масштабный коэффициент nP . Он употребляется со спецификацией $Fw.d$ при вводе и выводе, а со спецификациями $Ew.d$ и $Dw.d$ только при выводе. Спецификации с масштабным коэффициентом записываются следующим образом:

$$nP Fw.d, nP Ew.d, nP Dw.d,$$

где n — целая константа со знаком. Масштабный коэффициент nP означает, что при вводе вводимое число

умножается на 10^{-n} , а при выводе происходит умножение выводимого числа на 10^n . Например, если на карте пробито число 3.141592 и оно вводится по спецификации 2PF8.4, то в память заносится 3.141592E-2. При вводе этого числа по спецификации —3PF8.4 в памяти окажется 3.141592E3.

При выводе по спецификации *nPFw.d* происходит умножение выводимого числа на 10^n . Если, например, число 3.141592 выводится по спецификации 1P F8.3, то будет выдано 31.416 (с округлением третьей цифры дробной части). Вывод по спецификации —2PF8.3 даст в результате 0.031.

При выводе по спецификации *nPEw.d* и *nPDw.d* происходит сдвиг десятичной точки на *n* позиций (вправо при $n > 0$ и влево при $n < 0$) с соответствующим изменением порядка. Например, вывод числа 3.141592 по спецификации 2PE10.3 даст в результате 314.159—02, а после вывода по спецификации —1PE10.3 напечатается 0.314+01. Аналогичные результаты, только с тремя цифрами у порядка, получатся при выводе по спецификациям 2PD10.3 и —1PD10.3.

Если масштабный коэффициент не указан, что чаще всего и бывает, то он полагается равным 0. Однако, будучи указан один раз, он сохраняет силу для всех следующих за ним спецификаций F, E и D в данном операторе FORMAT. Для того чтобы отменить действие масштабного коэффициента на последующие спецификации F, E и D, необходимо перед одной из последующих спецификаций F, E или D указать нулевой масштабный коэффициент 0P. Масштабные коэффициенты при спецификациях E и D при вводе игнорируются.

6. Повторяемые спецификации. Любую спецификацию оператора FORMAT можно повторить, указав перед ней число повторений в виде целого числа без знака. Например, запись

40 FORMAT (3F10.3, 2I2)

эквивалентна записи

40 FORMAT (F10.3, F10.3, F10.3, I2, I2)

Можно повторить и группу спецификаций, которая в этом случае заключается в скобки, а перед скобками

указывается число повторений. Например, запись

20 FORMAT (3F8.2, A6, 3F8.2, A6)

можно представить в виде

20 FORMAT (2(3F8.2, A6))

Повторяемая группа спецификаций может иметь внутри себя другие повторяемые группы спецификаций. Однако глубина вложения скобок в повторяемых группах спецификаций не должна превосходить 3.

Например, запись

20 FORMAT (2(3X, 2(F5.1, I3, 3(F4.1, 2X))))

является допустимой, а запись

20 FORMAT (2(3X, 2(F5.1, I3, 3(F4.1, 2(2X, I4)))))

уже не допустима.

7. Управление расположением строк при печати.

При выводе информации на печатающем устройстве символ, попадающий в первую позицию каждой строки, используется для управления расположением строк. Приведем список управляющих символов на БЭСМ-6:

0 — пропуск одной пустой строки;

1 — переход на новую страницу;

4 — переход на новую половину страницы;

5 — переход на новую треть страницы;

6 — переход на новую четверть страницы;

+ — печать без перевода строки, т. е. повторная печать на той же строке (с наложением символов);

S — блокировка автоматического перехода на новую страницу во время печати данной строки.

Здесь каждая страница содержит 66 строк, каждая половина страницы — 32 строки, каждая треть страницы — 21 строку и каждая четверть страницы — 15 строк.

При печати происходит автоматический перевод страницы после каждых 66 строк. В некоторых случаях, например, при печати графиков, гистограмм и т. п. необходимо получить непрерывную выдачу. Для этого в первую позицию каждой строки ставят символ S.

Управляющий символ не печатается. Если в первой позиции оказался символ, отличный от управляю-

шего, то он печатается обычным образом. Отметим, что на машинах других типов список управляющих символов может быть иным либо управляющие символы могут отсутствовать.

П р и м е р ы:

а) PRINT 1
1 FORMAT (1H1)

Здесь производится переход на новую страницу.

б) PRINT 2, A, B
2 FORMAT (1H4/2HA=F10.3/2HB=F10.3)

Здесь происходит переход на новую половину страницы с последующей печатью значений A и B, каждого с новой строки.

в) DIMENSION NTEXT (20)
PRINT 3, NTEXT
3 FORMAT (1HS, 20A6)

Здесь производится блокировка перевода страницы с печатью в очередной строке элементов текстового массива.

В случае, если при печати произойдет непредвиденное попадание управляющего символа в первую позицию, то будет выполнено действие, предписанное этим символом. Пусть например, $A = 123.7$ и написаны операторы

PRINT 10, A
10 FORMAT (F5.1)

Тогда цифра 1 попадет в первую позицию, что вызовет переход на новую страницу, на которой со второй позиции будет напечатано 23.7. Отсюда следует, что надо соблюдать осторожность в использовании первой позиции при печати.

8. Взаимодействие управления форматом со списком оператора ввода-вывода. Начало выполнения оператора ввода-вывода запускает в работу управление форматом. Каждое действие управления форматом зависит от совместной информации, поставляемой элементом списка оператора ввода-вывода и спецификацией преобразования из оператора FORMAT. Если список оператора ввода-вывода непуст, то должна быть указана хотя бы одна спецификация преобразования.

В процессе взаимодействия управления форматом со списком оператора ввода-вывода происходит заполнение очередной единицы записи. В эту очередную единицу записи включаются поля, указанные как в спецификациях преобразования, так и в редакционных спецификациях. Например, при печати по операторам

DIMENSION A (10)
PRINT 10, A
10 FORMAT (1H1, 10 (3X, F10.3))

единица записи состоит из 131 символа, что превышает число символов в одной строке АЦПУ БЭСМ-6. В этом случае правильный вывод информации не гарантируется. Если бы длина массива А была равна не 10, а меньше, то заполнение единицы записи окончилось бы раньше и число символов в ней не превысило бы максимально допустимого их количества 128.

Заполнение очередной единицы записи заканчивается либо по окончании обработки последнего элемента списка оператора ввода-вывода, либо когда управление форматом встречает знак / или последнюю правую скобку оператора FORMAT. Всякий раз, когда встречается символ /, являющийся признаком начала новой единицы записи, происходит либо вывод символов, накопленных в текущей единице записи (при выводе), либо переход к чтению новой единицы записи (при вводе).

Например, операторы

PRINT 1, A, B, C
1 FORMAT (1H1/2HA=F10.3/2H0—F10.3/2X, F10.3)

означают, что сначала произойдет переход на новую страницу, затем переход на новую строку с печатью заголовка А= и значения переменной А. Затем произойдет переход на новую строку, в которой указан управляющий символ 0, что вызовет пропуск одной строки и печать в следующей строке (начиная с третьей позиции) значения переменной В. Затем произойдет переход на новую строку, на которой будет выполнен пропуск двух позиций с печатью значения переменной С, начиная с 3-й позиции.

Операторы

```
READ 40, A, B, C  
40 FORMAT (2F15.3/E10.2)
```

означают, что вводимые числа пробиты на двух перфокартах. При этом, в частности, колонки 31÷80 первой карты читаться не будут.

В случае, когда число элементов в списке оператора ввода-вывода окажется меньше числа спецификаций преобразования в операторе FORMAT (с учетом числа повторений), работа управления форматом закончится, как только встретится очередная спецификация преобразования, а список оператора ввода-вывода окажется уже исчерпанным.

В случае, когда управление форматом доходит до самой последней правой скобки оператора FORMAT, а в списке оператора ввода-вывода еще имеются необработанные элементы, происходит переход на новую единицу записи и повторение списка оператора FORMAT, начиная с группы спецификаций, ограниченных правой скобкой, предшествующей внешней правой скобке оператора FORMAT. Если такая скобка отсутствует, то происходит возврат к левой внешней форматной скобке.

П р и м е р ы. В операторах

```
DIMENSION A (10)  
PRINT 2, A  
2 FORMAT (1H1/2E10.3)
```

произойдет переход на новую страницу, где на первой строке, отпечатаются значения первых двух элементов массива A. Дойдя до правой внешней скобки, управление форматом вернется к левой форматной скобке и повторит еще 4 раза те же самые действия (т. е. переход на новую страницу и печать двух очередных элементов массива A).

Для того чтобы переход на новую страницу был выполнен однажды, оператор FORMAT можно переписать в виде

```
2 FORMAT (1H1/(2E10.3))
```

В этом случае возврат будет происходить не на спецификацию 1H1, а на спецификацию 2E 10.3.

Запись оператора FORMAT в виде 2 FORMAT (1H1/E10.3, (E10.3))

приведет к тому, что элементы массива A, начиная с A(3), будут печататься по одному в каждой строке.

Когда $n+1$ последовательных символов / появляются в конце спецификации формата, то при вводе пропускается $n+1$ единиц записи, а при выводе записывается n пустых единиц записи. Пустая единица записи состоит только из пробелов. Если, однако, управление форматом доходит до самой правой скобки списка спецификаций формата, а в списке оператора ввода-вывода есть еще необработанные элементы, то записывается $n+1$ пустых единиц записи. Когда $n+1$ последовательных символов / появляется в середине списка спецификаций формата, то пропускается n единиц записи при вводе и записывается n пустых единиц записи при выводе.

9. Переменный оператор FORMAT. На фортране имеется возможность в процессе счета формировать список спецификаций оператора FORMAT. В этом случае список спецификаций вместе с его левой и правой скобками, но без метки оператора и слова FORMAT заносится в виде текстовых величин в заранее зарезервированный массив целого типа. При использовании такого оператора FORMAT в операторе ввода-вывода вместо его метки указывается наименование элемента массива, содержащего начало списка спецификаций формата.

Пусть, например, мы хотим выводить информацию при участии оператора вида

2 FORMAT (1H1/5(2X, F10.3)),

причем в одних случаях надо, чтобы информация выводилась согласно полному списку спецификаций этого оператора, а в других случаях — только по спецификациям (2X, F10.3). Можно, конечно, написать еще один оператор FORMAT, содержащий только что указанные спецификации, и обращаться к каждому из этих двух операторов по мере надобности. Но можно занести полный список рассматриваемого оператора FORMAT в определенное место памяти и в каждом из двух случаев обращаться к началу соответствующего списка спецификаций.

Элементы списка оператора FORMAT должны быть представлены как текстовые константы в правой части арифметического оператора присваивания либо как строки текстовых символов в операторе DATA, либо введены с перфокарт по спецификации A.

В нашем примере мы можем описать

DIMENSION IFORM (3)

и занести туда элементы списка оператора FORMAT следующим образом:

IFORM (1)=6H(1H1/5
IFORM (2)=6H(2X, F1
IFORM (3)=6H0.3))

Здесь общий список спецификаций начинается с элемента IFORM(1), а список внутренних спецификаций — с элемента IFORM(2). Если теперь написать оператор

PRINT IFORM (1), A, B, C

то текст, записанный в массиве IFORM, начиная с элемента IFORM (1), будет интерпретироваться как спецификации формата. Признаком конца списка спецификаций будет правая скобка соответствующего уровня.

Если написать оператор

PRINT IFORM(2), A, B, C

то список спецификаций будет начинаться с левой скобки внутренних спецификаций и оканчиваться соответствующей правой скобкой.

Рассмотрим еще один пример. Пусть написаны операторы

DIMENSION LF (5)
DATA (LF= '(E12.3, F9.1) (2(1X, 3F10.5))')

Тогда оператор

PRINT LF(1), A, B

будет эквивалентен операторам

PRINT 2, A, B
2 FORMAT (E12.3, F9.1),

а оператор PRINT LF(3), A, B, C эквивалентен операторам

PRINT 20, A, B, C
20 FORMAT (2 (1X, 3F10.5))

Для организации формата, список которого менялся бы в процессе счета, иногда необходимо производить операции над числами, заданными в текстовом виде. Например, иногда необходимо изменять n в спецификации nX с некоторым шагом. Подобные операции требуют использования операторов ENCODE (см. § 30).

§ 30. Операторы ENCODE и DECODE

Эти операторы аналогичны форматным операторам WRITE и READ с той разницей, что в передаче информации с помощью операторов ENCODE и DECODE внешние устройства не участвуют. В этом случае происходит передача информации из одного места оперативной памяти в другое с преобразованием передаваемой информации согласно спецификациям оператора FORMAT.

Оператор ENCODE преобразует информацию, заданную в форме, определяемой оператором FORMAT, в последовательность текстовых символов. Оператор DECODE производит обратную операцию, т. е. перекодирует информацию, заданную в виде последовательности текстовых символов, в ту форму ее представления, которая определяется соответствующей спецификацией оператора FORMAT.

Оператор ENCODE имеет вид

ENCODE (c , n , V) L

Здесь c — длина единицы записи (т. е. количество символов в ней); c может быть задано в виде целой константы или простой целой переменной; n — метка оператора FORMAT, в соответствии с которым производится преобразование информации; V — наименование массива или переменной целого типа, куда передается информация, получаемая в текстовом виде; L — список, содержащий передаваемую информацию и имеющий структуру списка оператора ввода-вывода (см. § 28). Оператор DECODE, имеющий вид

DECODE (c , n , V) L

производит обратную операцию, включающую в себя перекодировку текстовой информации, заданной в ячей-

ках памяти, определяемых идентификатором V, и последующую запись перекодированной информации в ячейки памяти, определяемые списком L.

Пусть, например, переменная I содержит целое число 10. С помощью операторов

```
    ENCODE (2, 5, IE) I  
    5 FORMAT (12)
```

произойдет перекодировка этой величины в текстовую величину, имеющую вид текстовой константы 2H10, с записью результата в IE. Оператор

```
    DECODE (2, 5, IE) I
```

произведет обратное действие по перекодировке текстовой величины 2H10 в целую величину 10.

Отметим, что посредством операторов ENCODE и DECODE можно преобразовать несколько единиц записи. При этом каждая единица записи начинается с нового машинного слова. Если единица записи оканчивается в середине машинного слова (т. е. с не кратно 6), то в операторе ENCODE производится заполнение избыточных правых позиций последнего машинного слова пробелами, а в операторе DECODE — пропуск избыточных символов.

Пример 1. Пусть надо сформировать текст для выдачи на печать следующим образом. Сначала берутся три первых символа текстовой величины IT, затем значение переменной A, выдаваемое по спецификации F10.3, затем шестисимвольный заголовок — SPEED, затем пропуск пяти позиций (т. е. 5 символов «пробел»), затем значение переменной B по спецификации E15.7.

Решение. Длина формируемой последовательности символов будет равна $3+10+6+5+15 = 39$ символам. Следовательно, для размещения ее надо отвести 7 ячеек памяти. Таким образом, мы можем написать

```
    DIMENSION ITEXT (7)  
    ENCODE (39, 2, ITEXT) IT, A, B  
    2 FORMAT (A3, F10.3, 6H — SPEED, 5X, E15.7)
```

Полученный массив ITEXT теперь можно выдать на печать, например, по спецификации A6.

Если вместо указанных операторов написать
DIMENSION ITEXT (9)
ENCODE (15, 2, ITEXT) IT, A, B
2 FORMAT (A3, F10.3/6H — SPEED, 5X/E15.7)

то произойдет следующее. В первую единицу записи попадут значения переменных IT и A, которые займут в ней 13 левых позиций. Еще 5 позиций, дополняющих первую единицу записи до трех полных машинных слов, будут заполнены пробелами. Во вторую единицу записи попадет заголовок — SPEED, занимающий 6 позиций, затем 5 пробелов, определяемых спецификацией 5X, и еще 7 пробелов, дополняющих единицу записи до 18 позиций. Третья единица записи содержит значение переменной B в текстовом виде и три пробела в правых позициях.

Пример 2. Сформировать текст, состоящий из двух первых символов текстовой переменной KT и четырех последних символов текстовой переменной JT.

Решение. Расположим сначала указанные 4 символа в старших разрядах машинного слова NT, что можно сделать так:

DECODE (6, 1, JT) NT
1 FORMAT (2X, A4)

Теперь сформируем в машинном слове KJT требуемый текст с помощью операторов

ENCODE (6, 2, KJT) KT, NT
2 FORMAT (A2, A4)

В заключение отметим, что операторы ENCODE и DECODE позволяют выдавать на печать таблицы сложной структуры. При этом можно поступать двояко: либо формировать переменный оператор FORMAT, либо заготавливать строки печатаемых символов в текстовом виде с последующим выводом их по спецификации A.

§ 31. Диагностика ошибок

Любая программа, даже тщательно написанная, может содержать ошибки, которые необходимо выявить и исправить в процессе ее отладки на ЭВМ.

Отладка программы состоит в том, что ее работа проверяется на некоторых тестовых примерах и результаты счета сравниваются с контрольными результатами, полученными каким-либо другим способом.

Пакет задачи после ввода в БЭСМ-6 проходит несколько этапов обработки, на каждом из которых могут быть обнаружены ошибки. Можно выделить следующие 5 этапов обработки пакета задачи на БЭСМ-6:

1) декодировка информации, заданной, вообще говоря, в различных кодировках;

2) трансляция подпрограмм с фортрана на автокод (или с алгола на язык загрузки);

3) трансляция подпрограмм с автокода на язык загрузки;

4) загрузка транслированных подпрограмм в память машины;

5) счет по программе.

Рассмотрим наиболее характерные ошибки, обнаруживаемые на каждом из этих этапов (за исключением этапа трансляции алгольных подпрограмм; об этом см. [39]).

1. Диагностика при декодировке пакета задачи. Декодировка текстовой информации, пробитой на картах, состоит в том, что эта информация перекодировается во внутренний код системы «Дубна». Исключения составляют лишь карты с двоичной информацией, заключенные между парами управляющих карт * BINARY и * END — BINARY (см. § 8), которые не требуют перекодировки.

Информация на перфокартах может быть пробита в различных кодировках (см. § 8). В процессе декодировки проверяется правильность пробивки информации в смысле данной кодировки. При обнаружении ошибок содержимое ошибочной перфокарты печатается на листинге с указанием способа кодировки и результата декодировки. Вместо ошибочных символов печатается знак !. При этом указывается десятичный номер карты в пакете задачи. Ошибочная карта исключается из дальнейшего рассмотрения; это может повлечь за собой дополнительную диагностику ошибок на последующих этапах обработки пакета задачи.

Весьма неприятными ошибками, обнаруживаемыми на этапе декодировки, являются ошибки в стандартных

и COSY-массивах (см. § 8 и [27]). Появление таких ошибок приводит к тому, что обработка пакета заканчивается на этапе декодировки. В пакете задачи могут вообще отсутствовать стандартные и COSY-массивы, однако ошибки при перфорации, вызвавшие появление пробивок в строках 5-й и 7-й либо в 7-й и 9-й первой колонки, могут привести к тому, что данная карта будет восприниматься как карта стандартного или COSY-массива, что неизбежно вызовет ошибки при попытке декодировки этой карты в указанном смысле.

В случае диагностики по ошибкам в стандартных или COSY-массивах надо в первую очередь проверить карты с этими массивами. Если же стандартные или COSY-массивы в пакете задачи отсутствуют, то ничего не остается, как тщательно просмотреть все карты и проверить пробивки в первой колонке. Ошибочные карты надо искать среди тех, которые имеют пробивки в строках 5-й и 7-й либо в 7-й и 9-й первой колонки.

Диагностика ошибок, обнаруживаемых в процессе декодировки, печатается в самом начале листинга, до распечатки текста карты * NAME (см. § 8).

2. Диагностика при трансляции с фортрана на автокод. Если в пакете задачи имеются подпрограммы, написанные на фортране, то производится их трансляция с фортрана. Транслятор с фортрана производит анализ каждого оператора и, обнаруживая ошибки, выдает их диагностику. Диагностика печатается непосредственно после ошибочного оператора. Кроме того, по окончании трансляции невыполняемых операторов может быть выдана диагностика, относящаяся ко всем невыполняемым операторам. Эта диагностика печатается после текста первого выполняемого оператора.

Каждый диагностический текст содержит пятизначный номер ошибки. Старшая цифра номера указывает на последствия этой ошибки. Если эта цифра 0, то диагностика является предупредительной, т. е. ошибка не препятствует дальнейшей трансляции и возможному счету по транслированной программе. Если эта цифра 1 или 2, то ошибка не препятствует дальнейшей трансляции, но препятствует выходу на счет. Если старшая цифра номера ошибки равна 4,

то дальнейшая трансляция подпрограммы идет с неполным синтаксическим контролем. Ошибки, номера которых начинаются не с 0, закрывают задачу выход на счет. При наличии в данном операторе нескольких ошибок обычно обнаруживается лишь первая из них, встретившаяся в процессе трансляции.

Ошибочный оператор исключается из дальнейшего рассмотрения. Поэтому ошибки в одних операторах могут повлечь за собой диагностику ошибок по другим, вполне правильно написанным операторам. Например, если написан ошибочный оператор

DIMENTION X(10),

то массив X при трансляции не будет учитываться. Вследствие этого многие операторы, использующие переменную X с индексом, будут считаться ошибочными и сопровождаться диагностикой.

Особо следует сказать об операторах DO. Здесь транслятор проверяет не только правильность самих операторов, но и правильность вложения циклов и правильность использования переменных в циклах.

Пусть, например, написано

```

DO 10 I=1,5
    . . . . .
10 CONTINUE
    DO 15 I=1, 10, 2
    . . . . .
15 CONTINUE
    . . . . .
    END

```

Транслятор выдаст диагностику по ошибочно написанному оператору с меткой 10. Этот ошибочный оператор будет аннулирован, и тем самым первый цикл не будет закончен. Встретив второй оператор DO, транслятор выдаст диагностику о том, что переменная I уже использована во внешнем цикле. Дойдя до оператора END и не встретив оператора с меткой 10, транслятор выдаст диагностику об отсутствии оператора, заканчивающего цикл DO.

Рассмотрим теперь диагностику по неправильному использованию переменных с индексами. Такая диагностика выдается в случае, когда используется пере-

менная с индексами (т. е. элемент массива), а описание массива отсутствует. Однако указанная диагностика не выдается в случаях, когда переменная с индексами используется в правой части оператора присваивания или в операторах IF. В этом случае транслятор трактует элементы неописанных массивов как функции.

Пусть, например, написан оператор

$$B=A(I)$$

причем описание массива A отсутствует. Тогда идентификатор A будет воспринят как наименование подпрограммы-функции. По данному оператору диагностики при трансляции выдано не будет. Идентификатор A будет, однако, указан в списке вызываемых функций и подпрограмм, который выдается при отсутствии ошибок в подпрограмме. Поэтому необходимо внимательно просматривать этот список, чтобы иметь возможность самому обнаружить «посторонние» функции, вызванные отсутствием описаний массивов. При обработке пакета задачи такие «посторонние» функции могут быть обнаружены лишь на этапе загрузки подпрограмм в память машины, да и то не всегда.

В случае, когда переменная содержит больше индексов, чем указано в описании массива, выдается диагностика о том, что переменная имеет слишком много индексов. Например, запись

DIMENSION A (10)
A (I, J)=0.

вызовет диагностику ошибки, поскольку массив A описан как одномерный.

Во всех рассмотренных случаях обнаружения ошибок в записи фортранных подпрограмм выход задачи на счет будет закрыт.

Есть, однако, два особых случая предупредительной диагностики, не закрывающих выхода на счет. Первый случай имеет место тогда, когда длина идентификатора превышает 6 символов. В этом случае печатается диагностика: «СЛИШКОМ МНОГО СИМВОЛОВ В ИДЕНТИФИКАТОРЕ». У данного идентификатора будут рассматриваться только первые 6 символов, и с таким урезанным идентификатором будет продолжена трансляция и, быть может, проведен счет.

Второй случай предупредительной диагностики имеет место тогда, когда пропущен знак действия между константой и левой скобкой, либо между правой скобкой и константой, либо между правой скобкой и переменной, либо между правой и левой скобками (см. § 15).

Отметим в заключение, что ввиду автономности трансляции каждой подпрограммы транслятор в принципе не может обнаружить ошибок, связанных с взаимодействием подпрограмм между собой. Сюда входят, в частности, ошибки, обусловленные несоответствием длин одноименных COMMON-блоков и с нарушением соответствия между формальными и фактическими параметрами.

3. Диагностика при трансляции с автокода на язык загрузки. Каждая подпрограмма, написанная на фортране, проходит два этапа трансляции: сначала с фортрана на автокод и затем с автокода на язык загрузки. Часть ошибок, не обнаруженных на первом этапе трансляции, обнаруживается на втором ее этапе. Заметим, что выход на второй этап трансляции возможен лишь при отсутствии ошибок на первом этапе.

На втором этапе трансляции обнаруживаются все неописанные идентификаторы и все идентификаторы, описанные более одного раза (дважды описанные). Диагностика при трансляции с автокода выдается после распечатки фортранного текста (если не установлен режим * NO LIST). При этом печатается заголовок: АВТОКОД MADLEN и дата версии автокода.

Диагностика вида: НЕОПИСАННЫЙ ИДЕНТИФИКАТОР * 10 означает, что в подпрограмме отсутствует оператор с меткой 10, на который имеется передача управления.

Диагностика вида: НЕОПИСАННЫЙ ИДЕНТИФИКАТОР/10 означает, что в подпрограмме отсутствует оператор FORMAT с меткой 10, на который имеются ссылки в операторах ввода-вывода или в операторах ENCODE и DECODE.

Диагностика вида: ДВАЖДЫ ОПИСАННЫЙ ИДЕНТИФИКАТОР * 10 означает, что в подпрограмме имеется более чем один выполняемый оператор с меткой 10. При этом несущественно, имеются ли ссылки

на оператор с данной меткой, или нет. Количество диагностических текстов равно числу выполняемых операторов с данной меткой.

Диагностика вида: ДВАЖДЫ ОПИСАННЫЙ ИДЕНТИФИКАТОР /10 означает, что в подпрограмме имеется более чем один оператор FORMAT с меткой 10.

Диагностика вида: НЕОПИСАННЫЙ ИДЕНТИФИКАТОР A означает, что в подпрограмме есть переменная A, которая не является ни формальным, ни фактическим параметром, не описана ни в одном из декларативных операторов, не указана ни в списке операторов ввода-вывода, ни в левой части оператора присваивания. Иначе говоря, такая диагностика выдается, если для переменной A транслятором не поставлена в соответствие никакая ячейка памяти машины. Например, в подпрограмме

```
SUBROUTINE S  
  B=A  
  RETURN  
END
```

переменная A будет считаться неописанным идентификатором. При этом существенно, чтобы переменная A использовалась без индексов и не была описана в операторе описания типа. Та же запись, но с добавлением декларативного оператора

```
REAL A
```

приведет к тому, что идентификатор A перестанет считаться неописанным, хотя смысл подпрограммы не изменится. Аналогичная запись, но с оператором

```
B=A (1)
```

вместо B=A приведет к тому, что идентификатор A будет считаться наименованием функций.

Иногда выдается диагностика вида: ДВАЖДЫ ОПИСАННЫЙ ИДЕНТИФИКАТОР * W0005.

Такая диагностика означает, что в записи неявного цикла в операторе ввода-вывода имеются лишние скобки либо указаны элементы неописанных массивов. В данном случае восьмеричный номер 0005 означает, что неявный цикл, в котором допущена ошибка, является пятым по счету среди циклов DO и неявных

циклов (при этом двумерный неявный цикл считается за два цикла, а трехмерный — за три цикла).

Иногда под диагностическим текстом слева печатается относительный (восьмеричный) адрес ошибочного автокодного оператора, помогающий установить место ошибки.

В случае, когда длина подпрограммы (вместе с внутренними массивами, переменными и пр.) получается слишком большой (более 23 листов МОЗУ), при трансляции с автокода выдается диагностика: **ДЛИНА ПОДПРОГРАММЫ ПРЕВЫШАЕТ ВОЗМОЖНОСТИ МАШИНЫ.**

4. Диагностика при загрузке подпрограмм в память машины. Если в пакете задачи имеется управляющая карта *EXECUTE (или *CALL — EXECUTE) и при декодировке пакета и трансляции подпрограмм не было обнаружено ошибок, закрывающих выход на счет, то производится загрузка подпрограмм в память машины (см. § 7).

Загрузке может предшествовать поиск некоторых подпрограмм в библиотеках, заказанных задач (см. § 8) и запись их во временную библиотеку. Список подпрограмм, извлеченных из каждой библиотеки, печатается на листинге.

В процессе загрузки происходит заполнение таблицы (или листа) загрузки, в которой указываются восьмеричные адреса начала каждой подпрограммы, каждого отдельного входа и каждого COMMON-блока. При отсутствии карты *NO — LOAD — LIST лист загрузки печатается по окончании загрузки. Он имеет вид таблицы в пять столбцов, в каждом из которых указывается идентификатор загружаемого объекта, тип загружаемого объекта (подпрограмме соответствует пробел, входу — буква E, COMMON-блоку — буква C) и абсолютный восьмеричный адрес начала объекта в оперативной памяти машины. Идентификаторы фортранных COMMON-блоков обрамляются звездочками, как при записи их на автокоде.

В случае отсутствия требуемой подпрограммы во временной библиотеке задачи печатается диагностика: **ОТСУТСТВУЕТ ПОДПРОГРАММА** <наименование подпрограммы>. Отсутствие подпрограммы чаще всего объясняется тем, что пользователь забыл за-

казать нужные библиотеки (например, отсутствуют карты * LIBRARY:), либо тем, что в пакете оказались обращения к «посторонним» подпрограммам-функциям, вызванные отсутствием описаний массивов.

Другой распространенной диагностикой является:

НЕВЕРНАЯ ДЛИНА <наименование
COMMON-блока>

Эта диагностика выдается в тех случаях, когда COMMON-блок уже загружен в память и очередная загружаемая подпрограмма потребовала увеличения длины этого COMMON-блока. При получении такой диагностики надо узнать длину COMMON-блока в той подпрограмме, наименование которой предшествует диагностике, после чего либо исправить неверную длину блока, либо, например, описать указанный блок в PROGRAM на максимальную длину.

В случае, если для размещения очередной подпрограммы недостаточно места в памяти, печатается диагностика: МАЛО ПАМЯТИ В <адрес>. Далее производится фиктивная загрузка остальных подпрограмм. В конце листа загрузки во всех случаях печатается:

СВОБОДНО <адрес первой свободной ячейки>

Если места в памяти недостаточно, то разность между адресами СВОБОДНО и МАЛО ПАМЯТИ В определяет размер памяти, требуемой задаче сверх заказанной. Если адрес первой свободной ячейки не превосходит 73377В, то можно заказать расширенную память, что делается управляющей картой * CALL FICMEMORY (см. § 8). В случае, если первый свободный адрес превышает 73377В, необходимо либо сократить размеры массивов, либо попытаться сэкономить память (см. § 58).

Если при загрузке обнаружены ошибки, то задача снимается со счета.

5. Диагностика при счете. При счете могут возникнуть ситуации, когда дальнейшая работа программы становится невозможной. Например, машина не может разделить на нуль или извлечь корень из отрицательного числа. Во всех подобных ситуациях происходит передача управления диспетчеру, который снимает задачу со счета с выдачей диагностики. Однако, если заранее принять необходимые меры, то можно про-

должить счет и после возникновения некоторых аварийных ситуаций (см. § 32).

Во всех случаях возникновения аварийных остановов (авостов) печатается типовая диагностика:

**** . . . * ОШИБКА В ПРОГРАММЕ**

МАТЕМАТИКА: <вид ошибки> ** . . . *

Строкой ниже печатаются пояснения, а еще строкой ниже — следующие данные:

1) содержимое сумматора в восьмеричном (машинном) виде;

2) содержимое сумматора в десятичном виде;

3) восьмеричный адрес команды, на которой произошел авост (примерно с точностью до 4 команд);

4) восьмеричный адрес последнего экстракода, использованного при счете;

5) содержимое индекс-регистров в порядке убывания их номеров от 15 до 1.

Для установления причины ошибки надо сначала найти оператор, в котором произошел останов. Для этого следует обратиться к листу загрузки и найти в нем адрес начала подпрограммы, ближайший к адресу останова со стороны меньших адресов. Пусть, например, адрес останова есть 30113В, а в листе загрузки указаны восьмеричные адреса подпрограмм

SIGMA	25421
MATINV	26365
ARCTG	32007

Тогда останов произошел в подпрограмме MATINV и относительный адрес искомого оператора в ней равен

$30113 - 26365 = 01526$ (все числа — восьмеричные)

Теперь надо взять листинг подпрограммы MATINV и обратиться к таблице относительных адресов ее операторов. Эта таблица, напечатанная после текста подпрограммы, указывает относительные адреса выполняемых операторов в ней. Адреса печатаются по 20 штук в строку. Вверху таблицы указывается нумерация по строке от 1 до 20, а слева от строки идет нумерация с шагом 20. Для нахождения номера нужного оператора надо сначала найти в таблице относительный адрес, ближайший к вычисленному ранее со сто-

роны меньших адресов. После этого надо к номеру, стоящему слева от строки, прибавить номер, стоящий над данным столбцом. Пусть, например, мы нашли в таблице относительные адреса 01522 01531. Тогда интересующий нас оператор начинается с адреса 01522. Пусть слева от строки, в которой находится данный адрес, стоит число 220, а над столбцом — 16. Тогда искомый оператор имеет порядковый номер $220 + 16 = 236$.

Номера выполняемых операторов печатаются слева от текста самих операторов, что позволяет по номеру легко находить сами операторы.

В случае, если останов произошел в подпрограмме, составленной самим пользователем, остается установить его причину. Однако может случиться, что останов произошел не в подпрограмме пользователя, а например, в библиотечной подпрограмме. В этом случае обычно удается установить адрес возврата во внешнюю подпрограмму, который равен содержимому 13-го индекс-регистра. Полезно знать также, что при входе в фортранную подпрограмму 13-й индекс-регистр пересылается в 8-й индекс-регистр, что позволяет более надежно определить адрес возврата. Наконец, 7-й индекс-регистр является адресом ячейки, где начинаются константы в фортранной подпрограмме.

Рассмотрим основные типы ошибок, возникающих при счете, определим их вероятные причины и приведем диагностику.

1. ПЕРЕПОЛНЕНИЕ АУ. Возникает в случаях, когда результат арифметической операции (чаще всего умножения) превосходит по модулю число, максимально допустимое для БЭСМ-6, равное округленно 10^{19} .

2. ДЕЛЕНИЕ НА НУЛЬ. Возникает при делении на ноль либо при делении на ненормализованное число. При этом ноль может получиться в процессе счета, когда результат по модулю меньше 10^{-19} (машинный ноль). Деление на ненормализованное число происходит, например, тогда, когда в подпрограмме указан вещественный формальный параметр, а в качестве фактического параметра задается целая величина.

3. DOUBLE PRECISION OVERFLOW. Возникает при переполнениях в арифметических операциях с двойной точностью (в том числе при делении на ноль).

В этом случае адрес останова находится внутри системной подпрограммы DUBLPREC, а 13-й индекс-регистр указывает адрес возврата в подпрограмму, вызвавшую останав.

4. КОРЕНЬ, ЛОГАРИФМ, АРКСИНУС, ЭКСПОНЕНТА, ТАНГЕНС. Остановы с такой диагностикой возникают соответственно при извлечении корня из отрицательного числа, взятии логарифма от неположительного числа, вычислении арксинуса от аргумента, большего 1 по модулю, вычислении экспоненты от аргумента, превосходящего $63 \ln 2 \approx 43,5$ и вычислении тангенса вблизи точки разрыва.

При этом следует учитывать, что из-за ограничения машинной точности может произойти отклонение от точного значения величины и вследствие этого выход ее за допустимый диапазон. Поэтому, например, при извлечении корня из малых величин рекомендуется предварительно взять модуль подкоренного выражения, а затем уже извлекать корень. В случае переполнения при вычислении экспоненты или тангенса можно перейти на двойную точность (см. § 32).

5. ЧУЖОЙ ЛИСТ. Возникает в результате порчи памяти, вызванной, например, неверной индексацией. Может быть также следствием обращения к подпрограмме или подпрограмме-функции не с тем числом параметров, какого она требует.

6. ЗАПР. КОМ. Эта диагностика возникает при попытке использовать диспетчерские или несуществующие команды. Обычно она является следствием порчи памяти или потери управления. Это может произойти, например, при неверном использовании переключателя или при отсутствии описания подпрограммы (являющейся фактическим параметром) в операторе EXTERNAL.

7. ОСТАНОВ ПО АЧ <адрес>. Останов по адресу числа обычно возникает при переполнении либо исчерпании магазина. Чаще всего это происходит при многократном обращении к подпрограмме с меньшим или с большим числом параметров, чем требуется. В первом случае происходит исчерпание магазина (адрес равен 53400 В, 55400 В или 73400 В), во втором — переполнение магазина (адрес равен 54000 В, 56000 В или 74000 В).

8. КОНТРОЛЬ КОМАНДЫ. Может возникнуть при порче памяти, в результате чего на месте команды оказалось число. При попытке использования этого числа как команды может возникнуть указанная диагностика.

9. КОНТРОЛЬ ЧИСЛА. Возникает лишь при сбоях в работе машины.

10. ЧУЖАЯ МЛ. Возникает при выходе на обмен с магнитной лентой или файлом диска, математический номер которых не был заказан в карте заказа ресурсов. В этом случае в 13÷18 разрядах сумматора указывается математический номер «чужой» ленты.

11. МАЛО МБ. Возникает в бездисковом варианте машины при нехватке открытых на пульте магнитных барабанов (так называемых «рабочих» барабанов).

В этом случае надо открыть больше рабочих МБ и повторно пропустить задачу.

12. ЗАЦИКЛИВАНИЕ. Чаще всего происходит при передаче управления оператором вычисляемый GO TO с нулевым значением переключателя. В этом случае задача выбрасывается или оператором, или по истечению заказанного лимита времени.

6. Диагностика по операторам FORMAT. В процессе трансляции на БЭСМ-6 не производится анализа содержимого операторов FORMAT. Этот анализ производится в процессе счета при каждом использовании данного оператора FORMAT. При этом могут быть обнаружены как ошибки в записи оператора FORMAT (например, отсутствие или неправильное расположение скобок), так и ошибки, вызванные несоответствием между вводимой или выводимой информацией и спецификациями формата. Довольно часто ошибки возникают из-за неправильной пробивки данных на перфокартах.

В большинстве случаев при обнаружении таких ошибок задача снимается со счета. Можно, однако, заблокировать снятие задачи, что делается управляющей картой *NO — OPT (или *NO — OPTIMIZATION). Диагностика по операторам FORMAT, выдаваемая при счете, содержит адрес того оператора ввода-вывода (либо ENCODE и DECODE), из которого произошло обращение к ошибочному оператору FORMAT. Кроме

того, в явном виде печатается та спецификация формата, на которой произошла ошибка.

7. Ошибка по выходу на карту * END _ FILE. Эта ошибка чаще всего возникает при вводе данных и означает, что вместо очередной карты, запрашиваемой оператором READ, прочитана управляющая карта * END _ FILE. В этом случае на листинге печатается * END _ FILE и задача снимается со счета.

Рассматриваемая ошибка обычно объясняется тем, что число карт данных оказалось меньше, чем требуется, либо FORMAT, соответствующий данному READ, оказался ошибочным.

§ 32. Советы и рекомендации

Здесь мы попытаемся дать некоторые практические советы и рекомендации, основанные в большинстве своем на личном опыте программирования одного из авторов.

1. О некоторых особенностях фортрана. Как сказано в § 31, некоторые ошибки, допускаемые при программировании, трудно обнаружить. Сюда относятся ошибки, связанные с потерей управления и обусловленные, в частности, неправильным значением переключателя в операторах вычисляемого GO TO и GO TO по предписанию. Поэтому нужна особая аккуратность при использовании этих операторов.

Особенностью фортрана является также то, что взаимодействие между подпрограммами не контролируется в процессе трансляции. Многие ошибки, связанные с использованием формальных параметров, могут быть обнаружены только в процессе счета. Появление «посторонних» функций при отсутствии описаний массивов, являющихся формальными параметрами, приводит к трудно уловимой ошибке, связанной с потерей управления. То же самое происходит при отсутствии описания подпрограмм и подпрограмм-функций, являющихся фактическими параметрами, в операторе EXTERNAL. Таким образом, желательна тщательная проверка соответствия между формальными и фактическими параметрами, а также описаний подпрограмм в операторе EXTERNAL.

Многие пользователи забывают, что в операторе DATA не производится преобразование типов за-
сылаемых констант. Например, запись вида DATA
A/1/ означает, что вещественной переменной A будет
присвоено значение целой константы 1. Такое несоот-
ветствие типов может привести к ошибке.

2. Об использовании целых величин. Начинающие
пользователи часто без нужды вводят в программу
целые величины. Это, кроме излишней траты машин-
ного времени, чревато еще и тем, что при умножении
и делении целых величин может получиться совсем
не тот результат, который ожидается.

На БЭСМ-6 диапазон допустимых значений для
целых величин ограничен сверху константой $2^{40}-1$.
Поэтому при умножении целых величин результат
берется по модулю 2^{40} . Если, например, $N \approx 10^7$,
то $N ** 2$ получится не порядка 10^{14} , а как минимум,
на два порядка меньше. При делении целых величин
происходит отбрасывание дробной части у резуль-
тата, что может привести к резкому его искажению
(например, оператор $A=1/2$ даст в результате 0.,
а не 0.5).

**3. О некоторых способах устранения переполнений
и машинных нулей.** Особенностью машины БЭСМ-6
является узкий диапазон порядков представимых чисел,
вследствие чего довольно часто возникают переполне-
ния и машинные нули. Покажем, как в некоторых
случаях можно их избежать.

Пусть, например, написан оператор

$$C=A/B ** 2$$

При $|B| \approx 10^{10}$ произойдет переполнение, а при $|B| \approx 10^{-10}$ — деление на нуль. Переписав этот оператор
в виде

$$C=A/B/V,$$

мы значительно уменьшим вероятность авоста.

Запись вида

$$C=\text{SQRT}(A ** 2+B ** 2)$$

даст переполнение уже тогда, когда хотя бы одна из
величин A и B будет порядка 10^{10} , и машинный
нуль — при A и B порядка 10^{-10} , хотя C имеет тот же

порядок, что A и B. В этом случае можно воспользоваться библиотечной функцией CABS(Z), написав, например, операторы

```
COMPLEX Z
DIMENSION RZ (2)
EQUIVALENCE (A, RZ), (B, RZ (2)), (Z, RZ)
C=CABS (Z)
```

4. Об использовании величин с двойной точностью.
Более общим способом избавления от переполнений и машинных нулей является переход к двойной точности. В этом случае путем описания нужных переменных, массивов и функций оператором

DOUBLE PRECISION

мы можем, кроме увеличения точности, значительно расширить диапазон используемых чисел. Этот способ, правда, очень дорог, так как операции над величинами с двойной точностью выполняются примерно на порядок медленнее, чем над вещественными. Тем не менее он достаточно удобен и прост. Однако здесь есть одна трудность.

Если в подпрограмме используются библиотечные подпрограммы-функции, например, EXP, SIN или COS, то для перехода к двойной точности необходимо изменить наименования этих функций соответственно на DEXP, DSIN и DCOS. Переписывать из-за этого соответствующие операторы подпрограммы иногда чересчур накладно.

В этом случае удобно ввести функции-операторы, написав, например,

```
DOUBLE PRECISION X, EXP, SIN, COS
EXP (X)=DEXP (X)
SIN (X)=DSIN (X)
COS (X)=DCOS (X)
```

Представление указанных функций в виде подпрограмм-функций менее удобно, так как это будет затрагивать все подпрограммы задачи, а не только данную подпрограмму.

Замечание. Переход к двойной точности невозможен в случае использования в соответствующих операторах комплексных величин. В этом случае можно

использовать библиотечную программу комплексной арифметики с двойной точностью (см. [3], программа A200) либо программы комплексной арифметики с большим диапазоном чисел (см. [28]).

5. О блокировке авоста. В ряде случаев при появлении авоста нежелательно, чтобы задача снималась со счета. Например, если задача состоит из ряда последовательно считаемых вариантов, то было бы желательно, — невзирая на авосты в некоторых из них, — сосчитать все остальные.

В подобных случаях можно использовать системную подпрограмму-функцию IFOVFL, которая имеет один параметр, являющийся фиктивным и не используемый при вычислении функции. Значение этой функции равно 0, если авоста не было, и не равно 0, если авост был. Обращение к IFOVFL должно производиться до выполнения операторов; подозрительных на авост. При возникновении авоста управление передается в место вызова функции IFOVFL со значением, не равным 0. Обращение к подпрограмме-функции IFOVFL имеет смысл делать в задаче лишь в одном месте.

Приведем пример использования функции IFOVFL.

```
IF (IFOVFL(0).NE.0) GO TO 20
20 CONTINUE
```

Здесь между оператором IF и оператором с меткой 20 расположены операторы, в которых может произойти авост. В случае такого авоста произойдет передача управления на логический оператор IF и оттуда на оператор с меткой 20, обходя «опасный» участок. При авосте, как всегда, будет выдана диагностика.

6. Об управлении загрузкой подпрограмм. Иногда возникает необходимость в том, чтобы некоторая подпрограмма загрузилась ранее или, наоборот, позже других. Например, в некоторых случаях, изменяя порядок загрузки подпрограмм, можно установить причину ошибки, вызвавшей порчу памяти.

Порядок загрузки подпрограмм может быть точно указан путем описания соответствующих подпрограмм и подпрограмм-функций в операторе EXTERNAL,

написанного в PROGRAM. Список в этом операторе рассматривается загрузчиком от конца к началу, т. е. подпрограммы, указанные в конце списка, будут загружаться первыми.

7. Определение времени счета. Часто бывает необходимо определить время счета программы или ее участка. Для этого служит обращение CALL STIME (СТ, RT), где в СТ выдается значение коммерческого, а в RT — значение счетного времени (в секундах), прошедших с момента начала обработки пакета задачи. Под коммерческим временем понимается общее время, затраченное машиной на данную задачу. Сюда включается и время работы внешних устройств. Счетное время — это время, в течение которого центральный процессор машины был занят данной задачей. Беря разность между двумя последовательными отметками времени, полученными с помощью подпрограммы STIME, можно узнать время, затраченное при работе данного участка программы. Минимальный интервал времени, измеряемый по STIME, равен 0,02 сек.

8. Об одном способе «редактирования» пакета. В заключение расскажем об одном интересном программистском приеме, о котором сообщил нам Л. Г. Каминский.

При отладке программ часто возникает необходимость временно вставлять карты в пакет или изымать карты из пакета задачи. При этом изъятые карты могут вновь понадобиться. Поэтому желательно, чтобы такие временно изымаемые карты не выбрасывались из пакета. Для этого надо в 80-й колонке такой карты пробить букву С. Если потребуется временно отменить выполнение операторов, пробитых на этой карте, то достаточно повернуть данную карту на 180° вокруг поперечной оси и поставить на то же место. В результате выполнения описанной процедуры буква С окажется в первой колонке, т. е. карта будет восприниматься как комментарий. При этом карта будет иметь выступающий над срезом угол и тем самым четко выделяться на фоне остальных нормально расположенных карт. Рассмотренный прием проходит в том случае, когда карты пробиты на устройствах с поколонной кодировкой.

§ 33. Примеры и упражнения

Рассмотрим некоторые характерные примеры оформления вычислительных процедур на фортране.

Задача 1. В двумерном массиве $A(N, N)$ вычислить сумму элементов, расположенных на одной горизонтали, вертикали или диагонали с элементом $A(I, J)$. Элемент $A(I, J)$ в сумму не включать.

Если рассматривать двумерный массив как обобщенную шахматную доску размером $N \times N$, то условие задачи можно сформулировать так: найти сумму элементов массива A , соответствующих тем полям доски,

	($i, 2$)	+	($i, 4$)		
$I = 2$	+	+	($2, 3$)	+	+
	($3, 2$)	+	($3, 4$)		
	($4, 1$)	+		($4, 5$)	
		+			($5, 6$)
		+			

$J = 3$

Рис. 3.

на которые может пойти ферзь с поля, расположенного на пересечении I-й горизонтали и J-й вертикали. Для массива A (6,6) и I=2, J=3 в сумму войдут все элементы 2-й строки и все элементы 3-го столбца (за исключением A (2,3)) и элементы A (1,2), A (3,4), A (4,5), A (5,6), A (4,1), A (3,2), A (1,4).

Пусть $M1 = \min(I-1, J-1)$, $M2 = \min(N-I, N-J)$, $M3 = \min(N-I, J-1)$, $M4 = \min(I-1, N-J)$. Тогда количество суммируемых элементов, расположенных на диагонали, идущей от элемента $A(I-M1, J-M1)$, равно $M1+M2+1$ (считая и элемент $A(I, J)$). Вторая диагональ, идущая от элемента $A(I+M3, J-M3)$, содержит $M3+M4+1$ элементов с учетом $A(I, J)$ (см. рис. 3).

Для вычисления минимальной из двух величин (т. е. для нахождения M_1, M_2, M_3 и M_4) воспользуемся стандартной подпрограммой-функцией $MIN0$, которая вычисляет минимальное значение для n величин целого типа ($n \geq 2$), указанных в качестве фактических параметров. С учетом сказанного можно предложить такое решение задачи:

<pre> FUNCTION SQUEEN + (A, N, I, J) DIMENSION A (N, N) S=0. DO 1 KN=1, N 1 S=S+A (I, KN)+ + A (KN, J) M1=MIN0 (I-1, J-1) M2=MIN0 (N-I, N-J) M3=MIN0 (N-I, J-1) M4=MIN0 (I-1, N-J) IN=I-M1 JN=J-M1 N1=M1+M2+1 DO 2 KN=1, N1 </pre>	<pre> S=S+A (IN, JN) IN=IN+1 2 JN=JN+1 IN=I+M3 JN=J-M3 N1=M3+M4+1 DO 3 KN=1, N1 S=S+A (IN, JN) IN=IN-1 3 JN=JN+1 SQUEEN=S-4. * + A (I, J) RETURN END </pre>
---	---

Решение можно значительно упростить и сделать более компактным, если обратить внимание на одно обстоятельство. Элементы двумерного массива, расположенные по горизонтали, вертикали или диагонали, можно представить как переменные с одним индексом. Этот индекс при переходе от каждого элемента к соседнему по горизонтали, вертикали или диагонали будет изменяться на некоторое постоянное число. При переходе от элемента $A(L, M)$ к $A(L, M+1)$ этот индекс увеличится на N , а при переходе от $A(L, M)$ к элементам $A(L+1, M)$, $A(L+1, M+1)$ и $A(L-1, M+1)$ индекс увеличится на 1 , $N+1$ и $N-1$ соответственно.

Таким образом, если составить подпрограмму, которая суммировала бы значения переменных с индексом, изменяющимся с постоянным шагом, то решение нашей задачи значительно упростится.

Подпрограмма, которую мы собираемся составить, должна вычислять сумму M элементов вещественного массива X с номерами $1, K+1, 2 * K+1, \dots, (M-1) * K+1$. С помощью этой подпрограммы мы сможем про-

суммировать элементы, расположенные по любому из четырех направлений — горизонтали, вертикали и двум диагоналям. Эту подпрограмму удобно оформить как FUNCTION.

Выясним, какую размерность надо указать для массива X, который будет одним из формальных параметров этой подпрограммы-функции. Последним элементом массива X, входящим в сумму, будет элемент $X((M-1) * K + 1)$. Однако мы не можем описать массив X такой размерности, так как величина $(M-1) * K + 1$ является арифметическим выражением. Но так как массив X будет формальным параметром, то место в памяти под этот массив будет отведено вне подпрограммы-функции. Поэтому массиву X можно приписать произвольную размерность. В подобных случаях приписывают одномерному массиву размерность 1.

Напишем теперь требуемую подпрограмму-функцию

FUNCTION SDIREC (X, M, K)		S=S+X (J)
DIMENSION X (1)		1 J=J+K
S=0.		SDIREC=S
J=1		RETURN
DO 1 I=1, M		END

С помощью подпрограммы-функции SDIREC мы получим теперь новое, значительно более простое решение исходной задачи.

```

FUNCTION SQUSD (A, N, I, J)
  DIMENSION A (N, N)
  M1=MIN0 (I-1, J-1)
  IN=I-M1
  JN=J-M1
  N1=M1+MIN0 (N-I, N-J)+1
  M2=MIN0 (N-I, J-1)
  IM=I+M2
  JM=J-M2
  N2=M2+MIN0 (I-1, N-J)+1
  SQUSD=SDIREC (A (I, 1), N, N)+SDIREC
  * (A (1, J), N, 1)
  /+SDIREC (A (IN, JN), N1, N+1)+SDIREC
  * (A (IM, JM), N2, N-1)-4. * A (I, J)
  RETURN
END

```

Здесь каждое из слагаемых в последнем операторе присваивания соответствует сумме элементов массива A , расположенных по горизонтали, вертикали и каждой из двух диагоналей.

Решение можно было бы записать еще короче, с помощью всего лишь одного (достаточно громоздкого) оператора присваивания, в котором вместо переменных IN , JN , $N1$, IM , JM , $N2$ были бы указаны соответствующие арифметические выражения.

Обратим внимание на то обстоятельство, что подпрограмма-функция $SDIREC$ оказалась пригодной для суммирования, казалось бы, очень отличных друг от друга последовательностей элементов массива A . Такая универсальность стала возможной благодаря введению формальных параметров.

З а д а ч а 2. Составить подпрограмму нахождения корня уравнения $f(x) = 0$ на заданном отрезке $[a, b]$ методом половинного деления. В качестве исходных данных задать значения a , b и ϵ , с точностью до которого нужно получить значение корня. Точность считается достигнутой, если длина отрезка, на котором находится корень, не превосходит ϵ .

Отметим, что если на концах отрезка $[a, b]$ функция $f(x)$ имеет одинаковые знаки, то без дополнительного исследования мы ничего не можем сказать о корнях уравнения $f(x) = 0$ на данном отрезке. Если же $f(x)$ имеет в точках a и b значения с разными знаками и непрерывна на всем отрезке $[a, b]$, то внутри отрезка имеется хотя бы один корень уравнения $f(x) = 0$. Будем предполагать, что эти условия выполнены.

В процессе поиска корня нам придется вычислять значения функции $f(x)$ на отрезке $[a, b]$. Если это вычисление производить внутри подпрограммы поиска корня, то для каждой новой функции $f(x)$ придется изменять подпрограмму. Поэтому удобнее выделить вычисление значений функции $f(x)$ в самостоятельную подпрограмму-функцию. Эта подпрограмма-функция будет составляться всякий раз, когда потребуются найти корень конкретного уравнения. Наименование этой подпрограммы-функции и список ее параметров задается подпрограммой поиска корня.

Представим себе такую ситуацию. Мы хотим найти значение корня уравнения $e^x - 5x^2 = 0$ на отрезке $[1, 5]$

и корень уравнения $\operatorname{arctg} x - 1/x = 0$ на отрезке $[1, 2]$. Чтобы воспользоваться подпрограммой поиска корня, нам придется сначала составить подпрограмму-функцию, вычисляющую значение функции $f(x) = e^x - 5x^2$, со строго определенным наименованием, указанным в подпрограмме поиска корня. После вычисления корня первого из уравнений надо составить подпрограмму-функцию, вычисляющую значение функции $\varphi(x) = \operatorname{arctg} x - 1/x$ с тем же самым наименованием. Таким образом, фиксированное наименование подпрограммы-функции не всегда удобно. Поэтому желательно сделать так, чтобы наименование подпрограммы-функции, вычисляющей значения функции, необходимые для поиска корня, можно было изменять. Для этого надо наименование подпрограммы-функции объявить формальным параметром подпрограммы поиска корня.

Оформим подпрограмму поиска корня как

SUBROUTINE EQROOT (A1,B1,EPS1,FUN,ROOT)

<pre> A=A1 B=B1 EPS=ABS (EPS1) IF (FUN (A).LT.0.) *GO TO 1 A=B1 B=A1 1 X=0.5 * (A+B) </pre>	<pre> IF (FUN (X)) 2, 5, 3 2 A=X GO TO 4 3 B=X 4 IF (ABS (B-A) *.GT.EPS) GO TO 1 5 ROOT=X] RETURN END </pre>
---	--

Здесь поиск корня производится на текущем отрезке $[A, B]$, причем значение функции в точке A всегда отрицательно, а в точке B положительно (при условии, что на концах исходного отрезка $[A1, B1]$ функция имеет разные знаки). На каждом шаге поиска корня отрезок $[A, B]$ делится пополам и затем отбрасывается та его половина, на концах которой нет перемены знака функции. Как только длина отрезка $[A, B]$ уменьшается до заданной величины EPS , определяющей точность вычисления корня, происходит выход из подпрограммы с выдачей середины отрезка в качестве значения корня.

Составим теперь программу нахождения корней уравнения $e^x - 5x^2 = 0$ на отрезке $[1, 5]$ и уравнения $\arctg x - 1/x = 0$ на отрезке $[1, 2]$ с точностью до $\epsilon = 10^{-9}$. Нетрудно убедиться, что на каждом из этих отрезков имеется ровно по одному корню каждого из соответствующих уравнений.

Мы составим PROGRAM и две подпрограммы-функции. Наименования этих подпрограмм-функций будут указаны в качестве фактических параметров при обращении к подпрограмме EQROOT и описаны оператором EXTERNAL в PROGRAM.

```

PROGRAM ROOT2
EXTERNAL FEXP, FARCTG
CALL EQROOT (1., 5., 1.E-9, FEXP, R1)
CALL EQROOT (1., 2., 1.E-9, FARCTG, R2)
PRINT 1, R1, R2
4 FORMAT (2F20.9)
END

FUNCTION FEXP (X)
FEXP=EXP (X)-5. * X ** 2
RETURN
END

FUNCTION FARCTG (X)
FARCTG=ATAN (X)-1./X
RETURN
END

```

Читатель, быть может, обратил внимание на то, что во всех рассмотренных примерах передача информации от одной подпрограммы к другой производилась через параметры. Как видно из этих примеров, аппарат параметров является очень гибким и универсальным способом передачи информации от одной подпрограммы к другой.

Этот способ наиболее соответствует принципу автономности подпрограмм. Однако недостаток этого способа состоит в том, что каждому формальному параметру должен обязательно соответствовать фактический параметр.

У п р а ж н е н и я. 47. Дан трехмерный массив A (M, M, N). Вычислить сумму тех его элементов, у которых сумма номера строки и номера столбца равна числу K . Для каких значений K задача имеет решение?

48. Дан одномерный массив $X(N)$. Расположить его элементы в порядке возрастания, т. е. так, чтобы первым был минимальный, а последним — максимальный элемент массива.

49. Вычислить все положительные корни уравнения $\sin x - x/\pi = 0$ с точностью 10^{-6} .

50. Составить программу приближенного вычисления интеграла $\int_a^b f(x) dx$ по формуле Симпсона с заданным шагом h .

ОТВЕТЫ. УКАЗАНИЯ. РЕШЕНИЯ

Ниже приведены ответы, указания и решения к упражнениям.

Правильные решения во многих случаях могут быть записаны и в ином виде. В ряде случаев даны самые простые, но не самые экономные (по времени выполнения) решения.

1. а) -1.236 ; б) 0.00015 ; в) 1974 . или 1974 ; г) $(1., 2.)$; д) $(0., 1.)$; е) $1.35E-6$; ж) $2.78D20$; з) $3.14159\ 26535\ 89793D0$; и) $-1.16D-25$.

2. а) Вместо точки использована запятая. б) Использованы недопустимые символы $*$. в) Действительная и мнимая части комплексной константы должны быть вещественными константами. г) Константа превышает максимальное значение, допустимое для вещественных констант.

3. Обозначения а), б), г), ж), и) допустимы, остальные не допустимы (в обозначениях в), д) и е) использованы недопустимые символы, а обозначение з) начинается с цифры).

4. а) $A * B * C + 3. * \sin(X) + \exp(-X) - 1. / (A * B + C * D)$

б) $A + (A * D - B * C) / (A ** 2 + B ** 2 + C ** 2 + D ** 2)$

в) $\text{ABS}(A * X ** 2 + B * X + C) - A ** 2 / \text{ABS}(X + A * X ** 2)$

г) $3. * X * \sin(X) ** 2 - 5. * Y * \cos(Y) ** 3$

5. а) $a \sin^2 x + \frac{5}{2 + 3\sqrt{x^2 + y^2}}$,

б) $3e^x + \ln(2 \cos^4 x + 3x^{1/3})$,

в) $|x^3 + 2 \sin x - \sqrt{x-2}|$.

6. а) комплексный; б) вещественный; в) не допустима; г) не допустима; д) не допустима; е) с двойной точностью; ж) не допустима; з) не допустима; и) с двойной точностью; к) вещественный; л) целый.

7. Записи в), г), ж) и з) допустимы, а записи а), б), д) и е) не допустимы.

8. Записи б) и е) допустимы, а записи а), в), г) и д) не допустимы.

```
9. IF (X.GE.2.) Z=SQRT (X-2.)
   IF (X.LT.2. .AND.X.NE.0.) Z=1./X+SIN (X+3.)
   IF (X.EQ.0.) Z=0.
```

```
10. IF (X** 2-2.* X+3.) 2,1,1
    1 T=SQRT (X** 2-2.* X+3.)
      GO TO 3
    2 T=1./ (X** 2+1.)
    3 CONTINUE
```

<pre>11. S = 0. A = 1. B = 1. 1 S = S + A/B</pre>		<pre>A = -A B = B + 1. IF (B - 99999.) 1,1,2 2 CONTINUE</pre>
---	--	---

<pre>12. W=1. A=1. B=2. GO TO 2 1 A=-A</pre>		<pre>B=B+2. 2 W=W+A/B+A/(B+1.) IF (B-99998.) 1,3,3 3 W=W-A/100000.</pre>
--	--	--

<pre>13. IF (X-Y) 1,1,2 2 IF (X.GT.Y+5.) * F=X IF (X.LE.Y+5.) * F=Y+5. GO TO 3 1 IF (X+1.-Y** 2) * 5,5,6</pre>		<pre>5 IF (X+1.LT.3.) F=X+1. IF (X+1.GE.3.) F=3. GO TO 3 6 IF (Y** 2.LT.3.) F=Y** 2 IF (Y** 2.GE.3.) F=3. 3 CONTINUE</pre>
--	--	--

З а м е ч а н и е. С помощью стандартных математических функций AMAX1 и AMIN1 (см. [33] или [40]) эта задача решается значительно проще.

14. а) Значение переключателя N не удовлетворяет условию $1 \leq N \leq 4$.

б) Вместо оператора ASSIGN использован арифметический оператор присваивания.

в) Значение переключателя ISW не может быть определено оператором ASSIGN.

г) Произойдет переход на метку, не предусмотренную в операторе GO TO по предписанию.

15. а) В операторе DO использованы недопустимое выражение $N+1$ и вещественная константа 2.

б) В операторе DO поставлена лишняя запятая.

в) Цикл оканчивается недопустимым оператором GO TO

г) Неправильное вложение циклов,

д) Во внешнем и внутреннем цикле использована одна и та же переменная.

е) Вход в цикл производится не через оператор DO

ж) Внутри цикла переопределяется один из его параметров,

з) Цикл оканчивается недопустимым оператором DO

- | | |
|--|---|
| 16. S=1.
A=-1.
DO 1 I=2, 10000 |
1 S=S+A/I |
| 17. S=1.
DO 2 I=2, 10000
A=-1.
K=I/3. |
L=I-3* K
IF (L.EQ.2) A=1.
2 S=S+A/I |
| 18. S=1.5
K=215
A=-1.
J=0
DO 3 I=3, 10000
S=S+A/I |
J=J+1
IF (J. LT.K) GO TO 3
K=K+1
J=0
A=-A
3 CONTINUE |
| 19. S=1.5
A=1.
B=2.
5 C=A+B
A=B |
B=C
IF (C.GT.1.E9) GO TO 6
S=S+1./C
GO TO 5
6 CONTINUE |
| 20. A=1.
DO 6 K=2, 1000
6 A=A+1./A** 2 | |
| 21. E=2.
FAC=1.
F=1.
8 F=F+1. |
FAC=FAC/F
E=E+FAC
IF (FAC.GE.1.E-10) GO TO 8 |

22. Одно из возможных решений таково:

INTEGER R15, KSI (8), TRP, ANNA (18, 9, 6)
COMPLEX FI, ALLA (3, 6), NIK (12, 3, 7)

23. а) Массив А описан дважды.

б) Идентификатор ARG описан дважды.

в) Массив TOP описан дважды.

г) Массив СК и идентификатор P5 описаны дважды

- | | |
|---|--|
| 24. FUNCTION
* SUMSTR (A, N, I)
DIMENSION
* A (N, N)
S=0. |
DO 5 K=1, N
5 S=S+A (I, K)
SUMSTR=S
RETURN
END |
|---|--|

25.	FUNCTION * SUMDIA (A, N) DIMENSION * A (N, N) S=0.	DO 2 K=1, N S=S+A (K, K) SUMDIA=S RETURN END
26.	FUNCTION * NUMSTR (A, M, N) DIMENSION * A (M, N) SMAX=0. DO 5 K=1, N 5 SMAX=SMAX + * A (1, K) L=1 DO 3 I=2, M	S=0. DO 2 J=1, N 2 S=S+A (I, J) IF (S.LE.SMAX) GO TO 3 SMAX=S L=L+1 3 CONTINUE NUMSTR=L RETURN END
27.	FUNCTION * SUMMAX * (A, M, N) DIMENSION * A (M, N) AMAX=A (1, 1) IMAX=1 JMAX=1 DO 5 I=1, M DO 5 J=1, N IF (A(I, J) * .LE.AMAX) * GO TO 3	AMAX=A (I, J) IMAX=I JMAX=J 3 CONTINUE 5 CONTINUE S=-A (IMAX, JMAX) DO 6 J=1, N 6 S=S+A (IMAX, J) DO 7 I=1, M 7 S=S+A (I, JMAX) SUMMAX=S RETURN END
28.	FUNCTION * SCALR (A, M, N, * I, J) DIMENSION * A (M, N)	S=0. DO 5 K=1, N 5 S=S+A (I, K) * A (J, K) SCALR=S RETURN END

29. У к а з а н и е. Представить элементы массива A как переменные с одним индексом.

30. а) $F(X, Y) = 3 * \sin(X + Y) + 2.5 * \exp(-\sqrt{X^2 + Y^2})$

б) Нельзя, так как для вычисления значения $f(x)$ потребуется написать более одного оператора.

в) $F1(X, A1) = X * \sin(X) + A1$

З а м е ч а н и е. Для вычисления значения выражения, указанного в задании, надо обратиться к F1 (X, A1), например, следующим образом:

$$F = F1(X, A(I, J))$$

31. Нельзя, так как для решения задачи потребуется написать более одного оператора.

32. Нельзя, по аналогичной причине (см. предыдущий ответ).

33. SUBROUTINE * TRANSP (A, M, N) DIMENSION * A (M, N) AMIN=A (1,1) AMAX=A (1,1) IMIN=1 IMAX=1 DO 3 I=1, M DO 5 J=1, N B=A (I, J) IF (B-AMAX) * 7, 7, 6	6 AMAX=B IMAX=I 7 IF (B-AMIN) 8,5,5 8 AMIN=B IMIN=I 5 CONTINUE 3 CONTINUE DO 20 K=1, N B=A (IMAX, K) A (IMAX, K)=A (IMIN, K) 20 A (IMIN, K)=B RETURN END
34. SUBROUTINE * TRASTR (X) DIMENSION * X (10, 35) DO 5 I=1, 5 DO 3 J=1, 35	B=X (I, J) X (I, J)=X (11-I, J) 3 X (11-I, J)=B 5 CONTINUE RETURN END
35. SUBROUTINE * TRDIAG (A, N) DIMENSION * A (N, N) N2=N/2 DO 5 K=1, N2 B=A (K, K)	A (K, K)=A(N+1-K, * N+1-K) 5 A (N+1-K, N+1-K)=B RETURN END
36. SUBROUTINE * MATMUL (A, B, * C, N) DIMENSION * A (N, N), B (N, * N), C (N, N) DO 6 I=1, N DO 6 J=1, N C (I, J)=0	DO 6 K=1, N 6 C (I, J)=C (I, J)+A (I, K)+ * B (K, J) RETURN END
37. SUBROUTINE * SDIAG (A, N, S) DIMENSION * A (N), S (1) NN=N+N NM=NN-1 DO 5 I=1, NM S (I)=0. KR=N-I+1 KC=1 NJ=I	IF (I.LE.N) * GO TO 2 KR=1 KC=I-N NJ=NN-I 2 DO 5 J=1, NJ S (I)=S (I)+A (KR, KC) KR=KR+1 3 KC=KC+1 5 CONTINUE RETURN END

Здесь массив S, являющийся формальным параметром, описан как массив размерности 1, хотя число его элементов равно $2 * N - 1$ (см. § 26, пример 6).

38. а) COMMON /CLD/ C (15), L, D (10, 6), A (10, 7, 3)
COMPLEX C
LOGICAL L
DOUBLE PRECISION D

Длина блока равна $2 \cdot 15 + 1 + 2 \cdot 10 \cdot 6 + 10 \cdot 7 \cdot 3 = 361$ машинным словам.

б) COMMON /RB/ R (15, 10), F (7, 6), LT (16)
INTEGER R
DOUBLE PRECISION F
LOGICAL LT

Длина блока равна $15 \cdot 10 + 2 \cdot 7 \cdot 6 + 16 = 250$ машинным словам.

в) COMMON A (7, 5), T, TN, N, PSI, D
LOGICAL T (10)
INTEGER TN (10, 6, 2), PSI, D

Длина немеченного блока равна 168 машинным словам.

39. а) DIMENSION A (15, 6), C (10, 8)
COMPLEX C
EQUIVALENCE (A (47), C (48))

Суммарная длина массивов с учетом эквивалентности равна 160 машинным словам.

б) COMMON /DBLOCK/D
DOUBLE PRECISION D (16, 2, 5)
INTEGER N (6, 70)
EQUIVALENCE (D (85), N (137))

Эквивалентность допустима, так как элемент D (85) расположен, начиная со 169-го слова, в общем блоке, а элемент N (137) соответствует 137-му слову общего блока. Суммарная длина массивов (и, следовательно, длина блока) равна 452 машинным словам.

в) COMMON /DP/ CT (10, 6)
COMPLEX CT
✓ DOUBLE PRECISION D
DIMENSION N (5, 5)
EQUIVALENCE (D, CT (13), N (17))

Эквивалентность допустима. Длина блока с учетом эквивалентности равна 120 словам.

40. а) Эквивалентность недопустима, так как она требует изменения начала общего блока.

б) В операторе эквивалентности указаны переменные более чем с одним индексом.

в) Эквивалентность противоречива, так как, с одной стороны, элемент A эквивалентен C (6), а с другой стороны, C.

г) Эквивалентность связывает элементы общих блоков.

д) Эквивалентность требует изменения начала общего блока.

41. a) DIMENSION A (8, 3)
DATA (A (10))=5 (1.5), 2 (77B))
6) DOUBLE PRECISION D (18, 2)
DATA D/3 * 2.5D8, 5 * 1.65D4/
b) DATA T, NX /15, 5 * 1.5, 2 * 3.8, 3 * (-5.6)/
DIMENSION NX (10)
REAL NX
42. FUNCTION
* AMIMA (A, N)
DIMENSION A (N)
AMIN=A
AMAX=A
DO 1 I=1, N
IF (A(I).LT.
* AMIN) AMIN=A (I)
IF (A(I).GT.AMAX)
* AMAX=A (I)
1 CONTINUE
AMIMA=AMIN+AMAX
RETURN
END
- 43.a) SUBROUTINE
* SXMIMA (X, SUM)
DIMENSION X (30)
SUM=AMIMA
* (X, 30)
RETURN
END
44. SUBROUTINE
* INTRAP (A, B, N,
* TINT, FUN)
H=(B-A)/N
TINT=0.5* (FUN
* (A)+FUN (B))
M=N-1
X=A
DO 1 I=1, M
X=X+H
1 TINT=TINT+FUN (X)
TINT=H * TINT
RETURN
END
45. PROGRAM
* TRAP
EXTERNAL EXSQ
CALL INTRAP
* (0., 1., 50, R,
* EXSQ)
PRINT 10, R
10 FORMAT (1X, 2HR=,
* E15.9)
END
FUNCTION EXSQ (X)
EXSQ=EXP (-X ** 2)
RETURN
END

46. Составим подпрограмму-функцию, вычисляющую C_n^* так, чтобы по возможности избежать переполнений. У результата возьмем значение его ближайшего целого.

FUNCTION BINCO (N, K)
A=N
B=1.
C=N
DO 1 I=2, K
A=A-1.
B=B+1.
1 C=C * (A/B)
NC=C+0.5
BINCO=NC
RETURN
END

47. У к а з а н и е. Решение возможно лишь при $K \leq 2M$. Множество элементов, входящих в сумму, расположено на каждом «этаже» параллельно диагонали, соединяющей элементы $A(M, 1, J)$ и $A(1, M, J)$, где $J \leq N$ — номер этажа. Использовать подпрограмму-функцию SDIREC из § 33.

48. У к а з а н и е. См. [29], стр. 50, ответ к упр. 38.

49. У к а з а н и е. Использовать подпрограмму-функцию EQROOT из § 33.

50. У к а з а н и е. Формула Симпсона с шагом h имеет вид

$$\int_a^b f(x) dx \approx \frac{h}{3} \left(f(a) + f(b) + 4 \sum_{k=1}^n f(a + (2k-1)h) + \right. \\ \left. + 2 \sum_{k=1}^{n-1} f(a + 2kh) \right),$$

где $h = \frac{b-a}{2n}$.

Однако в условии задачи не сказано, что шаг h укладывается в отрезке $[a, b]$ четное число раз. Поэтому надо разбить отрезок $[a, b]$ на части $[a, t]$ и $[t, b]$, где $t = a + 2mh$ при m целом неотрицательном и $b-t < 2h$. Затем по каждому из этих отрезков проинтегрировать отдельно (с шагом h и $(b-t)/2$ соответственно).

Заметим, что интегрирование методом Симпсона обычно реализуется с автоматическим выбором шага и с контролем точности (см., например, [2], программа D120).

Г Л А В А III

ОПИСАНИЕ АВТОКОДА madlen

Введение

Автокод madlen (см. [5], [6]) используется в мониторинговой системе «Дубна» как самостоятельный язык программирования нижнего уровня и как промежуточный язык при трансляции подпрограмм с фортрана. При трансляции алгольных процедур (написанных на алголе-ГДР [39]), выхода на madlen не происходит, однако пользователь имеет возможность получить распечатку текста транслированной процедуры в виде, сходном с автокодной записью. При необходимости можно получить автокодный текст транслированной алгольной процедуры путем обратной трансляции ее с языка загрузки на автокод (см. [26]).

Часть пакета задачи, написанная на автокоде, оформляется как отдельная подпрограмма или несколько подпрограмм и транслируется независимо от других подпрограмм. Если автокодная подпрограмма оформлена по определенным правилам, то она может быть использована наравне с фортранскими и алгольными подпрограммами (см. § 46).

Текст подпрограммы на автокоде обладает достаточной наглядностью, что позволяет, в частности, с успехом анализировать подпрограммы, написанные другими авторами.

Основу автокодной подпрограммы составляют автокодные команды, представляющие собой символическую запись машинных команд. Каждая автокодная команда, как правило, транслируется в одну машинную команду (исключения описаны в § 47). То обстоятельство, что

madlen является промежуточным языком при трансляции с фортрана, естественным образом отразилось на его структуре. Например, любой невыполняемый (декларативный) оператор фортрана имеет автокодный аналог.

Транслятор с автокода выдает удобную для пользователя диагностику ошибок, имеет аппарат редактирования, позволяющий получать автокодный текст в виде, удобном для чтения, независимо от способа его пробивки. Все это в сочетании с высокой скоростью трансляции (200 операторов в секунду) делает автокод madlen удобным и эффективным средством программирования при решении задач, требующих максимального использования специфики БЭСМ-6.

§ 34. Операторы автокода

Подпрограмма на автокоде состоит из операторов, каждый из которых располагается на отдельной перфокарте. При необходимости запись некоторых операторов может быть продолжена на последующие перфокарты. Отметим, что в некоторых руководствах по автокоду наравне с термином «оператор» используется термин «инструкция».

Формат автокодного оператора подобен формату машинной команды. Это означает, что отдельные его составные части соответствуют индекс-регистру, коду операции и адресной части машинной команды. Кроме того, всегда можно обозначить адрес ячейки, в которой находится данная команда, а также поставить признак того, что данная команда является левой, т. е. занимает 24 старших разряда машинного слова. Поскольку для записи каждой части автокодного оператора нет специально выделенных колонок перфокарты, эти части отделяются друг от друга специальными разделителями (запятая, двоеточие, точка).

Любой автокодный оператор может быть расположен в произвольных колонках перфокарты с 2-й по 42-ю включительно. Некоторые операторы имеют «длинную» структуру и могут располагаться по 72-ю колонку включительно. Колонки с 73-й по 80-ю всегда относятся к комментарию.

При дальнейшем изложении мы будем использовать запись понятий, не являющихся элементами данного языка, с помощью угловых скобок $\langle \rangle$.

Большинство операторов автокода madlen можно записать следующим образом:

$$\langle \begin{array}{c} \text{указатель} \\ \text{метки} \end{array} \rangle : \langle \begin{array}{c} \text{указатель} \\ \text{индекс-регистра} \end{array} \rangle, \langle \text{мнемокод} \rangle, \langle \begin{array}{c} \text{полный} \\ \text{адрес} \end{array} \rangle$$

Обязательными элементами оператора являются запятые, отделяющие мнемокод от указателя индекс-регистра и от полного адреса, записываемого в адресной части. Мнемокод также всегда (за исключением единственного случая) присутствует в записи оператора. Остальные части оператора могут отсутствовать.

Назначение данного оператора (команда, константа, описание и т. д.) определяется его мнемокодом. Обозначения мнемокодов выбраны так, чтобы по их записи можно было легко определить назначение соответствующих операторов.

П р и м е р ы.

```
* 5 : 14, XTA,
ABT : , A+X, INT
/7 : , BSS,
      : , XTA, A+4
      , A * X, =R5.
PRINT8 : , NAME,
```

§ 35. Идентификаторы

Идентификаторы служат для записи символических обозначений.

Идентификатор — это последовательность букв и цифр, начинающаяся с буквы.

Под «буквами» в автокоде понимаются все буквы латинского и русского алфавитов и, кроме того, символы * и /. Максимально допустимая длина идентификатора равна 8 символам (пробелы игнорируются).

Таким образом, множество допустимых идентификаторов в автокоде шире, чем в фортране или алголе.

Перечислим остальные символы, составляющие алфавит языка madlen, но не входящие в состав

идентификаторов:

+	плюс	.	точка
—	минус	=	равенство
(левая скобка	;	двоеточие
)	правая скобка	'	апостроф
,	запятая		

В некоторых конструкциях языка (текстовых константах и комментариях) допустимы следующие знаки:

;, < >, [], _ (подчеркивание), !, \$ или ◇.

Особую роль играет идентификатор, состоящий из одного символа *. Он обозначает адрес той команды, в которой он написан. Например, оператор `UJ, * + 2` означает безусловный переход на левую команду машинного слова, адрес которого равен адресу данной команды плюс 2.

Примеры идентификаторов:

```
PROGRAM
* 15
/ 101
* AB *
```

§ 36. Мнемокоды

Мнемокод является важнейшей составной частью автокодного оператора, определяющей назначение и форму записи остальных его частей.

Мнемокод — это определенная последовательность букв, понимаемых в смысле автокода, цифр и знаков + и —.

Особую роль играют мнемокоды команд, которые выбраны таким образом, чтобы по мнемокоду можно было легко определить, какой машинной операции соответствует данная команда, является она короткоадресной или длинноадресной, соответствует ее адрес некоторой ячейке памяти или имеет особый смысл и т. д.

Мнемокоды команд, как правило, состоят из трех символов (см. приложение 1). Исключение составляют специальные мнемокоды (CALL, BASE и др.), а также мнемокоды экстракодов, имеющие иногда по несколько равноправных обозначений (см. приложение 1А).

Второй символ мнемкокода команды обычно определяет вид соответствующей машинной операции. В частности, для обозначения операций сложения, вычитания, умножения и деления используются символы $+$, $-$, $*$ и $/$. Дадим толкование некоторых других указателей вида операций:

- T — пересылка (Transfer),
- Z — условный переход по $\omega = 0$ (Zerojump),
- J — безусловный переход (Jump),
- S — сдвиг (Shift),
- A — логическое умножение (And),
- O — логическое сложение (Or),
- E — сравнение (Excluding or),
- R — циклическое сложение (Round),
- L — конец цикла (Loop).

Первый и третий символы мнемкокода команды обычно обозначают регистр машины (сумматор, регистр младших разрядов и т. д.) или тип адреса (короткий, длинный, модифицируемый с помощью индекса-регистра и т. п.). Если вторым символом мнемкокода команды является T, то третий его символ указывает, куда передается информация, а первый определяет источник информации.

Опишем наиболее употребительные указатели регистров машины и типов адресов:

- A — сумматор (Accumulator);
- S — магазин (Stack);
- Y — регистр младших разрядов (Young position);
- M — указатель индекс-регистра (Modifier);
- E — «регистр» порядка, т. е. 48÷42 разряды сумматора, в которых размещается порядок числа;

X — короткий адрес ячейки памяти;

V — длинный адрес, не модифицируемый по индекс-регистру;

U — длинный адрес, модифицируемый по индекс-регистру;

N — адрес, не являющийся адресом ячейки памяти и рассматриваемый по mod 2^7 .

Примеры мнемкокодов:

ATX — запись из сумматора в ячейку памяти с коротким адресом;

ASN — сдвиг сумматора на число разрядов, определяемое семью младшими разрядами исполнительного адреса;

UZA — условный переход по $\omega = 0$ с модификацией адреса по индекс-регистру;

VTM — запись исполнительного адреса команды (без-модификации по индекс-регистру) в индекс-регистр;

E+N — сложение порядка числа на сумматоре с адресом, определяемым семью младшими разрядами исполнительного адреса команды, минус 64.

Отметим, что вместо мнемонических обозначений команд можно использовать их восьмеричные (машинные) обозначения. Однако использовать такие обозначения не рекомендуется, так как это лишает команды наглядности и, кроме того, не позволяет пользоваться многими возможностями автокода (например, базированием).

§ 37. Полный адрес

В адресной части команды разрешается записывать:

- 1) идентификатор,
- 2) целое десятичное число без знака,
- 3) \langle восьмеричное число \rangle В,
- 4) *
- 5) адрес типа «литерал» (см. § 41),
- 6) полный адрес.

Полным адресом называется любая конструкция адресной части, состоящая не более чем из двух адресов типов 1) ÷ 5), взятых, быть может, со знаками + или —. Адрес может быть пустым, ему соответствует нулевое значение.

П р и м е р ы.

А
* +2
X +5
C +30B
—AB +C15

Адреса, записанные в виде десятичного или восьмеричного числа (или их комбинации со знаками + и —), являются абсолютными, т. е. они не зависят от расположения подпрограммы в памяти машины. Фактические значения адресов остальных типов будут определяться при загрузке подпрограммы в память машины. При этом вся адресная арифметика выполня-

ется по модулю 2^{15} , т. е. у результирующего адреса берется 15 младших разрядов. Отрицательные значения берутся в дополнительном коде. Например, адресу —1 соответствует восьмеричный адрес 77777В.

§ 38. Указатель индекс-регистра

Указатель индекс-регистра можно записывать в виде десятичного числа от 1 до 15. При этом пустой или нулевой указатели соответствуют отсутствию модификации. Однако удобнее использовать идентификаторы, так как это позволяет легко переопределять конкретные значения соответствующих индекс-регистров. В последнем случае идентификаторам должны быть поставлены в соответствие допустимые номера индекс-регистров (десятичные или <восьмеричные> В) с помощью операторов эквивалентности (см. § 42).

Отметим, что пустой и нулевой указатели индекс-регистра в некоторых случаях воспринимаются транслятором по-разному (см. § 47).

§ 39. Метки

Метка обозначается идентификатором и выполняет несколько функций. Во-первых, она является описанием данного идентификатора, т. е. однозначно его определяет. Поэтому каждый идентификатор может быть указан в виде метки, как правило, не более одного раза. Во-вторых, метка, относящаяся к команде, означает, что данная команда будет помещена в левую половину машинного слова. В-третьих, адрес метки, указанной перед командой, константой или оператором с мнемокодом BSS, является внутренним адресом данной подпрограммы (см. § 47).

Метка отделяется справа двоеточием. Двоеточие без идентификатора метки (пустая метка) означает, что данную команду надо поместить в левую половину слова. При этом может случиться, что предыдущее машинное слово оказалось не полностью сформированным. В этом случае транслятор автоматически дополняет его правой командой с мнемокодом UTC (см. приложение 1).

Отметим, что в большинстве необходимых случаев транслятор автоматически располагает нужную

команду в левой половине машинного слова независимо от наличия двоеточия.

Команды LIB, * 60 и * 66 всегда считаются помеченными, т. е. имеющими метку, даже если она не указана. Помеченными считаются также все команды, непосредственно следующие за командами

VJM, FUN, PRINT, TAPE, DRUM, SJ, CTX,
* 50 ÷ * 57, * 61 ÷ * 65, * 67, * 70 ÷ * 77.

З а м е ч а н и е. Описанные выше случаи автоматической помеченности не распространяются на восьмеричные коды операций, соответствующие указанным мнемосодам,

§ 40. Константы

Автокод допускает 4 типа констант:

восьмеричные (OCT, LOG),

целые (INT),

вещественные (REAL),

текстовые (ISO, GOST, TEXT, TEL).

Константы могут быть заданы как специальными операторами (мнемосокоды которых приведены выше в скобках), так и адресами типа «литерал», т. е. путем непосредственной записи константы в адресной части команды. В первом случае вид константы определяется ее адресной частью. Указатель индекс-регистра при трансляции констант не учитывается и может быть использован, например, для их нумерации. Указатель метки имеет тот же смысл, что и для команд. Отметим, что для правильной трансляции констант не требуется их помеченности, т. е. пустая метка перед константой не обязательна. Константы всегда транслируются в целое число машинных слов, причем для первых трех типов — в одно слово.

Константа типа OCT имеет в адресной части восьмеричное число, состоящее не более чем из 16 цифр (без буквы В справа). При трансляции константы эти цифры будут располагаться в машинном слове слева направо, начиная со старших его разрядов. Лишние (сверх 16) правые цифры будут отброшены, а недостающие до полного машинного слова — дополнены нулями *справа*.

П р и м е р ы.

Восьмеричная константа, ОСТ, 4 транслируется в машинное слово 4000 0000 0000 0000, а константа ,ОСТ, 1777 7356 4216 3730 157 будет представлена в машине как 1777 7356 4216 3730.

Константы типа LOG аналогичны предыдущим с той лишь разницей, что заполнение машинного слова производится справа налево, начиная с младших его разрядов, и дополнение нулями производится *слева*. Отметим, что константы этого типа являются аналогами восьмеричных констант фортрана.

П р и м е р ы.

Восьмеричная константа ,LOG, 4 транслируется в машинное слово 0000 0000 0000 0004. Однако константа ,LOG, 1777 7356 4216 3730 157 будет представлена в виде 1777 7356 4216 3730 т. е. так же, как и в случае константы типа ОСТ.

Константы типа INT могут иметь в адресной части любое целое десятичное число (без знака или со знаком —), не превосходящее $2^{40}-1$. В результате трансляции такой константы будет образовано машинное слово, представляющее ненормализованное число с порядком 40 и с запятой, фиксированной в конце мантиссы (мантиссы отрицательных чисел представляются в дополнительном коде с 1 в 41-м разряде).

П р и м е р ы.

Целая константа ,INT, 38 транслируется в машинное слово 6400 0000 0000 0046. При этом десятичному числу 38 соответствует восьмеричное число 46, расположенное в младших разрядах мантиссы. В разрядах порядка (48—42) указано машинное представление порядка 40_{10} .

Константа ,INT, —5 будет представлена в виде 6437 7777 7777 7773. Здесь мантисса представлена в дополнительном коде с 1 в 41-м разряде.

Константы типа REAL имеют в адресной части любое десятичное число, допустимое для БЭСМ-6. При этом правила записи аналогичны фортранным, т. е. обязательна десятичная точка и допустимо использование буквы E с последующим целым числом в качестве показателя степени при 10.

П р и м е р ы. а) Вещественная константа ,REAL, 1. транслируется в нормализованное машинное число

4050 0000 0000 0000 Здесь в разрядах $48 \div 42$ содержится двоичный код 1 000 001, а в разрядах $41 \div 40$ — код 01. Объединяя каждые три соседних двоичных разряда в один восьмеричный, получим в разрядах $48 \div 40$ восьмеричное число 405.

б) Константа ,REAL, 2. E1 будет представлена как 4252 0000 0000 0000. Здесь в разрядах $48 \div 42$ содержится двоичный код 1 000 101, а в разрядах $41 \div 37$ — код 01010, что соответствует порядку 5 и положительной нормализованной мантиссе, полученной из двоичного числа 10100, равного 20_{10} . Объединяя по три соседних двоичных разряда в один восьмеричный, получим в разрядах $48 \div 37$ восьмеричное число 4252.

в) Отрицательная константа ,REAL, -1, будет транслирована в 4020 0000 0000 0000 (мантисса в дополнителном коде с нормализацией и с 1 в 41-м разряде).

Константы типа INT и REAL аналогичны целым и вещественным константам фортрана, т. е. их запись и машинное представление идентичны.

З а м е ч а н и е. Перевод чисел из десятичной системы в двоичную в трансляторах с автокода и с фортрана (и алгола) производится разными подпрограммами. Поэтому возможны отличия в трех младших разрядах мантисс машинных представлений одинаковых вещественных констант, транслированных разными трансляторами (если эти константы не могут быть представлены в машине точно).

Константы типа ISO (см. приложение 2) могут содержать до 128 символов, которым предшествует указатель nH , где n — число символов. Например, текстовая константа

,ISO, 6HBESM-6

содержит 6 символов в восьмиразрядной кодировке ISO. В результате трансляции этой константы будет сформировано одно машинное слово. Отметим, что здесь пробелы после символа H уже не игнорируются и входят в общее число символов константы.

При желании любой символ может быть задан своим кодом ISO, т. е. в виде восьмеричного числа (не превосходящего 377B), заключенного в апострофы ('). Например, рассмотренная выше константа может быть

записана в виде

,ISO, 6HB'105''123'M-6

Здесь вместо букв E и S указаны их представления в кодировке ISO (см. приложение 2). Заметим, что таким способом можно задавать любые восьмиразрядные комбинации 0 и 1, а не только те из них, которые соответствуют какому-либо символу ISO. Ниже будет рассмотрено использование этого способа для записи восьмеричных констант (см. § 48). Отметим, что символ «апостроф» может быть задан только в восьмеричном виде, например,

,ISO, 1H'47'

Важно отметить, что для записи констант типа ISO можно использовать часть перфокарты по 72-ю колонку включительно. Если в указателе nH значение n превышает число символов, информация о которых содержится по 72-ю колонку, то вместо недостающих символов будут добавлены пробелы.

Если $n = 0$ или значение n не указано, то транслятор полагает $n = 6$ и формирует машинное слово, состоящее из 6 пробелов. Если n не делится нацело на 6, то транслятор добавляет необходимое для получения целого числа машинных слов количество пробелов. Однако это действие не выполняется, если вслед за оператором с мнемокодом ISO будет написан оператор с мнемокодом CONT. Его адресная часть должна быть аналогична адресной части предыдущего оператора. Оператор с мнемокодом CONT в свою очередь может иметь продолжение.

П р и м е р. Текстовая константа

, ISO, 9HBESM-6 — JI
T: , CONT, 2HNR
, CONT, 6H — DUBNA

состоит из 17 символов. В результате трансляции будет образовано три машинных слова, содержащих указанный текст, дополненный справа одним пробелом (в 8 младших разрядах третьего слова).

Метка в операторах ISO и CONT совпадает с адресом слова, в которое помещен первый символ из адресной части соответствующего оператора. В приведенном

примере метка Т соответствует второму машинному слову (содержащему символ N).

Константы типа GOST полностью аналогичны константам предыдущего типа с той лишь разницей, что кодирование символов при образовании машинных слов будет выполнено в соответствии с кодом АЦПУ-128 (приложение 3).

Константы типа ISO аналогичны текстовым (холеритовским) константам фортрана. Использование констант типа GOST связано со спецификой печатающего устройства БЭСМ-6. Необходимость в такой кодировке возникает при выдаче на печать текстовой информации путем непосредственного обращения к экстракоду печати (см. [22]), что используется довольно редко.

Константы типа TEXT (приложение 4) отличаются от предыдущих кодировкой, а также тем, что в одно машинное слово записывается 8 символов, так как каждый символ кодируется 6 двоичными разрядами. Эта кодировка используется для внутреннего представления текстовой информации в системе «Дубна». В частности, в кодировке TEXT представлены все наименования подпрограмм в библиотечных каталогах. Поэтому при работе с библиотеками (см. [27], стр. 114—118) часто используется эта кодировка.

Константы типа TEL имеют 5-разрядную кодировку (см. приложение 5). Однако упаковка символов в машинные слова производится по 6 штук. При этом пятиразрядный код дополняется двумя нулями слева и одним нулем справа, превращаясь в восьмиразрядный. Указатели телетайпных регистров могут добавляться транслатором автоматически с соответствующим увеличением длины константы. Отметим, что константы этого типа обычно используются только в системных программах.

§ 41. Адреса типа «литерал»

Адресами типа «литерал» называются адресные части команд, имеющих вид

=	<восьмеричное число>
=:	<восьмеричное число>
=I	<целое десятичное число>
=R	<вещественное число>
=nH	<список символов>

Знак = является признаком адреса типа «литерал». В первом случае восьмеричное число определяет константу типа LOG, во втором случае — типа OCT. Символы I, R и n указывают, что константы имеют тип INT, REAL и ISO соответственно. В последнем случае n не должно превосходить 6, так как каждая команда может оперировать с одним машинным словом.

При появлении в полном адресе команды одной из указанных конструкций транслятор формирует соответствующую ей константу и заносит ее в машинное слово, которое автоматически резервируется в конце подпрограммы. Всюду, где встретится рассматриваемый адрес типа «литерал», он будет заменен транслятором на адрес указанного машинного слова.

П р и м е р ы.

10, XTA, = -77 , A * X, = R .5E6
 , AAX, = : 774 , XTS, = 3NABC
 , XTA, = I2 14, VTM, = R1.

Если адреса типа «литерал» имеют одинаковые указатели (скажем, R) и в результате трансляции приводят к одинаковым машинным словам, то такие адреса считаются эквивалентными и для определяемых ими констант будет отведена одна и та же ячейка. Однако такая экономия выполняется строго внутри данного класса адресов типа «литерал» и не затрагивает идентичных (в смысле машинного представления) констант, определенных другими способами,

П р и м е р ы.

, XTA, = R1.
 , A * X, = R. 99999 99999 999
 , A / X, = : 4050
 , A + X, = 6H'202''200''0''0''0''0'

В приведенных примерах все 4 константы будут транслированы в одно и то же машинное слово 4050 0000 0000 0000 (вторая константа будет транслирована идентично первой из-за ограничения машинной точности 12 десятичными знаками). Однако эквивалентными будут считаться лишь адреса первых двух констант, т. е. всего будет сформировано 3 машинных слова, в каждом из которых будет одна и та же машинная константа, написанная выше,

Использование констант типа «литерал» очень удобно, так как это облегчает написание программ и делает их текст более наглядным. Однако в этом случае мы не можем предвидеть порядок расположения транслированных констант внутри подпрограммы.

§ 42. Описания

Идентификаторы, используемые в указателе индекс-регистра и в адресной части команд (а также в некоторых других операторах) и не являющиеся метками, должны быть каким-то образом определены. Для этого служат описания.

Описать идентификатор — это значит поставить ему в соответствие некоторое число (адрес некоторой ячейки памяти, номер индекс-регистра, адрес типа N и т. д.). Любой идентификатор может быть описан, вообще говоря, не более чем одним способом. Метки, как уже говорилось, сами являются описаниями.

Одним из часто используемых способов описания идентификаторов является резервирование для них ячеек памяти внутри подпрограммы. Эти ячейки используются для внутренних нужд подпрограммы (например, для хранения промежуточных результатов). Резервирование участка памяти осуществляется посредством автокодного оператора вида

<метка> : , BSS, <полный адрес>

П р и м е р ы.

A : , BSS, 1
* C : , BSS,
TAB : , BSS, 100
T : , BSS, 25B
Z : , BSS, T—A

Здесь для идентификатора A зарезервирована одна ячейка памяти. Метке * C соответствует конструкция BSS с пустой адресной частью. Это значит, что адрес метки * C совпадает с адресом следующего за ней машинного слова, в данном случае с адресом метки TAB. Для идентификатора TAB отведен массив из 100 машинных слов. Это означает, что в подпрограмме можно использовать адреса вида $TAB + n$ ($0 \leq n \leq 99$),

каждый из которых будет соответствовать $(n+1)$ -му слову из этого массива. Из примеров видно также, что количество резервируемых слов можно задавать в виде восьмеричного числа (с обязательной буквой В после числа). Последний из приведенных операторов означает, что число ячеек памяти, резервируемых для идентификатора Z, равно разности адресов Т и А, т. е. в данном случае десятичному числу 101.

Отметим особую роль конструкции вида

⟨метка⟩ : , BSS, ⟨пусто⟩

уже встречавшейся ранее. Эта конструкция не резервирует ячеек памяти, а служит всего лишь меткой следующего за ней машинного слова. Указанное машинное слово может в свою очередь иметь метку и содержать любую информацию, т. е. быть командой, константой и т. д.

Например, запись

A : , BSS,
* 3 : 14, XTA,

означает, что написанная команда имеет две эквивалентные метки А и * 3. Однако при необходимости между этими двумя операторами могут быть вставлены некоторые команды или другие операторы, и тогда указанные метки, вообще говоря, перестанут быть эквивалентными.

Опытные программисты, как правило, помечают нужные команды посредством «пустого» BSS, т. е. не привязывают метку жестко к определенной команде. Это дает возможность вставлять или изымать перфокарты, «эквивалентить» различные метки и т. д., не нарушая структуры подпрограммы.

Заметим, что описанная конструкция является автокодным аналогом фортранного оператора CONTINUE.

В тех случаях, когда для идентификатора не требуется резервировать память (например, если идентификатор использован в указателе индекс-регистра) или когда ему можно поставить в соответствие уже зарезервированные ячейки памяти, используется описание с помощью конструкции «эквивалентность».

Общий вид этой конструкции

⟨идентификатор⟩ : , EQU, ⟨полный адрес⟩

Примеры.

```
A : , EQU, 17B  
C : , EQU, A+3  
L : , EQU, * +5  
T : , EQU, =R1.
```

Эквивалентности можно располагать в любом месте подпрограммы с соблюдением единственного условия: к моменту своего появления в адресной части этого оператора идентификатор должен быть уже описан (например, через другие эквивалентности, написанные ранее, или через метки).

Отметим, что идентификаторы, которым по смыслу соответствуют номера индекс-регистров или адреса типа N (см. § 36), могут быть описаны только через эквивалентности или их модификации.

Эквивалентности чаще всего располагают в начале подпрограммы, соблюдая порядок их следования. Отметим, что оператор эквивалентности имеет много общего с фортранным оператором EQUIVALENCE.

Рассмотренных выше способов описания обычно бывает достаточно для несложных подпрограмм. Отметим важное свойство адресов ячеек памяти, описанных рассмотренными способами. Все эти адреса являются *внутренними* адресами данной подпрограммы. Это означает, что адрес любого такого идентификатора отличается от начального адреса подпрограммы на некоторое (конкретное для каждого идентификатора) число, не зависящее от расположения подпрограммы в памяти машины. Именно такие внутренние адреса подпрограммы могут базироваться при соблюдении дополнительных условий (см. § 47).

Мы рассмотрели способы описания внутренних объектов подпрограммы. Прежде чем перейти к описаниям внешних объектов, рассмотрим еще три типа операторов с мнемосодами NAME, END и CALL.

Мнемосокод NAME является составной частью заголовка подпрограммы, который записывается в одной из трех форм

```
<идентификатор> : , NAME,  
<идентификатор> : <индекс-регистр>, NAME,  
<идентификатор> : <индекс-регистр>, NAME, ***
```

Наиболее часто используется первая из приведенных форм записи.

Мнемокод END служит признаком конца записи подпрограммы (аналогично фортранному оператору END).

Мнемокод CALL служит для вызова внешних подпрограмм с помощью оператора

,CALL, <идентификатор вызываемой подпрограммы>

Возможны и другие, нестандартные способы вызова подпрограмм (см. ниже).

Внешними объектами данной подпрограммы могут быть массивы, являющиеся общими для нескольких подпрограмм (в частности, аналоги фортранных COMMON-блоков), а также массивы специального вида, необходимые для организации обмена информацией с магнитными барабанами и магнитными лентами (дисками). Для описания таких внешних массивов в автокоде предусмотрено несколько типов операторов, наиболее удобным из которых является оператор с мнемокодом BLOCK. Этот оператор имеет вид

$\langle \text{идентификатор блока} \rangle : \langle \text{характеристика и тип массивов} \rangle, \text{BLOCK}, \langle \text{список массивов} \rangle$

Характеристика массива определяется одной из букв:

L — *массив* — можно разместить, начиная с произвольной ячейки памяти;

P — *страничный* массив, начальный адрес которого должен быть кратным 1024_{10} ;

S — *секторный* массив, начинающийся с адреса, кратного 256_{10} .

Тип массива указывается одной из следующих букв:

P — *собственный* массив, т. е. массив, недоступный другим подпрограммам;

U — *несобственный* массив;

C — *общий* массив.

В настоящее время тип U не задействован и понимается как P. Отметим, что вместо массивов с указателем LP часто удобнее использовать конструкцию BSS, так как в этом случае массив будет внутренним и его можно базировать (см. § 47).

Общие массивы, в отличие от собственных, доступны любым подпрограммам, в которых они описаны теми же

идентификаторами блоков. Если идентификаторы блоков, описывающих общие массивы, удовлетворяют определенным требованиям (см. § 46), то эти блоки являются автокодными аналогами фортранных COMMON-блоков.

В списке массивов указываются идентификаторы массивов и далее в скобках длины этих массивов. При отсутствии скобок длина массива полагается равной 1, а при отсутствии длины внутри скобок она полагается равной 0, и тем самым начальный адрес описываемого массива будет совпадать с началом массива, следующего за ним в списке.

Под массивы, описанные конструкцией BLOCK, резервируется участок памяти длиной, равной сумме длин массивов, указанных в списке этого блока. При этом адрес начала первого массива совпадает с адресом идентификатора блока.

П р и м е р ы.

A : LC, BLOCK, B(3), D(5), CD, EF(), * SQ(12)

Здесь описан общий массив (начинающийся с произвольного адреса) длиной 21 машинное слово. При этом длина массива CD равна 1, а начальные адреса массивов EF и * SQ совпадают,

* ABC * : LC, BLOCK, A, B(7), C(18)

Этот блок является автокодным налогом фортранного COMMON-блока, описанного оператором

COMMON /ABC/ A, B(7), C(18)

В листе загрузки (см. [27], стр. 50) идентификаторы фортранных COMMON-блоков записываются «по-автокодному», т. е. обрамляются звездочками.

Конструкция BLOCK может располагаться по 72-ю колонку перфокарты включительно. Список в ее адресной части может быть продолжен применением конструкции CONT.

П р и м е р,

A : LC, BLOCK, B (3)
 , CONT, D (5), CD
 , CONT, EF (), * SQ (12)

Это другая запись рассмотренного ранее примера.

Вместо описания группы массивов с помощью конструкции BLOCK можно использовать оператор описания отдельного массива с последующими эквивалентностями. Рассмотренный выше пример можно записать следующим образом:

```

A : , LC, 21
B : , EQU, A
D : , EQU, B+3
CD : , EQU, D+5
EF : , EQU, CD+1
* SQ : , EQU, EF

```

Здесь первый оператор описывает массив A длиной, равной общей длине блока (резервируя тем самым участок памяти длиной 21 слово), а все остальные массивы, составляющие блок, описываются в виде цепочки эквивалентностей. Этот пример иллюстрирует удобство конструкции BLOCK.

К числу внешних объектов подпрограммы относятся все подпрограммы, которые ею вызываются. Как сказано выше, вызов подпрограммы можно осуществить с помощью оператора с мнемокодом CALL. Этот стандартный способ вызова не требует (но и не исключает) специального описания вызываемой подпрограммы. Отметим, что в этом случае возврат в вызывающую подпрограмму производится на левую команду следующей ячейки, адрес которой автоматически запоминается в 13-м индекс-регистре.

Вызов подпрограммы можно осуществить также оператором

13, VJM, <идентификатор подпрограммы>
или, например, так:

```

13, VTM, * 10
    , UJ, <идентификатор подпрограммы>
    . . . . .
* 10 : , BSS,

```

т. е. с возвратом на заранее предписанную ячейку (в данном случае на метку * 10). Во всех подобных случаях необходимо описание вызываемой подпрограммы, что делается оператором

<идентификатор подпрограммы> , SUBP,

Рассмотренный оператор является аналогом фортранного оператора EXTERNAL и, следовательно, он необходим в тех случаях, когда вызов подпрограммы производится через посредство другой подпрограммы (т. е. путем указания наименования вызываемой подпрограммы в качестве фактического параметра при вызове подпрограммы-посредника).

Рассмотрим теперь оператор вида

<идентификатор> : ENTRY,

который служит для описания дополнительного входа в подпрограмму аналогично фортранному оператору ENTRY.

Дополнительные входы позволяют объединить несколько подпрограмм в одну, с тем чтобы использовать одни и те же константы и не описывать лишних общих блоков. Однако каждый вход является самостоятельной подпрограммой в том смысле, что обращение к любому входу полностью аналогично вызову подпрограммы. Общее число входов в подпрограмму (включая основной, описанный заголовком) не должно превосходить 20. Входы, к которым есть обращения, описываются аналогично описаниям вызываемых подпрограмм.

Отметим, что, в отличие от фортрана, тип любого автокодного ENTRY (SUBROUTINE, FUNCTION и т. п.), равно как и список его формальных параметров, не обязан совпадать с таковыми у основного входа (см. § 46).

Заметим, что использование любого входа в подпрограмму внутри самой подпрограммы возможно либо с помощью оператора с мнемокодом CALL, либо при наличии дополнительного описания используемого входа как метки (например, через BSS).

Заметим также, что команда, следующая за оператором с мнемокодом ENTRY, считается помеченной.

Укажем еще несколько описательных конструкций, используемых сравнительно редко.

При составлении системных подпрограмм (например, работающих в диспетчерском режиме) могут быть известны абсолютные адреса некоторых команд и констант. Для описания таких адресов используются специальные разновидности конструкции BLOCK, позволяющие задавать начальные адреса блоков как в десятичной, так и в восьмеричной системе.

П р и м е р ы.

* 20 : B, BLOCK, A (23), D (2)

* 25 : , BLOCK, A (15), B (6)

Первая конструкция эквивалентна описаниям

A : , EQU, 20B

D : , EQU, A+23

Вторая конструкция равнозначна описаниям

A : , EQU, 25

B : , EQU, A+15

Еще одна разновидность конструкции BLOCK позволяет описывать последовательности эквивалентностей, что видно из примера

M : I, BLOCK, A, B (15), C (6)

Эта конструкция эквивалентна описаниям

A : , EQU, M

B : , EQU, A+1

C : , EQU, B+15

Идентификатор M, естественно, должен быть уже описан каким-то способом (например, через BSS). Это одно из немногих исключений из правила, согласно которому допускается не более чем однократное описание идентификатора в виде метки (см. § 39).

Отметим, что рассмотренные три разновидности конструкции BLOCK не резервируют ячеек памяти.

Рассмотрим еще несколько разновидностей эквивалентности. Конструкция вида

<идентификатор> : , WEQ, <полный адрес>

называется *косвенной* эквивалентностью. Она приписывает идентификатору значение, равное содержимому 1÷15 разрядов слова с заданным полным адресом в момент загрузки подпрограммы.

Косвенная эквивалентность позволяет, в частности, резервировать (например, посредством оператора BSS) участки памяти переменной длины. В этом случае длина участка может быть задана, например, с помощью оператора DATA, написанного в PROGRAM. Ячейка памяти, куда засылается нужное значение

длины (в виде целой константы) должна быть, естественно, элементом COMMON-блока, описанного в автокодной подпрограмме.

Конструкция

$\langle \text{идентификатор} \rangle : , R * R, \left(\left\langle \begin{smallmatrix} \text{полный} \\ \text{адрес} \end{smallmatrix} \right\rangle \right) \left(\left\langle \begin{smallmatrix} \text{полный} \\ \text{адрес} \end{smallmatrix} \right\rangle \right)$

приписывает идентификатору значение, равное произведению полных адресов (по модулю 2^{16}), приведенных в адресной части.

Эта конструкция позволяет удобным образом резервировать участки памяти под многомерные массивы, в том числе под массивы с переменными размерностями (в сочетании с косвенной эквивалентностью).

Аналогично, конструкция

$\langle \text{идентификатор} \rangle : , R / R, \left(\left\langle \begin{smallmatrix} \text{полный} \\ \text{адрес} \end{smallmatrix} \right\rangle \right) \left(\left\langle \begin{smallmatrix} \text{полный} \\ \text{адрес} \end{smallmatrix} \right\rangle \right)$

приписывает идентификатору значение, равное частному (округленному до целого) от деления полных адресов, указанных в адресной части. Одно из возможных применений этой конструкции — определение номера страницы или абзаца памяти по заданному значению адреса. Необходимость в этом возникает, например, при обращении к экстракоду обмена * 70 (см. § 6).

З а м е ч а н и е. В некоторых подпрограммах, составленных до 1970 г., в качестве заголовка использовалась конструкция

$: , SUBP, \langle \text{идентификатор} \rangle$

являющаяся «наследием» автокода SIBESM-6 [38]. Использовать ее в качестве заголовка не рекомендуется.

§ 43. Параметрические команды

Иногда бывает удобно задавать константы в формате команд. Использование для этих целей команд не всегда возможно, так как не всякая константа может быть представлена в виде команды. Для записи констант в виде команд служат так называемые пара-

метрические команды. Мнемокод параметрической команды имеет вид

Z <восьмеричное число от 0 до 37В>

Параметрические команды транслируются как команды с длинным адресом.

П р и м е р.

5, Z31, 06412В

транслируется в машинную команду

05 31 06412

§ 44. Данные и рассылки

В ряде случаев возникает необходимость занести некоторую исходную информацию в определенные ячейки памяти, не являющиеся внутренними адресами подпрограммы. Для этого служат специальные операторы автокода, располагаемые в конце подпрограммы. Эти операторы описывают величины, подлежащие рассылке (данные), и адреса ячеек памяти, куда эти величины рассылаются (указания о рассылке). В подпрограмме данные предшествуют указаниям о рассылке.

Группа данных начинается с оператора

,DATA,

и может состоять из констант любого типа.

Указания о рассылке представляются последовательностью пар операторов вида

$$\begin{array}{l} n_1, \text{ SET}, A_1 \\ n_2, \quad \quad , A_2 \end{array}$$

где n_1 и n_2 — целые десятичные числа, A_1 и A_2 — полные адреса. Такая пара операторов осуществляет пересылку группы из n_1 слов, начинающейся с адреса A_1 , в массив ячеек, начинающийся с адреса A_2 , n_2 раз.

Пересылать можно любые объекты подпрограммы (команды, константы), так что при наличии указаний о рассылке группа данных может и отсутствовать.

Отметим, что данные не загружаются в память машины. Загрузчик производит их рассылку согласно указаниям в подпрограмме, после чего уничтожает

«оригиналы». Таким образом, данные можно использовать лишь по их новым адресам. Чаще всего это адреса из общих блоков.

Пр и м е р. Запись вида

```

      , DATA,
A : , REAL, 1.
      , ISO, 14HABТOКOД — MADLEN
      4, SET, A
      1,      , TABLE

```

означает однократную рассылку написанных выше констант (занимающих 4 машинных слова, начиная с адреса A) в ячейки памяти, начиная с адреса TABLE. При этом указанные константы расположатся в машинных словах TABLE и от TABLE+1 до TABLE+3 соответственно. По этим «новым» адресам (но не по «старым»!) их можно использовать в подпрограмме.

Рассмотренные конструкции выполняют те же функции, что и фортранный оператор DATA.

§ 45. Комментарии

Комментарии в автокоде можно записывать по фортранным правилам (буква C в первой колонке перфокарты-комментария). Кроме того, на каждой перфокarte для комментария отведены колонки с 73-й по 80-ю. В большинстве случаев (за исключением операторов с мнемосодами ISO, BLOCK, CONT и некоторых других «длинных» операторов) поле комментария начинается с 43-й колонки. Для того чтобы запись оператора можно было продолжить за 42-ю колонку, в первой колонке надо пробить управляющий символ L. Однако необходимости в этом обычно не возникает. Наоборот, часто возникает необходимость размещения комментария ранее 43-й (или 73-й) колонки. Для этого служит точка, которая отделяет комментарий от адресной части команды.

Пр и м е р ы.

```

      , AAX, = : 774. ВЫДЕЛЕНИЕ ПОРЯДКА.
      , E+N, 64. НОРМАЛИЗАЦИЯ.

```

§ 46. Правила оформления автокодных подпрограмм

При оформлении автокодных подпрограмм допускается нестандартность. Однако соблюдение определенных правил при оформлении автокодной подпрограммы позволяет ей, с одной стороны, не «мешать» работе других подпрограмм и, с другой стороны, вызывать ее некоторым стандартным образом (в том числе из фортранных и алгольных подпрограмм). Такая автокодная подпрограмма может быть включена в библиотеку программ общего пользования.

Первая группа правил касается использования индекс-регистров и режимов работы арифметического устройства.

Эти правила предусматривают, что индекс-регистры 1÷7 можно использовать лишь при условии последующего восстановления их прежнего состояния. Индекс-регистры 8÷12 и 14 можно использовать без последующего восстановления. Это означает, что при выходе из любой подпрограммы состояние этих индекс-регистров не определено, т. е. указанные индекс-регистры могут быть «испорчены».

Индекс-регистр 13 играет особую роль — в нем хранится адрес возврата. Это означает, что если некоторая подпрограмма в свою очередь вызывает другую подпрограмму (скажем, посредством оператора CALL), то свой адрес возврата она должна, вообще говоря, сохранить (например, путем записи содержимого индекс-регистра 13 в некоторую ячейку памяти).

Индекс-регистр 14 является «рабочим» регистром. Его состояние «портится» любым экстракодом (остальные индекс-регистры экстракодами не «портятся»).

Индекс-регистр 15 является счетчиком магазина и устанавливается мониторной системой. Стандартное его значение равно 53401В, 55401В или 73401В (в зависимости от объема памяти, выделенной для задачи). Объем магазина во всех случаях равен 377В. Переполнение или исчерпание магазина диагностируется при счете.

Соглашение о регистре режима и блокировок предусматривает, что значение этого регистра при входе в автокодную подпрограмму равно 6 и должно быть таким же на выходе. Напомним, что это стандартное

значение соответствует режиму выполнения арифметических операций с нормализацией и с блокировкой округления и, кроме того, значению признака группы «логическая». Установка этого стандартного значения проще всего делается командой ,NTR, 6.

В т о р а я группа правил касается способа передачи фактических параметров при совместном использовании фортранных (алгольных) и автокодных подпрограмм.

При вызове фортранной подпрограммы, имеющей параметры, из автокодной необходимо загрузить адреса фактических параметров вызываемой подпрограммы в магазин в порядке их следования. Пусть, например, мы хотим оформить вызов подпрограммы аналогично фортранному оператору

CALL SUB (A, B, C)

Тогда надо написать такую последовательность операторов (не обязательно с использованием индекса-регистра 14):

14, VTM, A
 , ITS, 14
14, VTM, B
 , ITS, 14
14, VTM, C
 , ITS, 14
 , CALL, SUB

Как видим, магазин «проталкивается» вниз столько раз, сколько фактических параметров. При этом сначала в магазин записывается содержимое сумматора, которое должно быть восстановлено перед выходом из подпрограммы SUB.

Если подпрограмма SUB аналогична фортранной FUNCTION, а не SUBROUTINE, то вызов ее будет отличаться тем, что после возврата из такой подпрограммы необходимо выполнить одну из команд магазинного считывания. Чаще всего выполняют команду с мнемокодом STX. При этом происходит запись значения функции, полученного на сумматоре, в ячейку памяти и одновременно восстановление состояния сумматора, которое было до вызова подпрограммы.

Иногда при вызове FUNCTION адрес первого параметра засылают не в магазинном режиме (,ITA, 14

вместо ,ITS, 14). В этом случае после возврата из подпрограммы дополнительного магазинного считывания не делают, оставляя на сумматоре значение функции.

Извлечение адресов фактических параметров из магазина производится в порядке, обратном порядку их записи в магазин. Напомним, что адрес последнего из них находится на сумматоре. Пусть мы составляем на автокоде подпрограмму, аналогичную фортранной SUBROUTINE SUB (A, B, C). Тогда извлечение адресов фактических параметров можно сделать, например, командами

```
, STI, 14.C  
, STI, 12.B  
, ATI, 11.A
```

Как видим, в магазине осталось прежнее значение сумматора. Перед выходом из подпрограммы надо установить правильное состояние счетчика магазина, например, командой ,STX, либо 15 ,XTA, . Отметим, что после выполнения любой из этих команд будет установлен нужный признак группы «логическая». В случае FUNCTION такого дополнительного магазинного считывания делать нельзя. Иначе говоря, FUNCTION с n параметрами должна произвести «выталкивание» магазина $n-1$ раз, а SUBROUTINE — n раз.

Заметим, что в случае большого числа параметров для извлечения их адресов из магазина указанным способом может не хватить индекс-регистров. В этом случае надо либо записывать адреса параметров в некоторые ячейки памяти (как делается, например, в фортранных подпрограммах), либо извлекать их по частям, что требует известной аккуратности. Поскольку в автокодных подпрограммах число формальных параметров обычно невелико, то индекс-регистров чаще всего хватает.

В случае когда на автокоде пишется аналог фортранной SUBROUTINE без параметров, эта подпрограмма должна сохранить состояние сумматора (т. е. выполнить на входе 15, ATX, либо ,XTS, или ,ITS,). При выходе надо проделать обратную операцию восстановления состояния сумматора.

Аналогом фортранной PROGRAM на автокоде является подпрограмма, имеющая заголовок

PROGRAM : ,NAME,

либо один из входов

PROGRAM : ,ENTRY,

Третья группа правил касается выбора наименований подпрограмм (входов), совместимых с фортраном (алголом), и общих блоков, совместимых с фортранными COMMON-блоками.

Для того чтобы автокодную подпрограмму (либо отдельный ее вход) можно было вызывать из фортранной (алгольной) подпрограммы, наименование этой подпрограммы (или входа) должно быть идентификатором, допустимым для фортрана или алгола. Это означает, что рассматриваемый идентификатор должен состоять не более чем из 6 символов и не содержать символов * и /.

Аналогичное правило для идентификаторов общих блоков, совместимых с фортранными COMMON-блоками, выглядит так: идентификатор общего блока должен начинаться и оканчиваться звездочкой, а между звездочками должен содержать не более 6 символов, допустимых для фортранного идентификатора. Например, общий блок вида

* BLOCK 1 * : LC, BLOCK, A (10), B (25)

имеет тот же смысл, что и фортранный оператор

COMMON/BLOCK1/ A (10), B (25)

Описание же вида

BLOCK1 : LC, BLOCK, X (5), Y (10)

не имеет фортранного аналога и, стало быть, этот блок недоступен никакой фортранной подпрограмме.

В тех случаях, когда автокодная подпрограмма не предназначается для использования в фортранных или алгольных подпрограммах, ее наименование лучше выбрать недоступным для них, равно как и наименование ее общих блоков.

Приведем теперь схему построения подпрограммы на автокоде

Заголовок
Команды
Константы и BSS
Данные
Указания о рассылке
,END,

Комментарии можно располагать ранее заголовка и в любом месте до ,END, . Описания могут располагаться в любом месте после заголовка с единственным условием: описание любого идентификатора (кроме меток) должно предшествовать его использованию. Операторы с мнемокодом BSS и непустой адресной частью (т. е. фактически резервирующие место в памяти) могут перемежаться константами. Их рекомендуется располагать после констант, которые в свою очередь обычно располагаются после команд. Пустые конструкции BSS могут располагаться произвольно среди команд и констант. Данные и указания о рассылке располагаются в конце подпрограммы.

Подпрограмма должна составляться в расчете на то, что она будет помещена загрузчиком в произвольную область памяти. Это означает, что адреса команд с коротким адресом (кроме некоторых) должны быть, вообще говоря, каким-то образом модифицированы. Этому важному вопросу посвящен следующий параграф.

§ 47. Базирование

Как сказано выше, многие команды с коротким адресом требуют модификации своей адресной части. Самым простым способом такой модификации является запись вида

, UTC, A
, XTA,

вместо записи

, XTA, A

Возможна и такая запись:

J, VTM, A
J, XTA,

Однако эти способы неэкономны, так как для модификации каждой команды обычно требуется дополнительная команда. В случае большого числа команд с коротким адресом более эффективным приемом модификации адресов является базирование.

Суть базирования состоит в следующем. Большинство команд с коротким адресом оперирует либо с абсолютными адресами, допустимыми для таких команд (ASN, E+N, YTA, и т. п.), либо с внутренними адресами данной подпрограммы. Каждый внутренний адрес подпрограммы можно представить в виде $*C+D$, где $*C$ — адрес начала подпрограммы и D не превосходит длины подпрограммы, которая обычно не превышает 07777В. Указанные команды с коротким адресом (являющимся внутренним адресом подпрограммы) вида

$\langle \text{мнемокод} \rangle, A$

переписывают так:

$J, \langle \text{мнемокод} \rangle, A - *C$

При этом в индекс-регистр J должен быть занесен адрес базы, т. е. в начале участка программы должна быть выполнена команда

$J, VTM, *C$

В результате такого видоизменения команд с коротким адресом их исполнительные адреса остаются прежними, но в адресной части всюду получают значения, допустимые для команд с коротким адресом.

В автокоде madlen предусмотрена возможность автоматического базирования, при котором не требуется переписывать команды, подлежащие базированию. Автоматическое базирование может производиться несколькими способами.

Наиболее употребительным является базирование по одному индекс-регистру или локальное базирование. На протяжении этой главы термином «базирование» обозначается именно такой способ базирования. Этим способом могут базироваться лишь внутренние адреса подпрограммы. Базирование выполняется на линейных участках подпрограммы, каждый из которых открывается заказом на базирование (см. ниже) и закрывается

либо очередным заказом на базирование, либо отменой базирования (или оператором ,END,).

Базируются только команды, удовлетворяющие условиям:

- 1) команда должна быть с коротким адресом,
- 2) команда должна быть с пустым указателем индекса-регистра,
- 3) полный адрес команды должен быть внутренним адресом подпрограммы.

Команды с непустым (в том числе и нулевым) указателем индекса-регистра, полные адреса которых являются внутренними адресами подпрограммы, транслируются в пару команд. Например, команда вида

M, XTA, A

транслируется так же, как пара команд

, UTC, A
M, XTA,

Заказ на базирование (без установки базового индекса-регистра) имеет вид

I, BAS, * C

Здесь * C есть адрес базы, I — базовый индекс-регистр. Более часто используют оператор вида

I, BASE, * C

который, кроме заказа на базирование, производит и установку базы. Этот оператор эквивалентен паре операторов

I, BAS, * C
I, VTM, * C

Отмена базирования производится либо новым заказом на базирование, либо оператором

I, BAS, <пусто>

Отметим, что в качестве адреса базы можно использовать любой внутренний адрес A такой, что для каждого внутреннего адреса K выполняется условие: $A - 10000B \leq K \leq A + 7777B$. В качестве базового регистра часто используется один из индекса-регистров 8÷12. В случае, когда подпрограмма содержит вызовы

других подпрограмм, удобнее использовать в качестве базового регистра один из регистров 1÷7.

Заметим, что точные сведения о том, какие адреса пробазированы, а какие нет, всегда можно получить из текста подпрограммы на языке загрузки, расположенного на листинге слева от автокодного текста (см. § 49). В случае базирования транслированная команда будет содержать базовый индекс-регистр.

Другим способом базирования является глобальное базирование всей памяти с помощью трех индекс-регистров. Суть этого способа (используемого в основном в больших программах) состоит в следующем. Адреса от —10000В до 07777В не требуют базирования. Остальные адреса могут быть базированы с помощью трех индекс-регистров, равных соответственно 20000В, 40000В и 60000В, так как любой из этих адресов отстоит от одного из трех указанных значений индекс-регистров не далее чем на 10000В влево и не далее чем на 07777В вправо. Заказ на глобальное базирование задается заголовком подпрограммы вида

<идентификатор>: R, NAME,

где R — целое десятичное число от 1 до 13. Базирование производится загрузчиком, который подбирает надлежащий индекс-регистр из набора R, R+1, R+2 для каждой из команд, подлежащих базированию. Значения индекс-регистров с номерами R, R+1, R+2, равные 20000В, 40000В и 60000В соответственно, устанавливаются самой подпрограммой.

При глобальном базировании базируются все команды с коротким адресом, пустым указателем индекс-регистра и адресной частью, превышающей 07777В. Тем самым исключается базирование команд с мнемокодами ASN, NTR и т. п., адреса которых базировать не надо. Команды с коротким адресом и непустым указателем индекс-регистра заменяются двумя командами, как описано ранее. Если такие команды базировать не надо, то заголовок подпрограммы должен быть видоизменен так:

<идентификатор>: R, NAME, ***

Параметрические команды во всех случаях не базируются,

Следующие два способа базирования предназначены в основном для использования при составлении системных подпрограмм.

Иногда возникает необходимость оформления подпрограммы в виде, позволяющем использовать ее на любом участке памяти без предварительной «настройки» адресов. Такую «перемещаемую» подпрограмму можно получить, например, путем базирования ее адресов по некоторому базовому адресу.

Заказ на такое базирование производится оператором вида

M, REL, B

Здесь в качестве M обычно указывается индекс-регистр 14 (устанавливаемый загрузчиком), а в качестве B — начальный адрес базирuемого участка подпрограммы (например, начальный адрес подпрограммы).

При необходимости базирования адресов, зависящих от внешних объектов, заказ на базирование задается оператором вида

M, RELS, B

Отмена любого из двух видов базирования производится оператором

M, REL, <пусто>

§ 48. Примеры автокодных подпрограмм

Приведем примеры несложных автокодных подпрограмм, из которых читателю станет яснее, как на практике используются различные конструкции языка madlen.

Рассмотрим подпрограмму вычисления скалярного произведения двух N-мерных векторов. Оформим эту подпрограмму как FUNCTION SCAL (A, B, N)

<pre> SCAL: NAME, FUNCTION SCAL (A, B, N) , STI, 14.N , STI, 12.B , ATI, 11.A , NTR, 3 14 , XTA, , UTC, =11. (*) , X-A, . 1-N , ATI, 14 , NTR, 18 , XTA, </pre>	<pre> *1:, BSS, 11, XTS, 12, A * X, 11, UTM, 1 12, UTM, 1 15, A+X, 14, VLM, * 1 , NTR, 6 13, UJ, , END, </pre>
---	--

Как видно из текста подпрограммы, базирование здесь не понадобилось, так как используется всего одна команда (*), удовлетворяющая всем условиям базирования.

Рассмотрим теперь подпрограмму перекодировки целого числа, не превосходящего $2^{18}-1$, в код ISO. Здесь перекодировка означает, что каждая восьмеричная цифра этого числа перекодировается в соответствующую цифру, заданную в коде ISO (см. Приложение 2). Например, целое десятичное число 21, представленное в машине как 6400 0000 0000 0025, будет перекодировано в машинное представление, соответствующее константе ,ISO, 6H000025.

На первый взгляд решение этой задачи представляется сложным. Однако с помощью команды разборки (с мнемокодом AUMX) получается очень простое и изящное решение. Сначала надо сдвинуть содержимое сумматора влево так, чтобы 18 младших его разрядов переместились в старшие разряды. Затем производится разборка 18 старших разрядов на все машинное слово так, чтобы каждая восьмеричная цифра оказалась в трех младших разрядах соответствующего байта. При этом пять старших разрядов каждого байта будут нулями. Теперь для получения нужного результата достаточно логически сложить содержимое сумматора с константой, имеющей 60В в каждом байте. Запишем решение в виде следующей автокодной подпрограммы.

```
INTISO:, NAME, .FUNCTION INTISO (INT)
      8, BASE, *
      , ATI, 14. INT
    14 , XTA,
      , ASN, 64-30
      , AUX, =6H'7'7'7'7'7'7'7'
      , AOX, =6H'60'60'60'60'60'60'60'
    13, UJ,
      , END,
```

Мы использовали кодировку ISO для задания восьмеричных констант. Здесь это удобнее, чем использование констант типа OCT или LOG.

§ 49. Стандартный массив

Стандартный массив (или модуль загрузки) является результатом работы транслятора с автокода. Он включает в себя транслированную подпрограмму на языке загрузки и таблицу описаний, содержащую информацию, необходимую загрузчику для размещения подпрограммы в оперативной памяти. Текст стандартного массива печатается слева от автокодного текста при условии задания соответствующего режима печати (см. § 51).

Каждая команда печатается в две строки по формату машинных команд. Верхняя строка, которая соответствует левой команде, снабжается восьмеричным относительным адресом данной команды в подпрограмме, печатаемым слева от команды.

Индекс-регистр указывается в восьмеричном виде, а адресная часть состоит из 4 восьмеричных цифр у команд с коротким адресом и из 5 восьмеричных цифр у команд с длинным адресом.

Неполные машинные слова дополняются правыми командами

0022 00000

В случае, если команда базируется, в ней указывается базовый индекс-регистр, а адресная часть равна разности между полным адресом данной команды и адресом базы. Если этот относительный адрес оказался отрицательным, то в адресной части указывается адрес $4000B + n$, где n есть номер строки в таблице описаний (это справедливо для всех отрицательных коротких адресов). Если используется адрес типа «литерал», то он базируется при наличии заказа на базирование. В случае, когда базируемая команда имеет нулевой указатель индекс-регистра, она транслируется в две команды (см. § 47).

Если базирование не заказано, то любой короткий адрес, не являющийся абсолютным, будет иметь вид $4000B + n$, т. е. будет оформлен в виде ссылки на таблицу описаний. Заметим, что абсолютные адреса без модификации допустимы лишь в случаях, когда им соответствуют не ячейки памяти, а некоторые другие объекты (например, в командах с мнемокодами ASN, NTR, ATI и т. п.). Исключение составляют физические адреса и адреса специальных ячеек мониторинговой системы, используемые, как правило, в системных подпрограммах.

Длинный адрес вида $40000B + n$ означает ссылку на n -й адрес относительно начала подпрограммы, а длинный адрес вида $74000B + n$ — ссылку на n -ю строку таблицы описаний. Отметим, что начальная (нулевая) строка таблицы описаний содержит идентификатор данной подпрограммы в коде TEXT.

§ 50. Диагностика ошибок

Транслятор с автокода обнаруживает всевозможные формальные ошибки в подпрограмме, не связанные со способом загрузки ее в память машины. Это означает, что «незаконное» использование коротких адресов не может быть обнаружено транслятором.

В зависимости от вида ошибки игнорируется либо весь ошибочный оператор, либо его часть. К каждому ошибочному оператору может быть выдано несколько диагностических текстов, причем они, как правило, печатаются непосредственно перед ошибочным оператором.

Некоторые ошибки прекращают дальнейшую трансляцию. К числу таких ошибок относятся (в скобках указана диагностика):

1) отсутствие заголовка подпрограммы (ОТСУТСТВУЕТ ЗАГОЛОВОК ПОДПРОГРАММЫ),

2) слишком большое число используемых идентификаторов либо наличие более 20 входов (ПЕРЕПОЛНЕНА ТАБЛИЦА ОПИСАНИЙ),

3) для размещения подпрограммы требуется более 23 листов оперативной памяти (ДЛИНА ПОДПРОГРАММЫ ПРЕВЫШАЕТ ВОЗМОЖНОСТИ МАШИНЫ),

4) при обнаружении свыше 100 ошибочных операторов (ОЧЕНЬ МНОГО ОШИБОК),

5) при некоторых ошибках в адресе оператора BSS (НЕЯВНООПРЕДЕЛЕННЫЙ ИДЕНТИФИКАТОР BSS).

Остальные ошибки не препятствуют дальнейшей трансляции. Среди них наиболее часто встречаются ошибки, связанные с использованием неописанных идентификаторов либо идентификаторов, описанных более одного раза (дважды описанных). Отметим, что обнаружение таких идентификаторов входит в функции транслятора с автокода на этапе трансляции фортран-ных подпрограмм с языка madlen на язык загрузки (остальные ошибки обнаруживаются транслятором с фортрана).

§ 51. Управляющие карты, редактирование, сервис

Тексту автокодной подпрограммы в пакете задачи должна обязательно предшествовать управляющая карта

*** ASSEMBLER**

которая служит признаком того, что далее следует автокодный текст. При отсутствии других управляющих карт (если печать не отменена картой * NO_LIST) пользователь получает отредактированный текст автокодной подпрограммы, напечатанный в два столбца (билистинг), без текста стандартного массива.

Для получения текста стандартного массива необходимы две подряд стоящие управляющие карты

*** CALL — PUTFLAG ***
n 1

Здесь *n* может принимать значение 0, 4 или 6. Если *n* = 0, то печать производится в два столбца. При *n* = 4 (или 6) печать производится в один столбец. При печати в виде билистинга «длинные» операторы будут распечатаны в две строки.

При наличии управляющей карты * FULL_LIST после текста подпрограммы будет выдана таблица описаний, таблица ссылок и список неиспользованных идентификаторов.

Автокодный текст, выдаваемый на печать, редактируется, т. е. разделители (двоеточия, запятые и точки) печатаются в определенных позициях, метки и адресные части выравниваются по левым символам, а указатели индекс-регистров — по правым. Ошибочные операторы печатаются без редактирования.

Имеется возможность частичной распечатки текста подпрограммы, что делается с помощью управляющих карт * FULL_LIST и * NO_LIST, расположенных внутри подпрограммы. При этом режим печати, определенный в пакете перед подпрограммой, запоминается и восстанавливается после ее трансляции.

Внутри текста подпрограммы (до первой команды) может быть поставлена управляющая карта

*** MOSU_A**

где А — пятизначный восьмеричный адрес. В этом случае текст подпрограммы будет печататься не с относительной адресацией (т. е. не с адреса 00000), а начиная с адреса А.

§ 52. Советы и рекомендации

При автономной отладке автокодных подпрограмм, когда пакет задачи будет малым по объему, отлаживаемые подпрограммы часто будут загружаться по адресам памяти, не превосходящим 07777В, и тем самым при загрузке не будут выявлены случаи использования коротких адресов без модификации. При загрузке по адресам, превосходящим 07777В, такая «отлаженная» подпрограмма может не пойти. Для исключения подобных случаев необходимо в отладочной PROGRAM описать фиктивный массив длиной не менее 3600. Во всяком случае, необходимо проследить, чтобы отлаживаемая подпрограмма была загружена в память, начиная с адреса не менее 10000В.

Если, наоборот, необходимо загрузить подпрограмму на адреса до 07777В, то в PROGRAM не должно быть больших массивов и соответствующая подпрограмма должна быть описана в PROGRAM оператором EXTERNAL (или его автокодным аналогом). Тогда она будет загружена вслед за PROGRAM. Отметим, что подпрограммы, описанные в операторе EXTERNAL, загружаются в обратном порядке, т. е. первой загрузится та, что описана последней.

При составлении подпрограмм с многими входами необходимо помнить, что каждый из этих входов должен составляться как самостоятельная подпрограмма, т. е. содержать команды извлечения адресов фактических параметров из магазина, заказ на базирование и установку базы и т. д. Любой такой вход должен оканчиваться выходом по команде 13, UJ, (либо ей эквивалентной), выполняющей те же функции, что и фортранный оператор RETURN.

В процессе отладки автокодной подпрограммы может возникнуть необходимость отладочных печатей. Имеется целая серия системных подпрограмм, позволяющих удобно осуществлять такую печать.

Для печати текстовой строки (заданной в коде ISO) можно воспользоваться системной подпрограммой PRINT8. Обращение к ней является нестандартным и выглядит так:

```
14, VTM, <начальный адрес информации>  
  , ITS, 14  
14, VTM, <конечный адрес информации>  
  , ITS, 14  
14, VTM, <номер начальной позиции на АЦПУ>  
  , ITS, 14  
  , CALL, PRINT8
```

Нестандартность здесь состоит в том, что последним засылается на сумматор не адрес номера начальной позиции, а сам номер начальной позиции. При этом счет позиций идет от 0, так что роль управляющей позиции (см. § 29, п. 7) играет нулевая позиция. Необходимо соблюдать осторожность при пользовании этой управляющей позицией, а еще лучше здесь ею не пользоваться.

Вместо PRINT 8 можно использовать подпрограмму PRINTA со стандартным обращением, аналогичным фортранному оператору CALL PRINTA (АНАЧ, АКОН, НПОЗ)

Отметим, что за одно обращение к подпрограмме PRINT 8 или PRINTA можно отпечатать только одну текстовую строку, излишек текста печати будет обрезан.

Для печати массива чисел в формате, аналогичном фортранному формату E, служит системная подпрограмма PRINTE. Обращение к ней аналогично фортранному оператору

```
CALL, PRINTE (АНАЧ, АКОН, N, M)
```

Здесь АНАЧ и АКОН — начальный и конечный адрес печатаемого массива чисел, N — количество чисел, печатаемых в одной строке, M — число знаков после запятой. Массив печатается слева направо по N чисел в каждой строке, с одним пробелом между числами. Отметим, что число позиций, отводимых под одно число, равно $M+7$.

Печать восьмеричных чисел (аналогично фортранному формату O) можно сделать с помощью подпрограммы PRINTO, обращение к которой аналогично

обращению к PRINTЕ. При этом должно быть $M \leq 16$. Если $M < 16$, то старшие разряды восьмеричной константы игнорируются.

Более подробное изложение рассматриваемых вопросов имеется в [27] (стр. 195—200).

§ 53. Некоторые приемы программирования на автокоде

К использованию автокода чаще всего прибегают в тех случаях, когда возникает необходимость работать с частями машинного слова. Поэтому в большинстве таких подпрограмм отдельные операторы производят упаковку информации, другие — ее распаковку. Пользователю нелишне ознакомиться с некоторыми приемами упаковки и распаковки.

Упаковку обычно делают так. Сначала заносят нуль в нужное машинное слово, затем присоединяют к нему справа очередные разряды путем логического сложения с предварительным левым сдвигом на соответствующее число разрядов. При таком способе упаковки слово заполняется слева направо, причем каждый раз производится левый сдвиг уже упакованной части слова на столько разрядов, сколько требуется для очередного фрагмента, располагаемого в младших разрядах слова.

Пример. Пусть надо присоединить справа к машинному слову PACK очередные 6 разрядов, расположенные в младших разрядах слова BIT (остальные разряды этого слова произвольны). Это можно сделать такими командами:

```
, XTA, PACK  
, ASN, 64—6  
, XTS, BIT  
, AAX, =77  
15, AOX,  
, ATX, PACK
```

Если упаковку надо производить справа налево, то очередной фрагмент располагают в старших разрядах слова и каждый раз производят правый сдвиг на нужное число разрядов с последующим логическим сложением.

Распаковку удобно производить с использованием регистра младших разрядов. При распаковке слева направо, т. е. начиная со старших разрядов слова, сначала производится сдвиг влево, затем запись «остатка» в исходное машинное слово с последующей выдачей выделенного фрагмента из регистра младших разрядов в младшие разряды сумматора. При распаковке справа налево сначала производится сдвиг вправо, затем запись «остатка» в исходное слово и выдача выделенной части из регистра младших разрядов в старшие разряды сумматора.

П р и м е р. Пусть надо распаковать 10 старших разрядов слова DEP, получив в исходном слове оставшиеся младшие разряды, а распакованные разряды — в слове INT в виде целого числа. Указанные операции можно проделать с помощью такой группы команд:

```
, XTA, DEP
, ASN, 64—10
, ATX, DEP
, YTA,
, AOX, =10
, ATX, INT
```

В обоих рассмотренных примерах предполагается, что будет сделано базирование коротких адресов.

Среди арифметических операций известную сложность представляют операции над ненормализованными (в частности, целыми) числами, особенно умножение и деление. Последние удобно производить с помощью специальных системных подпрограмм, именуемых $I * MU * I$ и $I * DI * I$ соответственно. Первый операнд необходимо поместить в сумматор, а адрес второго занести в индекс-регистр 14. Например, умножение целого числа по адресу M на целое число по адресу N можно осуществить командами

```
, XTA, M
14, VTM, N
, CALL, I * MU * I
```

Результат умножения, взятый по модулю 2^{40} , выдается на сумматор в виде целого числа. Аналогично можно производить операции над комплексными величинами и величинами с двойной точностью (см. [27], стр. 64).

З а к л ю ч е н и е. Мы описали практически все конструкции автокода madlen по состоянию на 1975 г. При дальнейшем развитии этого языка не исключено появление новых конструкций, расширяющих его возможности.

Программист, пользующийся автокодом, должен помнить, что транслятор с автокода выполняет лишь функции «переводчика» с одного языка на другой. Это означает, что все машинные операции, которые должны быть выполнены в подпрограмме, необходимо задать в виде автокодных команд. Исключение составляет лишь рассылка данных, выполняемая загрузчиком согласно указаниям о рассылке. Транслятор с автокода, естественно, требует, чтобы исходная информация, т. е. запись подпрограммы на языке madlen, была правильной, полной (отсутствие неописанных идентификаторов) и непротиворечивой (отсутствие дважды описанных идентификаторов).

Диагностика выдается лишь при нарушении правил записи операторов на входном языке, а также при явном несоответствии задачного оператора сути соответствующей команды (например, команды с мнемосодами VTM, VLM или VJM и пустым указателем индекса регистра).

ГЛАВА IV

ОПТИМИЗАЦИЯ ФОРТРАННЫХ ПОДПРОГРАММ

В этой небольшой главе мы вкратце рассмотрим некоторые вопросы оптимизации фортранных подпрограмм. Под оптимизацией здесь будет пониматься локальная (т. е. не затрагивающая общей структуры) модификация подпрограммы или комплекса подпрограмм, позволяющая ускорить их работу или уменьшить требуемый объем памяти.

Многие подпрограммы устроены так, что основная доля счетного времени в них падает на выполнение сравнительно небольшой последовательности операторов (самого внутреннего цикла). В этом случае нет смысла заниматься оптимизацией всей подпрограммы. Достаточно оптимизировать выполнение того внутреннего цикла, который забирает основную долю общего счетного времени.

В ряде случаев не имеет смысла оптимизировать подпрограмму по причине малого абсолютного выигрыша во времени. Например, если подпрограмма работает 5 минут и используется не слишком часто, то нет особой необходимости ее оптимизировать.

§ 54. Оптимизирующие возможности транслятора с фортрана

Для того чтобы умело пользоваться различными приемами оптимизации, полезно знать оптимизирующие возможности транслятора. На БЭСМ-6 транслятор с фортрана производит оптимизацию вычисления арифметических выражений в пределах одного оператора. Эта оптимизация состоит в следующем:

1) Функции от одних и тех же аргументов, встречающиеся в данном арифметическом выражении, вычисляются один раз для всего выражения. Например, в операторе

$$Y = 2. * \sin(X ** 2 + C) + 1. / (\sin(X ** 2 + C) + 2.)$$

вычисление функции $\sin(X ** 2 + C)$ будет выполнено один раз.

2) Возведение в степень, показатель которой является целой константой, не превосходящей 8, транслятор заменяет последовательностью умножений. Например, оператор

$$Y = X ** 5$$

будет преобразован транслятором в последовательность команд, состоящую из трех команд умножения (и нескольких команд считывания и записи).

3) Константы и простые переменные, не являющиеся элементами COMMON-блоков или формальными параметрами, имеют адреса, базированные по 7-му индекс-реgistру (см. § 47). Это означает, что операции над этими величинами выполняются без дополнительных команд, необходимых для образования исполнительного адреса в командах с коротким адресом.

Отметим разницу в выполнении операций возведения в одну и ту же степень, заданных различными способами. Например, возведение вещественного X в степень 3 можно задать следующими 4 способами:

$$\begin{aligned} X ** 3 \\ X ** N, \text{ где } N=3 \\ X ** 3. \\ X ** Y, \text{ где } Y=3. \end{aligned}$$

Первая запись будет преобразована транслятором в последовательность из двух умножений.

Во втором случае транслятор сформирует обращение к системной подпрограмме $R * PO * I$, которая будет анализировать показатель N и производить нужные умножения. Здесь также будет выполнено два умножения, но в процессе анализа показателя степени N потребуется выполнить около десятка команд на каждое такое умножение. Тем самым время счета возрастает примерно на порядок по сравнению с первым случаем.

В двух последних случаях возведение в степень будет выполнено не через операции умножения, а как

$$\text{EXP}(3. * \text{ALOG}(X))$$

и

$$\text{EXP}(Y * \text{ALOG}(X))$$

соответственно. Это означает, в частности, что возведение в такую степень возможно лишь для $X > 0$. Отметим, что вычисление экспоненты и логарифма потребует примерно в 30 раз больше времени, чем вычисление $X ** 3$ в первом случае.

В заключение скажем еще об оптимизации арифметических выражений. Назовем *подвыражением* арифметического выражения его часть, заключенную в скобки либо состоящую из операции вычисления функции.

Транслятор строит программу так, что одинаковые подвыражения (самого внутреннего уровня) вычисляются один раз. Поэтому, например, арифметическое выражение

$$B * (X ** 2 + 3.) + (X ** 2 + 3.) + A$$

будет вычислено быстрее, чем выражение

$$B * (X ** 2 + 3.) + X ** 2 + 3. + A$$

§ 55. Оптимизация путем изменения способа адресации величин

По способу адресации величины, используемые в подпрограмме, можно разделить на три типа: внутренние величины, элементы COMMON-блоков и формальные параметры.

Образование исполнительных адресов для внутренних величин производится путем базирования по 7-му индекс-регистру, т. е. без дополнительных команд модификации адреса.

Исполнительные адреса для элементов COMMON-блоков образуются с помощью команды UTC.

Формальные параметры адресуются косвенно, через команду WTC, которая выполняется медленно и замедляет выполнение других команд.

Отсюда следует, что операции над внутренними величинами выполняются наиболее быстро и реали-

вуются посредством минимального числа машинных команд. Операции над элементами COMMON-блоков требуют большего числа команд и выполняются несколько медленнее операций над внутренними величинами. Операции над формальными параметрами занимают столько же места в памяти ЭВМ, что и операции над элементами COMMON-блоков, но выполняются значительно медленнее их. Поэтому переход к использованию внутренних величин вместо элементов COMMON-блоков и формальных параметров позволяет сэкономить счетное время и место в памяти.

Пусть, например, в некоторой подпрограмме простая переменная X служит формальным параметром и часто используется. Тогда введение дополнительного оператора присваивания $A = X$ с последующим использованием внутренней переменной A вместо X позволит сэкономить и счетное время, и место в памяти.

Замена формальных параметров элементами COMMON-блоков может дать выигрыш во времени счета, но такая замена не всегда удобна, да и не всегда возможна. В таких случаях иногда бывает выгодно перейти к автокоду (см. § 57).

§ 56. Оптимизация индексных выражений

Адресация элементов дву- и трехмерных массивов требует вычисления индексных функций, определяющих их относительные адреса в данном массиве. При этом существенно, что вычисляются сразу все индексные функции, зависящие от данной индексной переменной. Поэтому для образования различных индексных выражений иногда выгодно использовать отличные друг от друга индексные переменные. Например, запись вида

$$\begin{aligned} & \text{DO } 1 \text{ } I=1, 20 \\ & 1 \text{ } A(2 * I+46)=0. \\ & \text{DO } 2 \text{ } I=1, 15 \\ & 2 \text{ } B(I+1, 2 * I)=1. \end{aligned}$$

можно оптимизировать, использовав во втором цикле другую индексную переменную.

В ряде случаев нет необходимости использовать элементы трехмерного массива как переменные с тремя

индексами или элементы двумерного массива как переменные с двумя индексами. Например, вместо записи

```
DO 1 I=1, M
DO 1 J=1, N
DO 1 K=1, L
1 A (I, J, K)=0.
```

можно написать

```
MNL=M * N * L
DO 1 I=1, MNL
1 A (I)=0.
```

В некоторых случаях можно уменьшить число индексов путем введения эквивалентности. Например, в записи

```
DIMENSION A (30, 20)
D=0.
DO 1 I=1, 30
1 D=D+A (I, 5) * A (I, 15)
```

можно ввести две дополнительных эквивалентности и переписать цикл в более экономном виде. Это можно сделать, например, так:

```
DIMENSION A (30, 20), B (30), C (30)
EQUIVALENCE (A (121), B), (A (421), C)
D=0.
DO 1 I=1, 30
1 D=D+B (I) * C (I)
```

Здесь элемент A(1,5), представленный как A(121), эквивалентен первому элементу массива B, и элемент A(1,15), представленный как A(421), эквивалентен первому элементу массива C.

В настоящее время разработан вариант транслятора, позволяющий в ряде случаев оптимизировать выполнение самых внутренних циклов [31], [32].

§ 57. Оптимизация с помощью автокода

Многие пользователи знают, что переход к автокоду позволяет в ряде случаев получить существенный выигрыш во времени счета. Однако программировать на автокоде гораздо сложнее, чем на фортране (или

алголе). Разумный компромисс состоит в том, что в виде автокодных подпрограмм пишутся небольшие блоки фортранных (или алгольных) подпрограмм, представляющие собой самые внутренние циклы.

Пусть, например, написана подпрограмма

```
SUBROUTINE SCALAR (A, B, C, M, N)
  DIMENSION A (M, N), B (M, N), C (N)
  DO 2 J=1, N
    S=0.
    DO 1 I=1, M
      1 S=S+A (I, J) * B (I, J)
      2 C (J)=S
    RETURN
  END
```

Эта подпрограмма вычисляет скалярные произведения соответственных столбцов двумерных массивов A и B и заносит результаты в одномерный массив C. Здесь можно, конечно, провести оптимизацию, заменив во внутреннем цикле двумерные индексы на одномерные. Однако ввиду того, что массивы A и B являются формальными параметрами, внутренний цикл все равно будет выполняться довольно долго. Если M достаточно велико (скажем, не меньше 10), то оформление внутреннего цикла в виде простой автокодной подпрограммы даст значительную экономию времени. Такая подпрограмма уже была рассмотрена в § 48. Здесь остается ею воспользоваться. Итак, перепишем нашу подпрограмму в виде

```
SUBROUTINE SCALAR (A, B, C, M, N)
  DIMENSION A (M, N), B (M, N), C (N)
  I=1
  DO 2 J=1, N
    C (J)=SCAL (A (I), B (I), M)
  2 I=I+M
  RETURN
  END
```

Таким образом, оптимизация фортранных подпрограмм путем перехода к автокоду будет эффективной, если придерживаться следующих рекомендаций:

1. Нет необходимости всю подпрограмму переписывать на автокоде. Достаточно записать в виде авто-

кодной подпрограммы (или подпрограммы-функции) лишь ту часть фортранной подпрограммы, которая забирает основную долю счетного времени.

2. Наиболее выгодно записывать на автокоде небольшие циклы. Эффект от перехода к автокоду увеличивается, если действия производятся с формальными параметрами.

§ 58. Способы экономии памяти.

Сегментация задачи

До сих пор мы рассматривали способы экономии счетного времени (в ряде случаев попутно экономились и память). Рассмотрим теперь способы экономии памяти (иногда, быть может, ценой увеличения счетного времени).

Максимальный объем оперативной памяти, доступный пользователю БЭСМ-6, составляет 72400В машинных слов, т. е. несколько более 29 листов. Необходимость в экономии памяти обычно возникает в тех случаях, когда задача не помещается в этот максимальный объем памяти.

Рассмотрим следующие способы экономии памяти:

1. Экономия в рамках отдельных подпрограмм путем описания дополнительных эквивалентностей.

2. Экономия в рамках всей задачи за счет введения дополнительных COMMON-блоков.

3. Экономия за счет отказа от форматного способа ввода-вывода информации.

4. Экономия путем сегментации задачи.

Первый способ экономии состоит в том, что выясняются дополнительные возможности описания эквивалентностей. Если, например, описаны массивы

DIMENSION A(1000), B(2000)

и сначала используется массив А, а затем производится запись в массив В, то без ущерба для работы подпрограммы можно «заэквивалентить» начала этих массивов.

Второй способ позволяет экономить память в рамках всей задачи. При этом описываются дополнительные COMMON-блоки, и каждая подпрограмма, имеющая возможность передать часть своих внутренних

массивов в «общее пользование», описывает некоторые из этих блоков. Иногда выгодно описать один большой COMMON-блок, с тем чтобы его могли использовать подпрограммы, которым требуются большие массивы. Для устранения возможного разбаланса в длинах блоков рекомендуется в головной программе (PROGRAM) описать их на максимальную длину.

Применяя этот способ экономии памяти, необходимо помнить, что при переходе от внутренних переменных к элементам COMMON-блоков происходит увеличение длины каждой из подпрограмм, описывающих соответствующие COMMON-блоки, за счет дополнительных команд UTC. Иногда такая «экономия» может привести к увеличению общего объема занимаемой памяти. Поэтому данный способ годится, вообще говоря, лишь для экономии массивов достаточно большой длины.

Третий способ экономии памяти состоит в том, что форматный ввод и вывод информации производится не с помощью операторов ввода-вывода и операторов FORMAT, а путем использования специальных системных подпрограмм. Это позволяет сэкономить примерно 1000 ячеек памяти. При таком способе экономии памяти придется отказаться от ввода данных с перфокарт и значительно ограничить возможности вывода информации на печать. Вывод на печать текстовой информации в этом случае можно произвести с помощью системной подпрограммы PRINTA, а распечатку чисел (аналогично формату E) — с помощью подпрограммы PRINTE (см. § 52).

Сегментация задачи. Универсальным способом экономии памяти является сегментация задачи, которая состоит в разбиении ее на разделы, сменяющие друг друга в памяти. Рассмотрим конкретный пример.

Пусть схема взаимодействия отдельных подпрограмм задачи выглядит так (см. рис. 4).

Из схемы видно, что нет необходимости одновременно держать в памяти все подпрограммы. Сначала из PROGRAM достаточно вызвать подпрограмму S1, которая в свою очередь вызовет подпрограммы S11, S12 и S13. По окончании работы подпрограммы S1 на ее место можно загрузить S2, S21 и S31, оставив в памяти только S13. После того как сработает подпро-

грамма S2, на ее место можно загрузить S3 и S32, оставив в памяти только S31.

Таким образом, в памяти будут одновременно находиться лишь подпрограммы из каждого раздела S1, S2 и S3. Если и при таком способе сегментации все еще не хватает памяти, то можно произвести дополнительную сегментацию внутри каждого из разделов S1, S2 и S3. Например, внутри раздела S1 можно образовать подразделы S11, S12 и S13, поочередно загружая их в память,

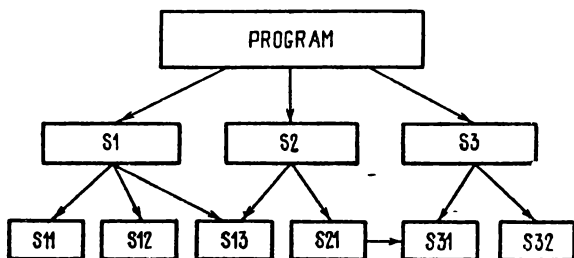


Рис. 4.

Для того чтобы подпрограмма образовала сменяемый раздел, ее необходимо вызывать не непосредственно, а с помощью системной подпрограммы LOADGO. Обращение к ней имеет вид

CALL LOADGO (P₁, P₂, ..., P_n, 'S')

Здесь S — наименование подпрограммы, заданное в виде текстовой константы; P₁, P₂, ..., P_n — список ее фактических параметров при данном обращении. Например, вместо оператора

CALL PSI (Z1, A (15, 3))

пишется оператор

CALL LOADGO (Z1, A (15, 3), 'PSI') или
CALL LOADGO (Z1, A (15, 3), ЗНPSI)

Наименование подпрограммы, вызываемой через LOADGO, должно состоять не более чем из 6 символов и не содержать русских букв. При этом идентификатор подпрограммы должен быть указан без пробелов. Отметим,

что для образования сменяемого раздела необходимо, чтобы все вызовы его головной подпрограммы из всех подпрограмм были оформлены через LOADGO.

Сегментация на данном уровне осуществляется лишь в том случае, если на этом уровне имеется по меньшей мере два сменяемых раздела.

Рассмотрим конкретный пример сегментации для приведенной выше схемы. Будем предполагать, что все подпрограммы не имеют параметров.

PROGRAM TOLEG	SUBROUTINE S1
CALL LOADGO ('S1')	CALL LOADGO ('S13')
CALL LOADGO ('S2')	CALL LOADGO ('S13')
CALL LOADGO ('S1')	CALL S12
CALL S3	CALL LOADGO ('S11')
CALL LOADGO ('S1')	CALL LOADGO ('S13')
END	RETURN
	END

Остальные подпрограммы вызываются непосредственно оператором CALL.

Работа программы будет происходить следующим образом. PROGRAM и все подпрограммы, не охваченные сегментацией (т. е. S3, S31 и S32), будут загружены в несменяемую область памяти. Затем будет загружен раздел S1 и подпрограмма S12 внутри этого раздела. Затем произойдет загрузка S13. При повторном вызове LOADGO с требованием на загрузку S13 выяснится, что данная подпрограмма уже загружена и повторной загрузки производиться не будет. Затем на место раздела S13 будет загружен раздел S11, после чего снова будет загружен S13 на место S11. После выхода из S1 на его место будет загружен раздел S2 и подразделы S13 и S21. Затем раздел S2 сменится разделом S1, который останется в памяти до окончания задачи.

При сегментации задачи надо учитывать наличие COMMON-блоков. Эти блоки необходимо дополнительно описать в одном из разделов, охватывающих все раз-

дела, в подпрограммах которых описаны COMMON-блоки.

Пусть, например, в нашей схеме подпрограммы S11 и S21 имеют общий блок. Если не предпринять никаких дополнительных мер, то этот блок для подпрограммы S11 будет загружен вместе с разделом S1, а блок для подпрограммы S21 — вместе с разделом S2, который сменит раздел S1. Для того чтобы этот COMMON-блок мог функционировать, необходимо описать его в PROGRAM либо в одной из подпрограмм несегментируемой части задачи. Если имеется COMMON-блок для S11 и S13, то он должен быть описан в S1 или в любой из несегментируемых подпрограмм.

Для того чтобы отменить сегментацию некоторых подпрограмм внутри данного раздела, нет необходимости переписывать операторы вызова этих подпрограмм. Для этого достаточно описать эти подпрограммы в операторе EXTERNAL одного из «вышестоящих» разделов. Например, если в разделе S1 написать оператор EXTERNAL S11, то тем самым будет отменена сегментация этой подпрограммы внутри раздела S1. Описание ее в операторе EXTERNAL, расположенном в PROGRAM, приведет к тому, что S11 будет загружена в несменяемую область памяти.

Отметим, что сегментация задачи требует дополнительного машинного времени на перезагрузку разделов. В случае, когда разделы перезагружаются многократно, это может привести к значительному увеличению времени счета. Поэтому наиболее эффективный способ сегментации состоит в том, что самые внутренние, т. е. наиболее часто используемые подпрограммы, описываются оператором EXTERNAL в PROGRAM, попадая тем самым в несменяемую область памяти. Остальные, более редко используемые подпрограммы, сегментируются в той мере, в какой позволяет наличный объем памяти.

Отметим также, что при каждой перезагрузке происходит выдача списка вновь загружаемых подпрограмм. При отладке сегментируемой задачи не всегда удобно отключать эту выдачу. При счете по отлаженной программе можно использовать системную карту * NO LOAD LIST, отключающую печать всех таблиц загрузки,

Описанный способ сегментации (называемый динамическим), требует при каждой перезагрузке разделов производить настройку входящих в них подпрограмм по месту их нового расположения в памяти. При очень частой смене разделов это может вызвать значительные дополнительные затраты времени. Поэтому при запуске больших задач, требующих очень частой смены разделов, может оказаться выгодным использовать статический способ сегментации. Суть этого способа состоит в том, что до выхода задачи на счет формируется (на МБ) библиотека статических разделов, в которую записываются отдельные разделы, настроенные на фиксированные адреса памяти. Загрузка каждого такого раздела в процессе счета всякий раз будет производиться на одни и те же заранее определенные адреса памяти.

При использовании этого способа сегментации необходимо описать структуру каждого раздела (см. [27]).

1. СИСТЕМА КОМАНД БЭСМ-6

КОП	Содержимое		Группа	Время выполнения				Содержание операции	
	Мнемо-код	А		У	В УУ	В АУ			
						min	средн.		max
000	ATX	не изм.	не изм.	не изм.	3	3	3	Запись слова в память	
001	STX	изм.	не изм.	логич.	6	5	6	Запись и магазинное считывание	
003	XTS	изм.	не изм.	логич.	6	5	6	Магазинная запись и считывание	
004	A+X	изм.	изм.	слож.	3	5	11	280	Сложение
005	A-X	изм.	изм.	слож.	3	5	11	280	Вычитание
006	X-A	изм.	изм.	слож.	3	5	11	280	Обратное вычитание
007	AMX	изм.	изм.	слож.	3	5	11	280	Вычитание модулей
010	XTA	изм.	не изм.	логич.	3	2	2.5	3	Считывание слова из памяти
011	AAx	изм.	нужь	логич.	3	4	4	4	Логическое умножение
012	AEX	изм.	изм.	логич.	3	2	2.5	3	Поразрядное сравнение
013	ARX	изм.	нужь	умнож.	3	3	6	27	Циклическое сложение
014	AVX	изм.	нужь	слож.	3	4	5	25	Изменение знака числа
015	AOX	изм.	нужь	логич.	3	4	4	4	Логическое сложение
016	A/X	изм.	не опр.	умнож.	3	47	50	174	Деление
017	A*X	изм.	изм.	умнож.	3	15	18	162	Умножение
020	APX	изм.	нужь	логич.	3	53	53	53	Сборка
021	AUX	изм.	нужь	логич.	3	53	53	53	Разборка
022	ACX	изм.	нужь	логич.	3	54	56	78	Подсчет числа единиц
023	ANX	изм.	изм.	логич.	3	7	32	78	Определение номера старшей единицы
024	E+X	изм.	нужь	умнож.	3	3	5	133	Сложение порядков
025	E-X	изм.	нужь	умнож.	3	3	5	133	Вычитание порядков
026	ASX	изм.	изм.	логич.	3	4	нет	68	Сдвиг по коду порядка

КОП	Мнемо-код	Содержимое		Группа	Время выполнения				Содержание операции
		А	У		В УУ	В АУ			
						min	средн.	max	
027	XTR	не изм.	не изм.	регистр.	3	2	2.5	3	Установка режима
030	RTE	изм.	не изм.	не изм.	3	2	3	3	Выдача содержимого рег. признаков
031	UTA	изм.	не изм.	не изм.	3	3	5	133	Выдача младших разрядов
034	E+N	изм.	нуль	умнож.	3	3	5	133	Сложение порядка с адресом
035	E-N	изм.	нуль	умнож.	3	3	5	133	Вычитание из порядка адреса
036	ASN	изм.	изм.	логич.	3	3	нет	68	Сдвиг по адресу
037	NTR	не изм.	не изм.	регистр.	3	2	2.5	3	Установка режима по адресу
040	ATI	не изм.	не изм.	не изм.	14	3	3	3	Запись из сумматора в индекс-регистр
041	STI	изм.	не изм.	логич.	14	3	3	3	Запись из сумматора в индекс-регистр и магазинное считывание
042	ITA	изм.	не изм.	логич.	6	3	3	3	Считывание индекс-рег. на сумматор
043	ITS	изм.	не изм.	логич.	9	5	6	6	Магазинная запись и считывание индекс-регистра на сумматор
044	MTJ	не изм.	не изм.	не изм.	6	—	—	—	Передача кода из индекс-регистра в индекс-регистр
045	J+M	не изм.	не изм.	не изм.	6	—	—	—	Сложение кода в индекс-регистрах
Команды с длинным адресом									
22	UTC	не изм.	не изм.	не изм.	4	—	—	—	Запись адреса в регистр изменения команд
23	WTC	не изм.	не изм.	не изм.	13	3	3	3	Запись 15 младших разрядов слова в регистр изменения команд

24	VTM	не изм.	не изм.	не изм.	4	—	—	—	Пердача адреса в индекс-регистр
25	UTM	не изм.	не изм.	не изм.	4	—	—	—	Сложение индекс-регистра с адресом
26	UZA	не изм.	нуль	не изм.	15; 12	3	3	3	Условный переход при $\omega=0$
27	U1A*)	не изм.	нуль	не изм.	15; 12	3	3	3	Условный переход при $\omega=1$
30	UJ	не изм.	не изм.	не изм.	7	—	—	—	Безусловный переход
31	VJM	не изм.	не изм.	не изм.	7	—	—	—	Безусл. переход с запом. адреса возврата в инд.-рег.
34	VZM	не изм.	не изм.	не изм.	7; 4	—	—	—	Условный переход по нулевому коду в индекс-регистре
35	V1M*)	не изм.	не изм.	не изм.	7; 4	—	—	—	Условный переход по ненулевому коду в индекс-регистре
37	VLM	не изм.	не изм.	не изм.	7; 4	—	—	—	Конец цикла
32	IJ	не изм.	не изм.	не изм.	7	—	—	—	Переход по индексу (только в режиме диспетчера)

Пр и м е ч а н и е. Коды операций 33 и 36 не используются.

*) Вместо U1A допускается U1A, а вместо V1M — V1M.

1A. ЭКСТРАКОДЫ

КОП	Мне-мокод	КОП	Мне-мокод	КОП	Мне-мокод	КОП	Мне-мокод	КОП	Мне-мокод	КОП	Мне-мокод	КОП	Мне-мокод	КОП	Мне-мокод
050	FUN	053	*53	057	*57	063	*63	066	LJB	070	DRUM	073	*73	075	*75
	*50	054	*54	060	*60	PRINT	PRINT	064	*66	071	*70	074	SJ	076	*76
051	*51	055	*55	061	*61	*64	*64	067	*67	072	*71	075	*74	077	*77
052	*52	056	*56	062	*62	065	*65	070	TAPE		*72		CTX		

2. СИМВОЛЫ ISO

Восьм. код	Символ	Восьм. код	Символ	Восьм. код	Символ	Восьм. код	Символ	Восьм. код	Символ	Восьм. код	Символ	Восьм. код	Символ	Восьм. код	Символ
005	Ъ	042	"	065	5	110	H	132	Z	155	M	156	Н	157	О
006	Х	044	\$	066	6	111	I	133	[156	Н	157	О	160	П
016	≤	045	%	067	7	112	J	135]	157	О	160	П	161	Я
017	≥	046	^	070	8	113	K	136	-	160	П	161	Я	162	Р
020	,	047	(071	9	114	L	137	Ю	161	Я	162	Р	163	С
025	.	050)	072	:	115	M	140	А	162	Р	163	С	164	Т
026	↑	051	*	073	:	116	N	141	Б	163	С	164	Т	165	У
027	₁₀ ≠	052	+	074	<	117	O	142	Ц	164	Т	165	У	166	Ж
030	÷	053	=	075	>	120	P	143	Д	165	У	166	Ж	167	В
031	·	054	-	076	—	121	Q	144	Е	166	Ж	167	В	170	Ь
032	·	055	/	077	A	122	R	145	Ф	167	В	170	Ь	171	Ы
033	≡	056	0	101	B	123	S	146	Г	170	Ь	171	Ы	172	З
034	∇	057	1	102	C	124	T	147	Х	171	Ы	172	З	173	Ш
035	┐	060	2	103	D	125	U	150	И	172	З	173	Ш	174	Щ
036	пробел	061	3	104	E	126	V	151	Й	173	Ш	174	Щ	175	Э
037	—	062	4	105	F	127	W	152	К	174	Щ	175	Э	176	Ч
040		063		106	G	130	X	153	П	175	Э	176	Ч		
041		064		107		131	У	154							

Примечание. Русские буквы, имеющие латинские аналоги, допускают двойную кодировку. Например, букве Н соответствуют коды 110 и 156.

3. СИМВОЛЫ GOST (АЦПУ-128)

Восьм. код	Символ	Восьм. код	Символ	Восьм. код	Символ
000	0	040	А	100	Ф
001	1	041	Б	101	Г
002	2	042	В	102	И
003	3	043	Г	103	Ж
004	4	044	Д	104	З
005	5	045	Е	105	И
006	6	046	Ж	106	Й
007	7	047	З	107	К
010	8	050	И	110	Л
011	9	051	Й	111	М
012	+	052	К	112	Н
013	—	053	Л	113	О
014	/	054	М	114	П
015	,	055	Н	115	Р
016	.	056	О	116	С
017	пробел	057	П	117	Т
020	¹⁰	060	Р	120	У
021	↑	061	С	121	Ф
022	(062	Т	122	Х
023)	063	У	123	Ц
024	×	064	Ф	124	Ч
025	=	065	Х	125	Ш
026	;	066	Ц	126	Щ
027	[067	Ч	127	Ы
030]	070	Ш	130	Ь
031	*	071	Щ	131	ъ
032	•	072	Ы	132	°
033	•	073	Ь	133	'
034	≠	074	Э	134	
035	<	075	Ю	135	
036	>	076	Я	136	
037	:	077	Д	137	

4. СИМВОЛЫ ТЕХТ

Восьм. код	Символ	Восьм. код	Символ	Восьм. код	Символ
00	—	27	7	54	L
02	Б	30	8	55	M
03	Ц	31	9	56	N
04	Д	32	Ь	57	O
05	Ф	34	П	60	P
06	Г	35	—	61	Q
07	И	36	+	62	R
12	*	37	Ы	63	S
13	Й	40	Э	64	T
14	Л	41	А	65	U
15	Я	42	В	66	V
16	Ж	43	С	67	W
17	/	44	Д	70	X
20	0	45	Е	71	Y
21	1	46	Ф	72	Z
22	2	47	Г	73	Ш
23	3	50	Н	74	Э
24	4	51	И	75	Щ
25	5	52	Ј	76	Ч
26	6	53	К	77	Ю

5. СИМВОЛЫ TEL

Восьм. код	Лат.	Рус.	Цифры	Восьм. код	Лат.	Рус.	Цифры
07	A	A		02	Q	Я	1
14	B	Б	?	25	R	Р	4
21	C	Ц	:	13	S	С	
15	D	Д	X	36	T	Т	5
17	E	Е	З	03	U	У	7
11	F	Ф	Э	20	V	Ж	=
24	G	Г	Ш	06	W	В	2
32	H	Х	Щ	10	X	Ь	/
23	I	И	8	12	Y	Ь	6
05	J	Й	Ю	16	Z	З	+
01	K	К	(35	Возвр. кар.	—	
26	L	Л)	27	Перев. стр.	—	
30	M	М	.	00	Лат.	—	
31	N	Н	,	04	Цифры	—	
34	O	О	9	33	Пробел	—	
22	P	П	0	37	Рус.	—	

ЛИТЕРАТУРА

1. М. А. Березовский. Многоязыковая система программирования. В сб. «Развитие программного обеспечения БЭСМ-6». ВЦ АН СССР, Москва, 1975.
2. Библиотека программ на фортрате. ОИЯИ, Б1-11-5144, Дубна, 1970.
3. Библиотека программ на фортране. ОИЯИ, Б1-11-7200, Дубна, 1973.
4. З. Бродзинский, Н. Н. Говоруны и др. Транслятор с языка фортран для системы систематического обеспечения БЭСМ-6. Тр. первой Всесоюз. конференц. по программпр., заседание В, изд. ИК АН УССР, Киев, 1968.
5. В. Ю. Веретенков. Работа с внешней памятью (лентами, барабанами) как устройствами прямого доступа в мониторной системе «Дубна». Информатор № 5, ИАЭ. М., 1974.
6. В. Ю. Веретенков, Н. Н. Говоруны и др. Вариант операционной системы для серийного образца машины БЭСМ-6. Тр. первой Всесоюз. конференц. по программпр., заседание Д, Изд. ИК АН УССР, Киев, 1968.
7. В. Ю. Веретенков, М. И. Гуревич, В. А. Федосеев. Мультидоступная система «Мультитайп» на БЭСМ-6. ИАЭ-2409, Москва, 1974.
8. В. Ю. Веретенков и др. Дисксовая операционная система, ИАЭ-2486, М., 1975.
9. В. Ю. Веретенков и др. Новый диспетчер для ЭВМ БЭСМ-6. ОИЯИ, 11—7059, Дубна, 1973.
10. А. И. Волков. Автокод MADLEN ОИЯИ, Б4-11-4654, Дубна, 1969.
11. А. И. Волков. Автокод MADLEN (описание транслятора). ОИЯИ, 11-5427, Дубна, 1970.
12. А. И. Волков. Дополнение к описанию автокода MADLEN ОИЯИ, 11-5426, Дубна, 1970.
13. А. И. Волков. Редактор текстов. ИАЭ-2351. М., 1974.
14. Н. Н. Говоруны, И. Н. Сплитны и др. Мониторная система «Дубна» для ЭВМ БЭСМ-6. Тр. второй Всесоюзной конф. по программпр., заседание Ж, ВЦ СОАН СССР, Новосибирск, 1970.
15. К. Джермейн. Программирование на IBM/360. Под ред. В. С. Штаркмана. М., «Мир», 1971.
16. Т. Д. Думшева, Ю. М. Морозов, В. К. Федосов. Язык Автокод СОМИ с расширенными возможно-

- стями для машины БЭСМ-6. Тр. первой Всесоюзной, конф. по программир., заседание В, ИК АН УССР, Киев. 1968.
17. Е. А. Жоголев, Сообщение об автокоде СИБЭСМ-6. Сб. работ ВЦ МГУ, вып. 17, 1971.
 18. И. Б. Задыхайло, С. С. Камынин, Э. З. Любимский, М. Р. Шура-Бура. Операционная система ИПМ АН СССР для БЭСМ-6 (ОС ИПМ). Тр. второй Всесоюз. конф. по программир., заседание В, ВЦ СО АН СССР, Новосибирск, 1970.
 19. Н. С. Заикин, И. Н. Силин. Принципы организации общей библиотеки стандартных подпрограмм и работа с ней в системе «Дубна» ЭВМ БЭСМ-6. ОИЯИ, 11-6410, Дубна, 1972.
 20. С. А. Зельдина, В. Ф. Тюрин и др. Структура и функционирование ОС ДИСПАК, Сб. научных трудов № 136, Изд. ЧПИ, Челябинск, 1973.
 21. В. П. Иванников, Л. Н. Королев, А. Н. Томиллин и др. Операционная система НД—69. Тр. второй Всесоюз. конференц. по программир., приглашенные доклады, вып. 2, ВЦ СО АН СССР, Новосибирск, 1970.
 22. Инструкция по программированию на БЭСМ-6. Изд. ИПМ АН СССР, М., 1967.
 23. В. И. Карначук. Язык фортран (Оптимизация и большие задачи). ВЦ СО АН СССР, Новосибирск, 1974.
 24. Л. Н. Королев. Структуры ЭВМ и их математическое обеспечение, М., «Наука», 1974.
 25. В. М. Курочкин, Д. Б. Подшивалов и др. Система БЭСМ-АЛГОЛ. Труды второй Всесоюзной конференции по программир., заседание В, ВЦ СО АН СССР, Новосибирск, 1970.
 26. О. Н. Ломидзе. Перевод стандартных массивов на язык MADLEN, ОИЯИ, 11-7356, Дубна, 1973.
 27. Г. Л. Мазный. Мониторная система «Дубна». ОИЯИ, 115974, Дубна, 1971.
 28. Г. Л. Мазный. Реализация комплексной арифметики с большим диапазоном порядков чисел на БЭСМ-6, ОИЯИ, Дубна, 11-8667, 1975.
 29. Г. И. Макаренко, А. В. Ракитский, А. И. Салтыков. Фортран, М., «Знание», 1973.
 30. Дж. Мак-Кракен, У. Дорн. Численные методы и программирование на фортране, М., «Мир», 1969.
 31. О. Ен Ир, В. П. Шириков. Об одном оптимизирующем варианте обработки циклов при трансляции с языка Фортран на ЭВМ БЭСМ-6. ОИЯИ, 11-9219, Дубна, 1975.
 32. О. Ен Ир, В. П. Шириков. Об оптимизирующем варианте ОРТ-1 обработки циклов при трансляции с языка Фортран на ЭВМ БЭСМ-6, ОИЯИ, 11-9220, Дубна, 1975.
 33. Ю. А. Первин. Основы Фортрана, М., «Наука», 1972.
 34. А. И. Салтыков. Новый вариант экстракода вычисления элементарных функций на БЭСМ-6. В сб. «Развитие аппаратного обеспечения БЭСМ-6». М., ВЦ АН СССР, 1975.
 35. И. Н. Силин. Загрузчик фортраноориентированной системы программирования, использующий «подкачку». Ма-

- терминалы совещания по программированию и матем. методам решения физических задач, ОИЯИ, 11-4655, Дубна, 1969.
36. И. Н. С и л и н. Диспетчер ДД-73 машины БЭСМ-6. В сб. «Совещание по программированию и матем. методам решения физических задач». ОИЯИ, Д10-7707, Дубна, 1974.
 37. И. Н. С и л и н. Операционная система «Дубна» и проблемы создания математического обеспечения мощных ЭВМ. Автореферат докторской диссертации. ОИЯИ, 11-9248, Дубна, 1975.
 38. Н. П. Т р и ф о н о в. SIBESM-6. Методическое пособие, М., ВЦ МГУ, 1967, ротاپринт.
 39. Р. Х и р р, Р. Ш т р о б е л ь. Алгол в мониторной системе «Дубна», перевод с нем., М., ИПМ АН СССР, 1973.
 40. Е. Л. Ю щ е н к о и др. Фортран. Под редакцией Е. Л. Ю щ е н к о. Киев, «Вища школа», 1976.
 41. Язык Фортран. Под ред. В. П. Ширикова, ОИЯИ, 11-4818, Дубна, 1969,

*Альберт Иванович Салтыков,
Григорий Иванович Макаренко*

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ФОРТРАН

(Серия: «Библиотечка программиста»)

М., 1976 г., 256 стр. с илл.

Редактор *Г. Я. Пирогова*

Технический редактор *Е. В. Морозова*

Корректоры *Г. В. Подвольская,*

М. Л. Медведская

Сдано в набор 22/IV 1976 г. Подпи-
сано к печати 4/VIII 1976 г. Бумага
84 × 108¹/₃₂ тип. № 2. Физ. печ. л. 8.
Условн. печ. л. 13.44. Уч.-изд. л. 12.8.
Т. 50 000 экз. (II-й з. 50 001—100 000). Т-151115,
Цена книги 77 коп. Заказ № 1192.

Издательство «Наука»

Главная редакция

физико-математической литературы

117071, Москва, В-71,

Ленинский проспект, 15

Отпечатано в ордена Трудового Крас-
ного Знамени Ленинградской типогра-
фии № 2 имени Евгении Соколовой
Союзполиграфпрома при Государствен-
ном комитете Совета Министров СССР
по делам издательств, полиграфии и
книжной торговли. 198052, Ленинград,
Л-52, Измайловский проспект, 29
с матриц 1-й типографии
изд-ва «Наука». 199034, Ленинград,
В-34, 9 линия, д. 12

77 коп.

