ГЛАВА V

АВТОМАТИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

16. РАЗВИТИЕ МЕТОДОВ АВТОМАТИЧЕСКОГО ПРОГРАММИРОВАНИЯ

1. Общий обзор методов

Непосредственное программирование представляет собой достаточно сложную и весьма трудоемкую работу, требующую большого внимания и аккуратности.

Одна ошибка, допущенная в написании какого-либо числа, особенно команды, может привести в негодность всю программу, а для обнаружения и исправления ошибки требуется затратить много труда и машинного времени. Хотя перед вводом в машину программы многократно проверяются, но обнаружить все ошибки удается не всегда и поэтому в практике эксплуатации больших машин становится неизбежной отладка программ на машинах, т. е. проверка и исправление программ путем анализа пробных машинных решений.

Исправление обнаруженных ошибок в программах требует во многих случаях переделки программы. Например, при пропуске каких-либо команд возникает необходимость вставить участок программы, для чего нужно изменить номера ячеек всех последующих команд и адреса в командах. Иногда этого можно избежать за счет усложнения структуры программы.

Ясно, что практическое получение проверенных рабочих программ связано с выполнением большой и весьма кропотливой работы и непроизводительным расходованием дорогостоящего машинного времени на отладку программ. Поэтому естественно, что усилия ученых и инженеров, работающих в области машинной математики, направлены на изыскание путей автоматизации программирования.

В настоящей главе мы рассмотрим некоторые методы использования электронных цифровых машин универсального назначения для выполнения работы по составлению программ.

Процесс подготовки любой задачи для решения на машине может быть расчленен на три этапа:

а) выбор численного метода, оценка погрешности, даваемой этим методом, и выбор способа контроля правильности и точности вычислений;

б) составление схемы программы (т. е. расчленение автоматического процесса вычислений на отдельные этапы, определение порядка выполнения этих этапов) и распределение памяти.

в) техническая работа по реализации этой схемы, состоящая в составлении команд.

Последние два этапа, взятые вместе, называются собственно программированием. Если первые два этапа требуют высокой математической квалификации работников, то третий этап, связанный с чрезвычайно кропотливой и трудоемкой работой, может быть выполнен значительно менее квалифицированным работником.

В связи с этим, естественно, возникает потребность в такой методике программирования, которая позволяла бы отделить выполнение первого и второго этапов от третьего, тогда работу, начатую высококвалифицированным математиком, сможет закончить техник, имеющий меньшую квалификацию, или даже сама машина.

Сущность проблемы здесь заключается в формализации второго и третьего этапов работы (программирования), т. е. в составлении системы достаточно простых и однотипных правил, определяющих процесс программирования и пригодных для переложения на «язык» машины. Эта система правил должна позволять осуществлять однозначный переход от некоторой принципиальной схемы программы к реализующей ее последовательности команд.

Первоначально развивалось преимущественно ручное программирование, о котором сложилось мнение, как об искусстве, доступном лишь узкому кругу специалистов. С течением времени были разработаны некоторые приемы и методы, упрощающие и облегчающие эту работу.

Основными усовершенствованиями в области ручного программирования явились:

а) метод библиотечных подпрограмм, детально разработанный практически применявшийся в г. Кэмбридже группой английских специалистов, работающих на машине ЭДСАК, под руководством Уилкса, Д. Уиллера и С. Гилла [8];

б) операторный метод программирования, разработанный в математическом институте АН СССР имени В. А. Стеклова под руководством профессоров А. А. Ляпунова и М. Р. Шура-Бура.

В области машинного программирования первоначально велись работы по созданию специальных программирующих приборов или приставок к электронным цифровым машинам общего назначения; однако это направление не привело к существенным результатам.

Более плодотворным оказалось другое направление, связанное с использованием для составления программ самих электронных цифровых машин, которые в силу их универсальности и широких логических возможностей оказались вполне приспособленными для выполнения подобной логической работы. Последнее направление может быть объединено под общим названием — разработка программирующих программ.

Программирующие программы разрабатываются применительно к конкретным машинам и по своему назначению и принципам построения могут отличаться друг от друга.

Основное различие заключается в степени механизации процесса программирования, которая обеспечивается той или иной программирующей программой.

В настоящее время можно указать, по крайней мере, три различных метода составления программирующих программ и, соответственно, три различных типа таких программ.

1. Метод символических адресов. Программирующие программы, построенные по этому методу, осуществляют автоматическое расписывание с помощью машины действительных адресов в программах на основе символических адресов, указываемых составителем. Этот метод разработан группой сотрудников фирмы ИБМ в США применительно к машине ИБМ-701 [9].

2. Метод сопроводительных величин. Этот метод предусматривает автоматическое составление машиной полных рабочих программ на основании подробной характеристики всего вычислительного процесса, задаваемой в виде последовательности специальных чисел. Этот метод впервые предложен Х. Рутисхаузером в

1952 г. [10].

3. О п е р а т о р н ы й м е т о д машинного программирования, разработанный в Математическом институте АН СССР Э. З. Любимским и С. С. Камыниным под руководством М. Р. Шура-Бура. Этот метод является развитием ручного операторного метода программирования, разработанного там же, и обеспечивает автоматическое составление машиной полной рабочей программы на основании сравнительно краткой исходной информации, вводимой в машину и характеризующей математическую формулировку задачи и метод решения. Повидимому, в настоящее время этот метод является наиболее совершенным.

Рассмотрим кратко перечисленные методы ручного и автоматического программирования. Более подробно остановимся на операторном методе программирования, как имеющем наибольшее практическое значение и перспективы развития. Мы не будем приводить здесь конкретных программ, используемых для разных машин при различных методах программирования, а опишем только основные идеи и принципы построения таких программ. Детальное изучение конкретных программ, вообще говоря, является весьма кропотливым делом и приносит немного пользы, так как программы, составленные для одних машин, неприменимы к другим машинам. Важно знать идею программы, тогда можно построить подобную программу для любой машины.

2. Метод библиотечных подпрограмм

Сущность метода заключается в том, что для наиболее часто встречающихся вычислительных процессов составляются типовые программы, образующие библиотеку подпрограмм. Эти подпрограммы составляются таким образом, чтобы их можно было комбинировать, объединять друг с другом для получения программ более сложных задач.

В общем случае библиотечную подпрограмму можно определить как подпрограмму, удовлетворяющую следующим трем условиям:

а) она должна иметь определенное функциональное назначение, т. е. соответствовать определенной, более или менее законченной совокупности действий над числами, что позволит использовать подрограмму как единое целое, без дробления на части и внесения существенных изменений;

б) подпрограмма должна иметь вход, т. е. должно быть предусмотрено несколько ячеек, обеспечивающих задание необходимой информации для работы подпрограммы;

в) подпрограмма должна иметь выход, т. е. группу ячеек, в которых получаются результаты вычисления по данной подпрограмме и откуда эти результаты передаются для дальнейшего использования.

Применение библиотеки подпрограмм имеет два основных достоинства. Во-первых, наличие библиотеки подпрограмм может во многих случаях значительно облегчить работу по программированию задач. Библиотечные подпрограммы вставляются в общую программу как единое целое, не требуя затраты времени на программирование отдельных операций. Некоторые программы сложных задач могут быть составлены почти полностью из библиотечных подпрограмм.

Во-вторых, применение библиотечных подпрограмм, проверенных путем решения задач на машине, в значительной мере уменьшает возможность появления ошибок в программе и во всяком случае существенно облегчает отыскание ошибок.

Для эффективного использования библиотечных подпрограмм необходимо, чтобы они обладали достаточной гибкостью; должно допускаться внесение некоторых изменений в подпрограммы, возникающих в конкретных задачах, а также объединение подпрограмм между собой и с основной программой.

К числу недостатков метода библиотечных подпрограмм относятся:

1. Ограниченность круга задач, решаемых с использованием библиотечных подпрограмм при данном составе библиотеки. Этот недостаток является особенно существенным при необходимости решать широкий круг разнотипных задач, а также на начальном этапе формирования библиотеки подпрограмм.

2. Некоторое удлинение как самих библиотечных подпрограмм, так и полных программ, получаемых из них, а также увеличение количества тактов работы машины.

3. Строение библиотечных подпрограмм в значительной степени зависит от конкретной конструкции машины, для которой составляются подпрограммы, и поэтому трудно приспособить библиотеку подпрограмм, составленную для одной машины, к другой машине.

4. Объединение библиотечных подпрограмм в сложную программу во многих случаях требует большой работы по согласованию адресов и тщательной проверки переходных участков. Эта работа при ручном программировании может оказаться в отдельных случаях более сложной, чем непосредственное программирование новой задачи. Существенным образом меняется дело при использовании для расписывания адресов в командах самой электронной вычислительной машины, как это имеет место при программировании с символическими адресами.

Перечисленные недостатки метода библиотечных подпрограмм объясняют сравнительно небольшое распространение этого метода при ручном программировании. В то же время применение библиотечных подпрограмм для машинного программирования методом символических адресов, когда автоматизируется процесс присвоения адресов и объединения подпрограмм, может обеспечить значительное сокращение трудоемкости программирования.

Рассмотрим некоторые особенности организации библиотеки подпрограмм.

В к л ю ч е н и е п о д п р о г р а м м в. п р о г р а м м у . При использовании библиотечных подпрограмм полная программа любой задачи строится из двух частей:

а) основной или объединяющей программы, обеспечивающей связь и переходы между отдельными подпрограммами;

б) набора библиотечных подпрограмм, соответствующих отдельным участкам вычислительного процесса.

Могут быть применены два способа объединения подпрограмм в программу. При первом способе подпрограммы ставятся друг за другом в естественной последовательности, так что возрастание номеров ячеек памяти, в которых располагаются команды основной программы и подпрограмм, идет в том порядке, в котором должны выполняться команды. Между отдельными подпрограммами ставятся в случае необходимости связующие участки основной программы. При этом, команды основной программы и переходы между

подпрограммами выполняются автоматически в нужной последовательности за счет естественного порядка выполнения команд программы, принятого в машине. Схематически этот способ соединения подпрограмм показан на рис. V.*1,а.* Второй способ соединения подпрограмм показан на рис.V.1,*б.*

При втором способе включения подпрограмм участки основной программы располагаются в одной части памяти, а все подпрограммы могут располагаться в любых других местах памяти. Связь между отдельными подпрограммами и основной программой осуществляется с помощью команд условного или безусловного перехода.

Второй способ является более гибким, так как позволяет располагать подпрограммы в любых местах памяти и, в частности, обеспечивает возможность расположения некоторых подпрограмм всегда в одних и тех же фиксированных ячейках независимо от того, в какой задаче они участвуют. Последнее обстоятельство является важным с точки зрения упрощения как самих подпрограмм, так и связующих участков основной программы.

Действительно, многие подпрограммы содержат внутри себя циклы вычислений с преобразованиями команд и переходами внутри подпрограмм. Естественно, что такие подпрограммы проще использовать в том случае, когда они располагаются в постоянных ячейках, чем в том случае, когда они для разных задач переносятся из одного места памяти в другое. Для упрощения переходных участков основной программы удобно исходные данные, необходимые для работы подпрограмм, и результаты вычислений по этой подпрограмме помещать также всегда в одних и тех же фиксированных ячейках. Особенно удобно применять второй способ включения

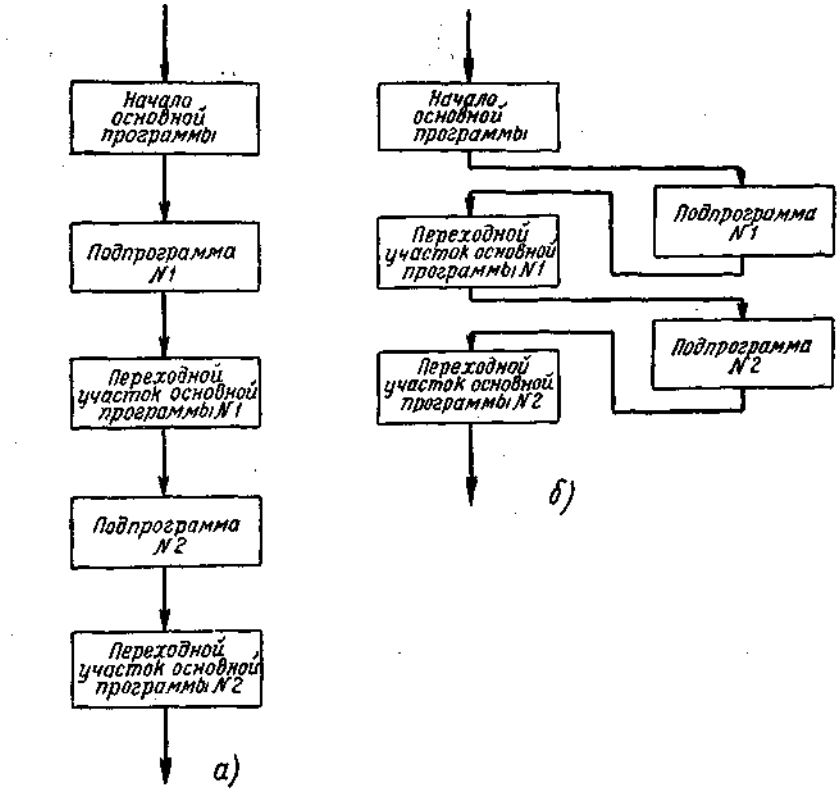


Рис. V.1.

подпрограмм в тех машинах, в которых имеется вторая команда условного перехода, обеспечивающая, помимо выбора очередной команды по заданному условию, также и автоматическое возвращение к прерванному выполнению основной программы после выполнения под программы.

А в т о к о р р е к т и р у ю щ и е с я п о д п р о г р а м м ы

Возможно построение таких подпрограмм, которые, будучи помещены в любое место памяти, при первом выполнении автоматически приспосабливаются к действительным номерам занимаемых ячеек, с соответствующей переадресацией внутри подпрограммы.

При этом на долю связующих участков основной программы остается задание исходных данных для работы подпрограммы и направление в нужные ячейки полученных результатов, если самой подпрограммой не предусмотрена внешняя связь с остальной программой при помощи фиксированных ячеек.

Отправной точкой для автоматической привязки подпрограммы к заданному месту памяти является обычно номер ячейки, назначаемой для помещения первой команды подпрограммы. Предполагается, что все остальные команды подпрограммы размещаются подряд за первой командой в порядке возрастания номеров.

В начале автокорректирующейся подпрограммы должен помещаться дополнительный участок подпрограммы, обеспечивающий коррекцию. Этот участок выполняется только один раз при первом обращении к подпрограмме. Пропуск этого участка при повторных обращениях к подпрограмме может быть осуществлен, например, постановкой перед этим участком следующих двух команд:

$$(\mathrm{K}+1)\ \mathrm{B},\quad a,\quad b,\quad a$$

$$(\mathrm{K}+2)\ \mathrm{E},\ \mathrm{K}+3,\ K+n,\ 0,$$

где (a) = 1,5; (b)=1, $K+n$ — номер ячейки, содержащей команду, к которой необходимо перейти после пропуска данного участка. Коррекция заключается в том, что к адресам команд условного перехода, служащих для осуществления переходов внутри подпрограммы, и к адресам команд, служащих для преобразования команд (C_1,B_2), прибавляется постоянная поправка, равная разности между действительным номером ячейки, в которой помещена первая команда подпрограммы, и некоторым условным номером этой начальной ячейки, на который была рассчитана подпрограмма. Эта поправка должна определяться заранее составителем и задаваться при помощи основной программы.

Помимо рассмотренного случая автоматической корректировки подпрограмм в зависимости от занимаемых ячеек памяти в подпрограммах можно предусмотреть автоматическое изменение других параметров, характеризующих подпрограмму (количество циклов, точность расчетов, коэффициенты и вид расчетных формул и др.). Исходные данные для подобных изменений должны задаваться составителем при помощи основной программы, а для выполнения этих изменений в самих подпрограммах могут быть предусмотрены соответствующие участки.

Естественно, что все эти расширения возможностей подпрограмм связаны с их усложнением и увеличением объема и, облегчая пользование этими подпрограммами, приводят к увеличению времени решения и загрузки памяти. Поэтому во многих случаях целесообразно вместо приспособления одной подпрограммы для разных задач иметь несколько простых подпрограмм одинакового назначения.

Организация библиотеки подпрограмм

Для удобства пользования библиотечными подпрограммами в библиотеке должен быть каталог подпрограмм, в котором подпрограммы располагаются по классам решаемых задач и приводятся краткие сведения, необходимые для удобства пользования подпрограммами. В каталоге указываются следующие сведения: класс подпрограммы (т. е. ее назначение); имеются ли ограничения на номер первой ячейки подпрограммы (т. е. должна ли подпрограмма располагаться в фиксированных ячейках памяти или может располагаться произвольно); характер связей с остальной программой; общее количество ячеек, занимаемых подпрограммой; количество и номера ячеек памяти, необходимых в качестве рабочих ячеек для подпрограммы; если возможно, количество тактов работы машины по данной подпрограмме (иногда это нельзя установить заранее).

Во многих случаях для выполнения одной и той же операции в библиотеке содержится несколько подпрограмм, отличающихся по времени работы и количеству занимаемых ячеек памяти. Как правило, выбирается та подпрограмма, которая требует меньше времени, если это допускается наличным объемом свободной памяти.

Время работы имеет большое значение только при многократном выполнении подпрограмм. Подпрограммы могут отличаться также по точности вычислений, и в каждом конкретном случае выбирается подпрограмма, обеспечивающая требуемую точность.

3. Метод символических адресов

Сущность этого метода заключается в следующем. Программы для автоматических электронных цифровых машин составляются, как и при непосредственном программировании, полностью программистом, только вместо действительных номеров ячеек и адресов в командах ставятся символические номера и адреса.

Автоматическая вычислительная машина при помощи специальной преобразующей программы преобразует символические номера и адреса в действительные и выдает полностью оформленную рабочую программу.

Этот метод имеет следующие достоинства:

а) значительная часть технической работы по оформлению программы (присвоение адресов и номеров ячеек) выполняется не составителем программы, а вычислительной машиной;

б) обеспечиваются широкие возможности для внесения в программы любых изменений, дополнений или исключений как в процессе их составления, так и в готовые программы, применяющиеся для решения задач на машине. При этом изменение программ требует незначительной затраты труда и самое главное может производиться без ошибок;

в) обеспечивается возможность удобно использовать библиотечные подпрограммы, так как почти вся техническая работа по включению подпрограмм в основную программу выполняется самой вычислительной машиной.

Рассмотрим применение метода символических адресов в случае машин с одноадресными командами. Предположим, что непосредственным программированием составлен некоторый участок одноадресной программы с действительными адресами и номерами ячеек, показанный в табл. V.I.

Таблица V.1

Номера ячеек	Команды		Пояснения
	Операция	Адрес	
0161	$П_1$	0521	Перенести число из ячейки 0521 в регистр
0162	С	0522	Сложить число 0522 с числом в регистре (результат остается в регистре)
0163	В	0524	Вычесть из числа, находящегося в регистре, число находящееся в ячейке 0524, (результат остается в регистре)
0164	$Е_1$	0166	Если число в регистре отрицательно, то перейти к выполнению команды 0166, если не отрицательно, то выполнять очередную команду 0165
0165	С	0525	Прибавить к числу в регистре число, находящееся в ячейке 0525
0166	$П_2$	0640	Перенести число из регистра в ячейку 0640

Предположим, что после составления программы выявилась необходимость поставить между командами 0162 и 0163 еще одну команду, например, команду С 0523 (прибавить к содержимому сумматора число, находящееся в ячейке 0523). Для того чтобы вставить эту команду, необходимо сдвинуть все последующие команды, начиная с 0163, вниз на одну ячейку, т. е. увеличить на единицу номера ячеек этих команд, и, кроме того, необходимо изменить адрес в команде 0164, а именно вместо команды E_1 0166 записать E_1 0167. После этого участок программы будет иметь вид, показанный в табл. V.2.

Таблица V.2

0161	$П_1$	0521
0162	С	0522
0163	С	0523
0164	В	0524
0165	E_1	0167
0166	С	0525
0167	$П_2$	0640

В длинных и сложных программах подобные изменения чрезвычайно затруднительны: проверка нескольких сот команд с изменением номера каждой ячейки или адреса и выяснением дальнейших изменений, вытекающих из сделанных изменений, требует большой затраты труда и чревата опасностями внесения новых ошибок. Обычно избегают переделки всей программы за счет некоторого усложнения ее структуры путем постановки «заплат», т. е. введения некоторых дополнительных участков программы, располагаемых в свободных местах памяти и вызываемых в нужные места программы с помощью команд безусловного перехода.

Этот способ приводит к увеличению объема программы, затрудняет проверку и тоже может явиться источником ошибок.

Метод символических адресов в значительной степени устраняет эти затруднения. При программировании по этому методу вместо действительных адресов и номеров ячеек пишутся условные символы — числа, буквы или какие-нибудь другие знаки, которые могут восприниматься машиной. (Заметим, что в машине ИБМ-701 предусмотрена возможность задания не только числовых данных, но и буквенных). Важно только, чтобы различным адресам и номерам' ячеек соответствовали различные символы. При этом не требуется, чтобы символы имели количественный или порядковый смысл (хотя удобно, с точки зрения простоты получения разных символов, пользоваться, например, числами, увеличивая или уменьшая для получения каждого нового символа какое-нибудь исходное число на некоторую величину). Не обязательно, чтобы все адреса и номера ячеек в программе записывались символически. Если есть возможность сразу определить действительный адрес или номер ячейки, то можно ставить действительные адреса, которые войдут в готовую программу. Необходимо, чтобы вид записи действительных адресов был отличен от вида записи символических адресов.

Для машины ИБМ-701 действительные адреса и номера ячеек записываются четырехзначными десятичными числами, а символические адреса и номера ячеек — шестизначными десятичными числами, разделенными точками на три двухзначные группы (например, 06.21.03). При этом две крайние слева цифры используются для обозначения отдельных блоков программы, две средние цифры служат для указания отдельных команд в каждом блоке и две крайние, справа цифры используются для введения изменений в программу.

Приведенный выше участок программы в случае применения символических адресов может иметь вид, представленный табл. V.3.

Таблица V.3

До внесения изменений			После внесения изменений		
11.01.00	$П_1$	15.62.00	11.01.00	$П_1$	15.62.00
11.02.00	С	13.32.00	11.02.00	С	13.32.00
11.03.00	В	14.40.00	11.02.01	С	16.11.00
11.04.00	E_1	11.06.00	11.03.00	В	14.40.00
11.05.00	С	16.35.00	11.04.00	E_1	11.06.00
11.06.00	$П_2$	17.21.00	11.05.00	С	16.35.00
			11.06.00	$П_2$	17.21.00

Таким образом видно, что внесение в программу новой команды совершенно не изменило вида остальных команд, так как номера ячеек и адресов в командах обозначены условными символами, не имеющими конкретного значения.

Важно только, чтобы эти символы были различными для разных адресов и номеров ячеек.

Ясно, что для обозначения символических номеров ячеек памяти, отведенных для размещения команд, исходных данных, констант и промежуточных результатов, должны применяться те же символы, которые используются для обозначения соответствующих символических адресов в командах.

После того, как все команды программы записаны составителем указанным символическим способом, программа перфорируется на перфокарты таким образом, что на каждой карте перфорируется одно число или одна команда. На этой же карте перфорируется и символическое обозначение номера той ячейки памяти, в которой должно находиться это число или команда.

Перфокарты располагаются в колоде в том порядке, в котором команды и исходные данные должны поступать в память, и подготовленная таким образом колода перфокарт вкладывается в устройство для чтения и передачи данных с перфокарт в память. По специальной преобразовательной программе, введенной заранее в машину, производится последовательное чтение поступающих данных преобразуемой программы вместе с их символическими адресами и преобразование символических адресов в действительные. В результате машина выдает, во-первых, рабочую программу на перфокартах и, во-вторых, для ознакомления и контроля — программу, отпечатанную в двух видах с символическими и действительными адресами и номерами ячеек.

Следует заметить, что для подобной работы важным свойством является способность машины воспринимать буквенный текст, закодированный пробивками на перфокартах, и выдавать этот текст в отпечатанном виде, пригодном для обычного чтения.

Поэтому при первоначальной записи символических команд и исходных данных на перфокартах на эти же

перфокарты заносятся и необходимые словесные пояснения, и при выдаче преобразованной программы машина вместе с символическими адресами печатает также и ранее введенные словесные пояснения. Таким образом получается полный контрольный материал, обеспечивающий возможность удобного изучения и проверки программы.

Рабочую программу машина выдает в виде пробивок на перфокартах в двоичной системе счисления в окончательном виде, содержащем только необходимые действительные адреса, без каких-либо пояснений.

Эта рабочая программа вводится в машину перед непосредственным решением задачи.

Сама преобразовательная программа является весьма сложной, ее структура в значительной степени зависит от конкретных особенностей машины, для которой она составлялась. Однако принципы, положенные в основу преобразовательной программы, достаточно просты и могут быть использованы при составлении подобной программы для любой другой электронной счетной машины универсального назначения. Основным в принципе работы преобразовательной программы является следующее. Так как данные поступают в машину с перфокарт в том порядке, в каком они должны быть расположены в памяти, то машина может путем счета количества поступивших данных определить действительные адреса для каждого введенного числа (под числами мы в данном случае подразумеваем и числа, представляющие команды).

При первоначальной записи на перфокартах вместе с каждым числом записывается на одной карте и символический номер ячейки, предназначаемой для хранения этого числа.

Так как в машину каждое число передается вместе с этим символическим номером ячейки, то имеется возможность установить однозначное соответствие между действительными адресами, определенными машиной для каждого числа в процессе ввода, и символическими номерами ячеек, введенными с перфокарт. Каждое введенное число преобразуемой программы записывается в памяти вместе с символическим и действительным номерами ячейки.

Само число в том случае, когда оно представляет команду, может содержать в себе еще символический адрес, который должен быть заменен действительным. Это выполняется на следующем этапе работы преобразовательной программы, причем здесь могут иметь место два различных случая:

1) вся преобразуемая программа вместе со всеми дополнительными данными полностью помещается в память машины;

2) программа и дополнительные данные не помещаются полностью в память и должны вводиться частями.

Первый случай является более простым. При этом машина для замены в командах символических адресов действительными просматривает всю программу и отыскивает те команды, у которых имеются только символические адреса без действительных (для некоторых команд, как упоминалось выше, могут быть записаны сразу действительные адреса). После того как такая команда обнаружена, машина запоминает символический адрес этой команды и отыскивает, просматривая символические номера* ячеек, соответствующий действительный адрес, который и заносится в соответствующую ячейку для этой команды.

Таким образом просматриваются все команды программы и все символические адреса заменяются действительными. Полученная программа печатается в двух видах: с символическими и действительными адресами и необходимыми пояснениями. Кроме того, машина выдает готовую рабочую программу, т. е. действительные номера ячеек и соответствующие числа, помещаемые в этих ячейках на перфокартах в двоичной системе.

Во втором случае возникает трудность, связанная с тем, что преобразуемая программа вводится в память частями и отдельные команды могут иметь в качестве символических адресов адреса таких чисел, которые еще не введены в память.

В этом случае часть памяти отводится для временного запоминания таких неопределенных адресов. Разделение преобразуемой программы на части для ввода в машину производится с таким расчетом, чтобы обеспечить минимум переходов между частями. Весь массив перфокарт разделяется специальными разделительными перфокартами; считывание данных с перфокарт производится от одной разделительной перфокарты до другой. После этого машина производит в описанном выше порядке для каждой введенной части программы поиски действительных адресов для каждого символического адреса. Если для какого-либо символического адреса не находится действительный адрес, то машина записывает этот символический адрес в группу неопределенных адресов (если там еще не был записан такой адрес, для чего перед записью каждого нового неопределенного символического адреса машина просматривает все ранее записанные неопределенные символические адреса). Проведя возможные преобразования введенной части программы, машина переписывает все данные, относящиеся к этой части, во внешний накопитель, освобождая тем самым память машины для следующей части программы.

После того как все части программы пройдут преобразование и будут переписаны во внешний накопитель, появляется возможность провести поиски действительных адресов для всех оставшихся неопределенных символических адресов. Это выполняется путем последовательного ввода из внешнего накопителя в память частями параллельно записанных действительных и символических номеров ячеек и сравнения с символическими номерами ячеек оставшихся неопределенных символических адресов. После получения всех действительных адресов машина печатает указанный выше контрольный материал и рабочую программу.

Метод символических адресов удобно применять при использовании библиотечных подпрограмм или программ ранее решенных задач. При этом библиотеки подпрограмм могут принести значительно больше пользы, чем это имело место до сих пор, так как почти вся техническая работа по включению библиотечных подпрограмм и соединению отдельных программ в новую программу выполняется вычислительной машиной. При включении библиотечных подпрограмм в основную программу имеют место два следующих обстоятельства.

Во-первых, может получиться так, что один и тот же символический адрес имеет различные значения в разных библиотечных подпрограммах. Для устранения неопределенности в этом случае составитель программы должен обеспечить различие между этими символами, что осуществляется путем добавления к символам новых значков.

* Заметим, что между терминами номер и адрес ячейки имеется некоторое различие: номер ячейки является адресом в том случае, когда он записан в составе команды.

Для этого нет необходимости переписывать подпрограммы и изменять вручную все символы. Достаточно указать дополнительные знаки на разделительных перфокартах, разделяющих подпрограммы между собой, после чего необходимые изменения в символы будут внесены машиной автоматически в процессе преобразования.

Во-вторых, может оказаться, что одно и то же число или команда в разных подпрограммах имеет различные символические адреса, т. е. два или больше символических адреса являются синонимами, так как относятся к одному и тому же числу. Составитель программы должен определить, какой из синонимов должен остаться, а какие должны быть отброшены. Синонимы записываются по одной паре на перфокарту. При первом считывании начальных карт машина отбрасывает ненужные синонимы и заменяет их везде одинаковыми значениями.

В заключение следует сказать, что метод символических адресов, так же как и метод библиотечных подпрограмм, целесообразно применять при составлении достаточно сложных программ. При программировании простых задач эти методы не дадут заметной выгоды в отношении сокращения времени программирования по сравнению с непосредственным программированием.

Метод символических адресов может быть с успехом применен и для машин с трехадресной системой команд. Принципы построения и порядок работы преобразовательной программы в этом случае полностью остаются такими же, как в рассмотренном выше случае одноадресных команд. При этом только несколько усложнится сам процесс сравнения символических адресов в командах с символическими номерами ячеек, производимый при поиске действительных адресов, так как нужно производить отдельно сравнение каждого из трех адресов в трехадресной команде.

4. Метод сопроводительных величин

Метод сопроводительных величин, разработанный Х. Рутисхаузером, относится к методам автоматического программирования **с** помощью вычислительной машины.

В отличие от предыдущих двух методов (библиотечных подпрограмм и символических адресов), в которых программы составлялись в основном самими программистами, а машина использовалась для технической работы по расписыванию адресов, в этом методе программы составляются самой машиной на основании мая тематических формул задачи. Математические формулы должны быть предварительно арифметизированы, т. е. представлены в виде; подробной вычислительной схемы, соответствующей выбранному численному методу решения.

Для ввода в машину все величины и знаки, входящие в формулы (скобки, знаки действий, знак равенства, знаки сложных действий $\sqrt{x}, \log x$ и др.), должны быть закодированы воспринимаемыми машиной символами, как правило, числами.

Применительно к одноадресной машине, система команд которой приведена в § 11, кодировка формул производится с помощью табл. V.4.*

Таблица V.4

№№ п/п.	Символ, входящий в формулу	Код		Пояснения
		Цифровой	Буквенный	
1	Открытая скобка . .	010000	$П_1$0000	Команда переноса из ячейки номер 0000 в регистр
2	Закрытая скобка . . .	070000	$П_2$0000	Команда переноса из регистра в нулевую ячейку
3	Результативный знак			
	$\rightrightarrows$	070000	—	—
4	+	020000	С0000	Команды с нулевым адресом, содержащие приказы о выполнении соответствующей операции
5	—	030000	В0000	
6	×	050000	М0000	
7	:	060000	Д0000	
8	Число или буква, обозначающая величину	Номер ячейки, в которой хранится величина. (Этот номер меньше или равен 1000)		

Результативный знак ($\rightrightarrows$) ставится между выражением, содержащим ряд математических действий, и буквой, обозначающей (пока еще неизвестный) результат этих действий.

Порядок действий в кодируемой формуле должен быть вполне определен скобками, причем Скобки следует ставить также и для того, чтобы отметить, что умножение или деление выполняются раньше, чем сложение или вычитание. Например, вместо $a + b \times c$ следует написать $a + (b \times c)$. Характер скобок безразличен: вместо фигурных, квадратных и других скобок можно писать обычные круглые скобки. Вся формула записывается в виде выражения, соединенного результативным знаком с некоторой буквой (не имеющей конкретного числового значения), и заключается в скобки.

Покажем на примере, как выполняется кодировка формулы.

Пусть требуется закодировать

$$\frac{A_1}{A_2 + A_3} - A_1 \cdot A_2 \cdot A_3 \rightrightarrows B.$$

* В таблицу включены не все знаки действий, а лишь необходимые для приводимого дальше примера.

Перепишем эту формулу в виде

$$((A_1:(A_2+A_3))-(A_1\times A_2\times A_3)\Rightarrow B).$$

Для размещения величин A_1 ,A_2,, A_3 отведем ячейки с номерами 201, 202, 203, а для результата B—ячейку с номером 100.

С помощью приведенной выше таблицы наша формула может быть представлена в виде последовательности чисел:

b_0=010000;	b_1=010000;	b_2=000201;	b_3=060000;
b_4=010000;	b_5=000202;	b_6=020000;	b_7=000203;
b_8=070000;	b_9=070000;	b_{10}=030000;	b_{11}=010000;
b_{12}=000201;	b_{13}=050000;	b_{14}=000202;	b_{15}=050000;
b_{16}=000203;	b_{17}=070000;	b_{18}=070000;	b_{19}=000100;
b_{20}=070000.			

Закодированную в виде последовательности чисел формулу вводят в машину, которая при помощи специальной программирующей программы автоматически перерабатывает ее в программу решения задачи.

Осуществляется это следующим образом.

1. Машина вычисляет для каждого b_k „сопроводительную величину" a_k (отсюда название: метод сопроводительных величин) по формулам *

$$a_0=0$$
$$a_k=a_{k-1}+\operatorname{sign}(1500-b_k).$$

Из табл. V.4 видно, что для каждого b_k, обозначающего открытую скобку или оперируемую величину,

$$\operatorname{sign}(1500-b_k)=+1$$

и, следовательно,

$$a_k=a_{k-1}+1.$$

Точно так же для каждого b_k, обозначающего знак действия или закрытую скобку,

$$\operatorname{sign}(1500-b_k)=-1$$

и значит

$$a_k=a_{k-1}-1.$$

Закодированная формула вместе с сопроводительными величинами записывается машиной в ее памяти в следующем порядке:

$$a_0,\ b_0,\ a_1, b_1,\ a_2,\ b_2,\ \ldots$$

2. Сопроводительные величины служат машине для определения порядка действий в формуле.

Вернемся к рассматриваемому примеру. Если в декартовых координатах построить точки (k, a_k) и последовательно соединить их прямыми отрезками, то получится ломаная, приведенная на рис. V.2.

Внизу рисунка для каждого значения k дано соответствующее значение b_k.

Легко видеть, что максимумы ломаной соответствуют величинам, минимумы — знакам операций, скобки же расположены на „склонах" ломаной.

Дальнейшая работа машины состоит в том, что она путем просмотра чисел a_k отыскивает наибольшие равные между собой и смежные максимумы ломаной (если окажется несколько групп таких максимумов, то берется группа, отвечающая меньшим значениям индекса k).

В нашем примере были бы выбраны $a_i = a_5 = 3$ и $a_{i+2} = a_7 = 3$. Дальнейшие операции, выполняемые машиной, мы для простоты проследим только на этом примере.

Складывая между собой $b_{i-1} = b_4$ и $b_i = b_5$, машина получает

$$010000 + 000202 = 010202 = \Pi_1 0202,$$

что представляет команду: „перенести величину из ячейки № 202 в регистр".

---

* Напомним читателю, что $\operatorname{sign} x=\begin{cases}+1, & \text{если } x>0\\ -1, & \text{если } x<0\end{cases}$ при x = 0 эта функция не определена

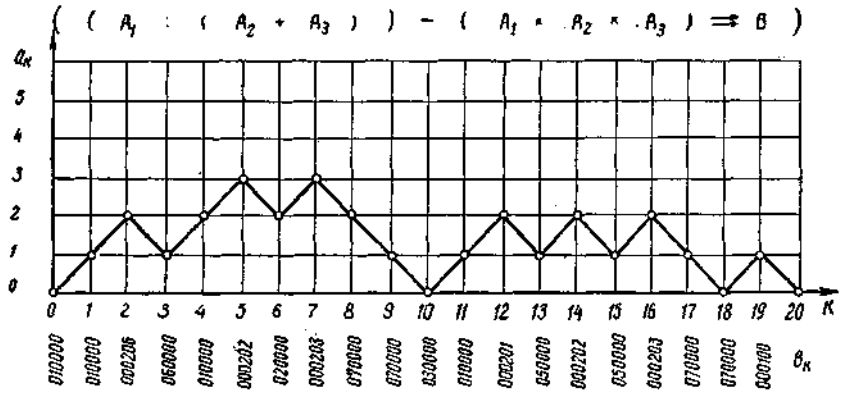


Рис. V.2.

Путем сложения чисел $b_{j+1}=b_6$ и $b_{j+2} = b_7$

$$020000 + 000203 = 020203 = C0203$$

формируется команда: „сложить содержимое ячейки № 203 с числом, состоящим в регистре" (сумма при выполнении такой команды остается в регистре).

Наконец, к величине $b_{i+3} = b_8$ машина прибавляет номер ячейки, предназначенной для хранения промежуточного результата,' например, число 1000 — r, где r — номер полученного результата. В нашем случае r= 1 и, следовательно, к b_8 прибавляется число 999, т. е. 070000 + 999 = 070999 = $П_2$0999, что представляет команду: „перенести содержимое регистра в ячейку 999".

3. После произведенных операций машина стирает числа

$$Ь_{j\_1}., a_j, b_j, a_{/+1}, b_{j+1}, a_{j+2}, b_{j+2}, a_{j+3}, b_{j+3}$$

а в нашем примере

$$b_4, a_5, b_5, a_6, b_6, a_7, b_7, a_8, b_8,$$

вписывая при этом на место $Ь_4$ величину b'_4 =000999 и перенося остальные числа a_k и b_k так, чтобы заполнить образовавшийся пробел в последовательности.

При этом нумерация чисел $a_к$ и $b_к$ изменится и вместо ломаной, изображенной на рис. V.2, можно построить новую ломаную, приведенную на рис. V.3, где $A_4=A_2+A_3$.

Теперь машина опять повторяет описанный процесс и так до тех пор, пока вся закодированная формула не будет переработана в некоторую систему команд. Последние, в порядке их составления, записываются в специально отведенные ячейки памяти. Полученная таким образом программа решения задачи может быть выведена из машины и в дальнейшем многократно использована.

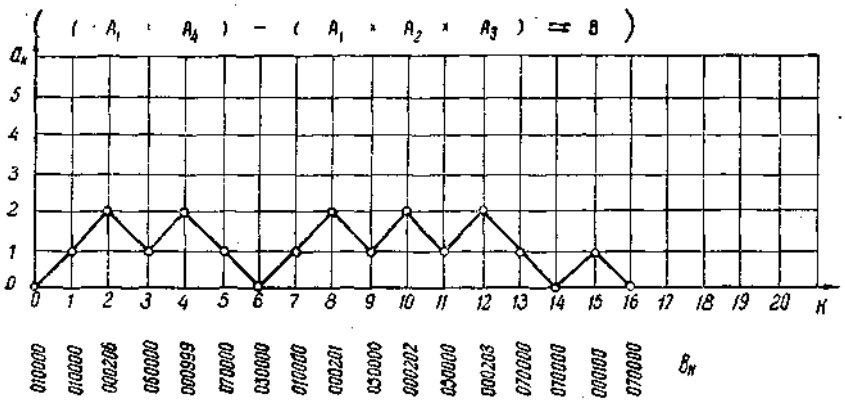


Рис. V.3.

Мы рассмотрели простейший частный случай метода сопроводительных величин, когда машина вырабатывает программу для развернутого вычислительного процесса. Этот же метод может быть применен и для построения программ сложных циклических вычислительных процессов. В циклических вычислительных процессах действия по данной формуле выполняются не один раз, а многократно, но с различными численными значениями входящих в формулу букв.

Для автоматического программирования циклического вычислительного процесса уже недостаточно сопроводительных величин a_k. В машину вместе с кодом формулы должны быть введены еще новые сопроводительные величины, с помощью которых машина будет формировать команды, нужные для построения циклов вырабатываемой программы.

Не приводя здесь дальнейших деталей метода сопроводительных величин, отметим, что применение этого метода для построения циклических программ не является целесообразным, так как сама предварительная работа по подготовке сопроводительных величин, нужных машине для построения циклов, достаточно сложна

и займет, повидимому, не меньше времени, чем непосредственное программирование вручную.

Кроме того, метод сопроводительных величин не является вполне универсальным и требует для каждой новой группы задач, отличающейся от прежних характером цикличности, разработки новых приемов построения циклов, а значит и составления (с последующей отладкой на машине) новой программирующей программы.

Эти серьезные недостатки не присущи операторному методу автоматического программирования, с появлением которого метод сопроводительных величин в том виде, в каком он нам известен, потерял практическое значение.

17. ОПЕРАТОРНЫЙ МЕТОД*

1. Операторы, схемы счета, схемы программ

При операторном методе программирования математическая задача, представленная в виде полной вычислительной схемы, расчленяется на отдельные более или менее самостоятельные этапы, называемые операторами.

Операторы различаются по характеру выполняемых ими действий, по их функциональному назначению, а операторы одинакового функционального назначения — по своему содержанию, определяющему их положение в вычислительной схеме.

При ручном счете участвуют операторы двух видов функционального назначения — так называемые арифметические операторы, предназначенные для выполнения расчетов по формулам, и логические операторы, определяющие порядок применения формул. При автоматических вычислениях появляются еще некоторые другие операторы, которые будут рассматриваться ниже.

Арифметические операторы принято обозначать большой буквой A с номерами (A_1, A_2, A_3, A_4...).

Например, при численном интегрировании обыкновенных дифференциальных уравнений по методу Адамса необходимы следующие арифметические операторы:

а) оператор подготовки начальных данных A_1;

б) оператор, вычисляющий ординаты первых трех точек интегральной кривой, A_2;

в) оператор, вычисляющий ординату очередной *i-ой* точки интегральной кривой по уже известным ординатам предыдущих трёх ее точек, A_3; этот оператор зависит от текущего номера точки i (так называемого параметра). Так как при переходе к расчету ординаты новой точки числа, участвующие в вычислениях, изменяются, поэтому символ оператора удобно снабдить индексом i и вместо A_3 писать A^i_3.

Для решения задачи оператор A^i_3 должен быть применен n—3 раза (здесь n—количество рассчитываемых точек интегральной кривой, занумерованных числами 0, 1, 2... n—1).

Необходимость повторения оператора иногда обозначается буквой П

$$\prod_{i=3}^{n-1} A_3^i .$$

Последовательность арифметических и логических операторов, составляющих полный вычислительный процесс, называется схемой счета задачи. Например, схема счета для получения методом Адамса одной интегральной кривой будет иметь вид

$$A_1 A_2 \prod_{i=3}^{n-1} A_3^i .$$

Если требуется рассчитать не одну, а m интегральных кривых, то схема счета будет иметь вид

$$\prod_{j=1}^{m} (A_1^j A_2^j \prod_{i=3}^{n-1} A_3^{(ij)}).$$

Такое изображение вычислительного процесса показывает, что комплексы операций A_1, A_2, A_3 закономерно повторяются при изменении индексов i и j. Знаки П заменяют собой логические операторы.

В процессе вычислении, помимо выполнения тех или иных арифметических действий по заданным формулам, иногда необходимо изменять ход вычислений, т. е. переходить от одних формул к другим, изменять числовое значение констант, параметров и т. д. в зависимости от некоторых условий, например, в зависимости от значений получающихся промежуточных результатов. Для этого не обходимо в определенные моменты вычислительного процесса осуществлять сравнение некоторых чисел или, как принято говорить, производить проверку логических условий. Такие операторы уже нельзя записать с помощью знаков П.

Поэтому в схему счета задачи, помимо арифметических операторов, входят и операторы проверки логических условий, называемые более кратко просто логическими условиями. Они обозначаются обычно в виде $p(x = y)$ или $p(x \geq y)$ и т. п., причем в скобках указываются сравниваемые величины и характер проверки.

Логическое условие может быть либо выполнено, либо не выполнено.

При написании схем счета принимают за правило, что если проверка логического условия дает положительный ответ, то выполняется следующий по порядку оператор, если — отрицательный ответ, то некоторый другой оператор, указываемый в схеме счета стрелкой, направленной от логического условия.

Например, схема счета может иметь вид

$$\prod_{i=1}^{n} A_1 p_2(x_i \geqslant a) A_3 A_4,$$

* В данном параграфе использованы материалы лекций проф. А. А. Ляпунова

где A_1 — оператор вычислений по формуле $x_t = f_1(x_{i-1}, y_{i-1})$; A_3 — оператор вычислений по *формуле* $y_i = f_2(x_{i-1}, y_{i-1})$; A_4 — оператор вычислений по формуле $y_i = f_3(x_{i-1}, y_{i-1})$.

В данном случае после выполнения арифметического оператора A_1 проверяется логическое условие $p_2(x_i \geq a)$. Если величина x_i больше или равна некоторой константе *a,* то следующим выполняется оператор A_3, если x_i меньше *a,* то оператор A_3 пропускается и выполняется оператор A_4. Буква П указывает, что весь процесс повторяется *n* раз.

Естественно, что расчленение любой математической задачи на отдельные операторы не является однозначным.

Схема счета является формальной записью последовательности операторов, в результате действия которых будет решена данная задача. Для любой задачи схема счета может быть одна и та же, независимо от того, какими конкретными средствами будут осуществляться вычисления: будет ли их выполнять человек карандашом на бумаге или автоматическая цифровая электронная машина. Схема счета характеризует собой только непосредственный вычислительный процесс, но не дает никаких указаний относительно порядка управления автоматическим процессом вычислений. Поэтому при составлении программы для универсальной цифровой машины от схемы счета нужно перейти к схеме программы, представляющей собой формальное описание автоматического процесса решения задачи на машине и являющейся исходным материалом для ручного или машинного составления программы.

Арифметические и логические операторы, фигурирующие в схеме счета, в целом или раздробленном виде (быть может, придется некоторые из них расчленить на более простые) будут присутствовать и в схеме программы. Но помимо них, неизбежно появятся новые операторы (операторы управления и ряд дополнительных логических условий), назначение которых состоит в том, чтобы обеспечить работу арифметических операторов в необходимой последовательности и выполнять необходимые изменения входящих в них команд.

В схеме программы каждый оператор изображает некоторую совокупность команд будущей программы. С одной стороны, желательно, чтобы он охватывал возможно большее число команд. С другой стороны, удобство программирования требует, чтобы эта совокупность команд удовлетворяла определенным требованиям простоты. Приведенные соображения убеждают нас в том, что в дальнейшем целесообразно в понятие оператора вкладывать более узкое и формальное содержание, чем до сих пор.

Элементарным оператором мы будем называть совокупность команд программы, удовлетворяющую следующим условиям:

1) только первая команда оператора может получать управление извне (т. е. от других операторов);
2) внутри оператора управление от одной команды к другой передается только в той последовательности, в какой эти команды расположены;
3) только последняя команда оператора может передать управление другому оператору;
4) произвольный набор команд, удовлетворяющий четырем предыдущим условиям, еще не является оператором. Оператор должен иметь определенный смысл, определенное и притом единственное функциональное назначение в программе.

По своему функциональному назначению элементарные операторы подразделяются на арифметические операторы, операторы управления и логические условия. В свою очередь, операторы управления делятся на несколько типов: переадресации, восстановления и другие.

Операторы переадресации предназначаются для преобразования команд и обозначаются буквой *F*.

Обычно производится переадресация тех команд, адреса которых зависят от какого-либо параметра. Если переадресация связана с изменением значения, например, индекса /, то чтобы это показать пишут *F(j)*.

Переадресация, обусловленная изменением нескольких параметров (индексов), выполняется таким количеством операторов, каково число этих параметров.

Оператор восстановления обозначается О и служит для восстановления команд и параметров, измененных в процессе работы.

Логические условия в общем случае обозначаются буквой *P* и состоят из команды условного перехода и из предшествующей команды, при выполнении которой вырабатывается критерий, определяющий тот или другой переход.

Конкретные логические условия в схеме программы обозначаются при помощи греческих букв μ, ν, λ. Например, условие „если a<b" обозначается буквой μ; условие „если $a = b''$ обозначается буквой ν. Часто логические условия не являются постоянными, а зависят от переменных параметров *i, j, k.*

Эта зависимость изображается при помощи индексов ($\mu^{(ij)}$, ν^k, λ_j,... и т. д.).

Группу элементарных операторов, обладающих тем свойством, что лишь один оператор этой группы может получать управление извне (т. е. при пуске машины или от операторов, не входящих в эту группу), называют обобщенным оператором.

Особо отметим обобщенный оператор, состоящий из нескольких логических условий. В дальнейшем отдельное логическое условие или такой обобщенный оператор мы будем называть логическим оператором. Логический оператор может проверять не только отдельное логическое условие, но и сложную логическую функцию, представляющую из себя ряд логических условий, соединенных знаками логических связей. При написании схем программ нужно стараться, чтобы операторы счета, операторы управления и логические условия располагались подряд, слева направо, по возможности в том порядке, в котором они будут выполняться во время вычислений. Для указания возможных переходов между отдельными операторами с пропусками других операторов, стоящих в схеме программы, применяются дополнительно стрелки; причем переходы вперед указываются стрелками, идущими слева направо снизу формулы, а переходы назад — стрелками, расположенными сверху и направленными справа налево. Все операторы снабжаются номерами в порядке их расположения в схеме. Например, схема программы может выглядеть, как показано на рис. V.4.

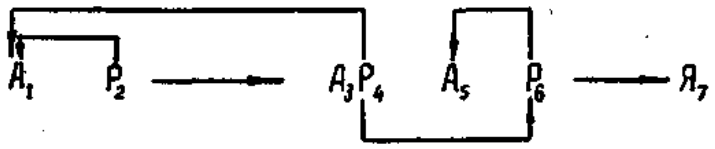


Рис. V.4.

Вместо стрелок в последнее время стали входить в обиход другие значки, более удобные для начертания и, главное, позволяющие ввести своего рода «алгебру операторов»[†]. Повидимому, дальнейшее развитие этой «алгебры» даст способы формально преобразовывать схему программы, что может оказаться полезным при поисках оптимального варианта программы.

Поскольку методика формальных преобразований схемы программы находится еще в стадии начальной разработки и пока только намечается, мы ограничимся лишь самым кратким изложением правил записи схемы программы с помощью новых значков.

Правила записи схемы программы:

1. Схема программы представляет собой последовательность операторов, пронумерованных в порядке их записи и соединенных знаками, показывающими порядок передачи управления от одного оператора другим.

2. Оператор может передавать управление либо одному оператору, либо одному из двух операторов.

3. После каждого оператора K_k ставится знак $\overset{i}{\underset{j}{\square}}$, означающий возможность передачи управления по двум направлениям. Номера, стоящие сверху и снизу, являются номерами тех операторов, которым передается управление.

4. Если от оператора K_k управление получает лишь один оператор L_i, то перед L_i ставится знак $\overset{k}{\underset{k}{\square}}$. Стоящие сверху и снизу номера означают, что оператор L_i получает управление от оператора K_k.

5 Если от оператора K_k может быть передано управление либо оператору L_i либо оператору M_m, то перед одним из них ставится знак $\underset{k}{\square}$, а перед другим — знак $\overset{k}{\square}$.

Передача управления в одном из двух направлений может происходить только от логического условия. При этом принято знак $\overset{k}{\square}$ ставить перед оператором, получающим управление в случае, когда вопрос, выясняемый логическим условием, получает положительный ответ („да"). Знак же $\underset{k}{\square}$ ставится перед оператором, получающим управление, если ответ на вопрос, решаемый логическим условием — отрицательный („нет").

6. Знаки передачи управления взаимодействуют по следующим правилам:

а) вместо $\overset{k+1}{\underset{k+1}{\square}}\ \underset{k}{\square}$ не пишется никакого знака (эти два знака взаимно уничтожаются), поэтому, например,

вместо $A_1 \overset{2}{\underset{2}{\square}}\ \overset{1}{\underset{1}{\square}} A_2$ пишут A_1A_2; вместо $A_1 \overset{2}{\underset{2}{\square}}\ \overset{1}{\underset{1}{\square}} F_2(i)$ пишут просто $A_1F_2(i)$ и т. п.;

б) вместо $\overset{i}{\underset{k+1}{\square}}\ \underset{k}{\square}$ пишется знак $\overset{i}{\square}$. Например, вместо $P_3 \overset{8}{\underset{4}{\square}}\ \overset{}{\underset{3}{\square}} A_4$ следует писать $P_3 \overset{8}{\square} A_4$;

в) вместо $\overset{k+1}{\underset{j}{\square}}\ \overset{k}{\square}$ пишется знак $\underset{j}{\square}$. Например, вместо $P_5 \overset{6}{\underset{10}{\square}}\ \overset{5}{\square} A_6$ следует писать $P_5 \underset{10}{\square} A_6$. Поясним сформулированные правила несколькими примерами.

Запись $A_1 \overset{}{\underset{}{\square}} A_2$ означает, что оператор A_1 передает управление оператору A_2 (и только ему).

Запись $P_5 \overset{}{\square} A_6$ означает, что оператор P_5 передает управление либо оператору A_6 (если вопрос, выясняемый логическим условием P_5, получает положительный ответ), либо оператору номер 10, в нашей записи не

[†] Идея нового способа обозначения принадлежит Ю.И.Янову.

показанному.

Запись $P_7 \left\lfloor {}^{12} A_8$ означает, что P_7 передает управление либо оператору A_8 (в случае отрицательного ответа на вопрос, выясняемый логическим условием), либо оператору номер 12.

Запись $\left\lfloor {}^{2}_{2} A_1 A_2 \right\rfloor^{1}_{1}$ означает, что A_2 получает управление от оператора А, (и только от него) и, в свою очередь, передает управление оператору Ai (и только ему).

Запись $P_1 \left\lfloor {}^{}_{3} \left\lfloor {}^{}_{3} A_2 \right\rfloor P_3 \left\lfloor {}^{}_{2} A_4 \right.$ означает, что P_1 передает управление либо оператору A_2 (в случае „да"), либо оператору P_3.Оператор A_2 получает управление либо от P_1, либо от P_3 и передает управление только оператору P_3. Оператор P_3 получает управление либо от P_1, либо от A_2 и передает управление либо A_4, либо A_2. В заключение приведем запись с помощью новых значков той самой схемы программы, которая была приведена выше как пример на использование стрелок.

$$\left\lfloor {}^{}_{4} \left\lfloor {}^{}_{2} A_1 P_2 \right\rfloor A_3 P_4 \left\lfloor {}^{6}_{1} \left\lfloor {}^{}_{6} A_5 \right\rfloor^{4} P_6 \right\rfloor_{5} Я_7$$

2. Операторный метод при ручном программировании

Рассмотрим применение операторного метода при ручном программировании. В качестве примера возьмем уже нам известную задачу о перемножении двух квадратных матриц. Элементы матрицы произведения С двух квадратных матриц А и В определяются по формуле

$$c_{ik} = \sum_{j=1}^{n} a_{ij} \cdot b_{jk}.$$

Будем предполагать, что в начале счета на местах, отведенных для помещения матрицы С, находятся нули.

Введем арифметический оператор $A_1^{(i,j,k)}$, который должен выполнять следующие действия: извлечение элементов a_{ij}, и b_{ik} из соответствующих ячеек, перемножение этих элементов, прибавление произведения к содержимому некоторой ячейки, в которой будет запоминаться элемент c_{ik}.

Элемент c_{ik} получится в результате n-кратного действия оператора $A_1^{(i,j,k)}$ при изменении параметра j от 1 до n.

Следовательно, схема счета при вычислении c_{ik} имеет вид

$$\prod_{j=1}^{n} A_1^{(i,\ j,\ k)}.$$

Для вычисления всех элементов i-ой строки матрицы произведения нужно повторить n раз всю предыдущую схему счета, изменяя при этом индекс k от 1 до n. Значит, схема счета для получения всех элементов i-ой строки имеет вид

$$\prod_{k=1}^{n} \prod_{j=1}^{n} A_1^{(i,\ j,\ k)}.$$

Для получения всей матрицы С необходимо повторить весь процесс еще n раз при изменении параметра i от 1 до n. Таким образом, полная схема счета при перемножении матриц будет иметь вид

$$\prod_{i=1}^{n} \prod_{j=1}^{n} \prod_{k=1}^{n} A_1^{(i,\ j,\ k)}.$$

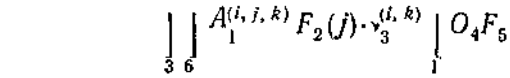
Составим схему программы для решения этой задачи. Повторение n раз оператора $A_1^{(i,j,k)}$ с переадресацией по параметру j будет изображено схемой

$$\left\lfloor {}^{}_{3} A_1^{(i,\ j,\ k)} F_2(j) v_3^{(i,\ k)} \right\rfloor_{1}.$$

Логическое условие $v_3^{(i,k)}$ проверяет, сколько раз повторилась работа операторов $A_1^{(i,j,k)} F_2(j)$. Пока число повторений меньше, чем n, $v_3^{(i,k)}$ передает управление оператору $A_1^{(i,j,k)}$, когда число повторений станет равным n, логическое условие передает управление дальше.

В качестве счетчика повторений может быть использована одна из команд оператора $A_1^{(i,j,k)}$, которая

изменяется с каждым изменением *j*. Так как вид этой команды зависит также и от параметров *i* и *k,* то буква *v* снабжена индексами *i* и *k.* После *n*-кратного повторения оператора $A_1^{(i,j,k)}$ с соответствующими переадресациями необходимо восстановить начальное значение параметра *j* (это делает оператор O_4), увеличить на единицу параметр *k* (это делает $F_5(k)$), т. е. перейти к вычислению следующего элемента строки матрицы С, и повторить *n* раз все предыдущие вычисления для получения всех элементов данной строки матрицы С. Эти действия описываются следующей схемой

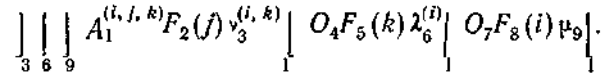


Логическое условие $\lambda_6^{(i)}$ следит за тем, чтобы вся предшествующая ему группа операторов проработала при каждом значении *k,* начиная с *k*= 1 й кончая *k* = *n.* Вид критерия, по которому проверяется это условие, зависит от параметра *i*.

После вычисления всех элементов одной строки необходимо провести подготовку к вычислению первого элемента следующей строки, т. е. восстановить начальное значение параметра *k* и увеличить на единицу параметр *i*. Параметр *j* восстанавливать не требуется, так как он автоматически восстанавливается каждый раз после вычисления очередного элемента, благодаря введенному ранее оператору O_4.

При переходе к вычислению каждой новой строки матрицы необходимо проверять количество вычисленных строк, чтобы определить момент окончания вычислений. Для этого вводится логическое условие μ ; это условие не зависит ни от каких параметров.

Схема программы для умножения двух квадратных матриц будет иметь вид



Эта схема программы составлена в предположении, что все исходные, промежуточные и окончательные данные полностью помещаются в память машины и не требуется в процессе счета обращаться к внешнему накопителю. В схему не включены операторы для контроля правильности вычислений, выдачи данных и других вспомогательных целей.

Рассмотрим, каким образом при помощи этой схемы можно развернуть программу.

Примем следующий порядок размещения данных в памяти.

Элементы матрицы А располагаются по строкам в ячейках с номерами от *a*+1 до $a+n^2$, причем зависимость номеров ячеек от индексов *i* и *j* имеет следующий вид:

$$<a_{ij}> = a + j + n(i - 1). \qquad (V.3)$$

Элементы матрицы В располагаются по столбцам в ячейках с номерами от *b+1* до $b+n^2$*;* зависимость номеров ячеек от индексов *j* и *k* имеет вид

$$<b_{jk}>=b + j + n(k-1). \qquad (V.4)$$

Элементы матрицы С располагаются по строкам в ячейках с номерами от *c*+1 до $c+n^2$; зависимость номеров ячеек от индексов *j* и *k* имеет вид

$$<c_{ik}> = c + k + n\,(i - 1). \qquad (V.5)$$

Вспомогательные величины и условные числа будем вводить по мере необходимости в процессе составления программы, размещая их в ячейках *d, d*+1, d+2,... Ячейка *c* — рабочая.

При программировании операторным методом удобно пользоваться формой бланка программы, приведенной ниже. В этом бланке введены три дополнительные колонки, обозначенные римскими цифрами I, II, III, в которых указывается зависимость 1; 2 и 3 адресов команд от переменных параметров. Таким же образом записываются и константы сравнения (запасные коды команд), зависящие от параметров. Для краткости можно также при программировании вместо специальных чисел, используемых для преобразования команд, указывать только, на сколько единиц и в каком адресе необходимо произвести изменение. Например, если нужно в какой-либо команде первый адрес увеличить на 1 единицу, а третий — уменьшить на 3 единицы, то это может быть записано в виде 1/1,—3/III.

Программа, составленная по указанной выше схеме, представлена в виде табл. V.5.

Поясним процесс составления программы по операторной схеме.

После того как составлена операторная схема программы, составление самой программы сводится к формальному расписыванию команд, руководствуясь имеющейся схемой. Первым в нашей схеме программы стоит оператор $A_1^{(i,j,k)}$, с которого мы и начинаем расписывание команд программы. Ставим две команды *K* + I и *K* + 2, реализующие этот оператор. В качестве рабочей ячейки выбираем некоторую ячейку *c.* После того как записаны команды оператора $A_1^{(i,j,k)}$ просматриваем все адреса в этих командах с целью выявления адресов, зависящих от параметров *i, j, k.* Такими адресами являются 1 и 2 адреса в команде *K* + 1 и 1 и 3 адреса в команде K+2. На основании принятого расположения материала в памяти [см. формулы (V.3), (V.4), (V.5)] пишем в колонках I, II, III зависимости этих адресов от параметров. После этого, следуя схеме программы, строим оператор переадресации $F_2(j)$; просматривая колонки I, II и III, видим, что от параметра *j* зависят 1 и 2 адреса команды *K* + 1, причем с увеличением *j* на единицу и тот и другой адрес должен увеличиваться на единицу. На основании этого пишем команду *K* + 3, предусматривающую специальное сложение команды *K* + 1 с условным числом *(d),* а в ячейку *d* (раздел дополнительные данные) записываем сокращенно необходимое число (1/I, 1/II). Следующим в схеме программы стоит логическое условие $v_3^{(i,k)}$, которое обеспечивает *n* повторений оператора $A_1^{(i,j,k)}$. Выбираем в качестве контролируемой величины (счетчика повторений) код команды *K*+1, который изменяется на число (*d*) с каждым повторением. Для обеспечения контроля необходимо составить константу сравнения. Определяем вид команды K + 1 после *n* повторений:, M, *a* + *n* + 1, *b* +*n*+ 1,C и записываем его в качестве константы сравнения в ячейку *d+1*. Одновременно проверяем, не зависит ли эта константа сравнения от параметров *i* и *k.* Для этого обращаемся к команде *K*+ 1 и

Таблица V.5

Номер ячеек	Команды: Символ операции	Команды: Адреса 1	2	3	Зависимость адресов от параметров I	II	III	Пояснения
1	2	3	4	5	6	7	8	9
$K+1$	M	$a+1$	$b+1$	c	$ni+j$	$nk+j$	—	Арифметический оператор $A_1^{i,j,\kappa}$
$K+2$	C	$c+1$	c	$c+1$	$ni+k$	—	$ni+k$	
$K+3$	C_1	$K+1$	d	$K+1$	—	—	—	Оператор переадресации $F_2(j)$
$K+4$	Cp	$K+1$	$d+1$	000	—	—	—	Логическое условие $\nu_3^{i,\kappa}$
$K+5$	E	$K+6$	$K+1$	000	—	—	—	
$K+6$	B_1	$K+1$	$d+2$	$K+1$	—	—	—	Оператор восстановления O_4
$K+7$	C_1	$K+2$	$d+3$	$K+2$	—	—	—	Оператор переадресации $F_5(k)$
$K+8$	C_1	$d+1$	$d+4$	$d+1$	—	—	—	
$K+9$	Cp	$K+2$	$d+5$	—	—	—	—	Логическое условие $\lambda^i{}_6$
$K+10$	E	$K+11$	$K+1$	—	—	—	—	
$K+11$	C_1	$K+1$	$d+6$	$K+1$	—	—	—	Оператор восстановления O_7
$K+12$	C_1	$d+1$	$d+6$	$d+1$	—	—	—	Оператор переадресации $F_3(l)$
$K+13$	C_1	$d+5$	$d+7$	$d+5$	—	—	—	
$K+14$	Cp	$d+5$	$d+8$	—	—	—	—	Логическое условие μ_9
$K+15$	E	$K+16$	$K+8$	—	—	—	—	
$K+16$	Остановка							
Дополнительные данные								
d	—	1/I	I/II	—	—	—	—	Число для изменения I и 2 адресов на единицу
$d+1$	M	$a+n+1$	$b+n+1$	c	ni	$n\kappa$	—	Константа сравнения
$d+2$	—	n/I	—	—	—	—	—	Число для изменения 1 адреса на n единиц
$d+3$	—	1/I	—	1/III	—	—	—	Число для изменения 1 и 3 адреса на единицу
$d+4$	—	—	n/II	—	—	—	—	Число для изменения 2 адреса на n единиц

Продолжение

Номер ячеек	Команды: Символ операции	Команды: Адреса 1	2	3	Зависимость адресов от параметров I	II	III	Пояснения
1	2	3	4	5	6	7	8	9
$d+5$	C	$c+n+1$	c	$c+n+1$	ni	—	ni	Константа сравнения
$d+6$	—	n/I	$-n^2$/II	—	—	—	—	Число* для увеличения 1 адреса на n и уменьшения 2 адреса на n^2
$d+7$	—	n/I	—	n/III	—	—	—	Число для изменения I и 3 адресов на n
$d+8$	C	$c+n^2+n+1$	c	$c+n^2+n+1$	—	—	—	Константа сравнения

* Для того чтобы одно число обеспечило одновременно увеличение одного адреса и уменьшение второго адреса команды, необходимо в этом числе группы разрядов, соответствующих второму адресу команды, записать в обратном или в дополнительном коде в зависимости от конструкции сумматора.

по формулам в колонках I и II против этой команды определяем, что в константе сравнения должны изменяться: первый адрес на n единиц при изменении на единицу параметра I и второй адрес на n единиц при

изменении на единицу параметра *k.* Эти зависимости (*ni* и *nk*) адресов константы сравнения от параметров *i* и *k* ставим в колонки I и II против константы сравнения (d+ 1). После этого; записываем команды *K* + 4 и K+ 5, составляющие логическое условие $v_3^{(i,k)}$ и обеспечивающие *n*-кратное повторение группы операторов, заключенных в схеме программы в круглые скобки с показателем *n.* Для повторения управление передается первой команде первого оператора в повторяемой скобке.

Затем, следуя схеме программы, приступаем к формированию операторов O_4 и $F_5(k)$. Необходимо уменьшить параметр *j* на *n* единиц, а параметр *k* увеличить на единицу. Проверяем, какие команды и константы зависят от этих параметров *j* и *k.* В командах *K* + 1 первый адрес зависит от *j* и не зависит от *k,* его нужно, следовательно, уменьшить на *n* единиц. Второй адрес зависит и от *j* и от *k* по формуле *nk* +*j.* Ясно, что при увеличении *k* на единицу и уменьшении *j* на *n* единиц выражение *nk*+*j,* остается без изменения, т. е. второй адрес команды *K*+ 1 операторами O_4 и $F_5(k)$ фактически изменяться не будет.

Команда *K* + 6 уменьшает первый адрес команды *K+1* на *n* единиц путем вычитания числа (*d* + 2).

В команде *K* + 2, согласно I и II колонкам, первый и третий адреса не зависят от *j*, но зависят от *k:* с изменением *k* на единицу они также должны изменяться на единицу. Это обеспечивается командой *K+7* и числом *d* + 3 (1/I, 1/Ш). Просматриваем остальные команды и константы и определяем, что от параметра *k* зависит константа сравнения, находящаяся в ячейке d + 1. С изменением *k* на единицу второй адрес в этой константе должен изменяться на *n* единиц. Для преобразования константы сравнения (d+1) ставим команду *K* + 8 и записываем в ячейку *d*+ 4 соответствующее число (*n*/II).

Таким образом, команда *K*+6 является оператором O_4, а команды *K+7, K+8* — оператором $F_5(k)$. Переходим к построению логического условия $\lambda_6^{(i)}$, которое должно обеспечить *n.*-кратное повторение операторного выражения, заключенного в квадратные скобки, т. е. изменение параметра *k* от 1 до *n.* Выбираем необходимый счетчик повторений — команду *K* + 2, которая изменяется в зависимости от параметра *k,* но не зависит от предыдущего параметра *j*.

Определяем вид этой команды при *k* = *n:* C, *c*+*n*+*1, c, c*+*n*+*1,* что дает необходимую константу сравнения. В ячейку d + 5 записываем найденную константу сравнения и ставим команды *K* + 9 и *K* + 10, реализующие логическое условие $\lambda_6^{(i)}$.

Константа сравнения *(d* + 5) зависит от параметра *i*, так как от этого параметра зависит команда *K+2*; эту зависимость указываем записью против константы *(d* + 5) в колонках I и III величины *ni* . Следующими в схеме программы стоят операторы O_7 и $F_8(i)$. Для построения их просматриваем все команды и константы с целью выявления величин, зависящих от параметров *k* и *i.*

В команде *K+1* первый адрес изменяется на *n* при изменении *i* на единицу, а второй адрес изменяется на *n* при изменении *k* на единицу. Следовательно, при уменьшении *k* на *n* единиц и увеличение *i* на единицу первый адрес команды *K+1* должен увеличиться на *n,* а второй адрес — уменьшиться на n^2 . В ячейку *d*+6 ставим число (*n*/1, —/n^2/II) и записываем команду K+11, осуществляющую требуемое изменение команды K+1и являющуюся оператором O_7.

В команде *K* + 2 адреса не изменяются, так как при увеличении *i* на единицу и уменьшении *k* на *n,* выражение *ni*+*k* не изменяется. Кроме того, от параметров *i u k* зависит константа сравнения (*d* + 1): при увеличении *i* на единицу первый адрес увеличится на *n,* а при уменьшении *k* на *n* второй адрес уменьшится на n^2. Для преобразования константы (d + 1), предусмотренного оператором $F_8(i)$, записываем команду K + 12 и используем число (*d* + 6).

От параметра *i* зависит константа сравнения (d+5); первый и третий адреса изменяются на *n* при изменении *i* на единицу. В ячейку *d*+7 ставим число *n*/1, *n*/III и записываем команду *K*+13, необходимую для изменения константы сравнения (d+5).

Таким образом, оператор $F_8(i)$ реализуется командами *K*+12, K+13.

Логическое условие μ_9 должно обеспечить n-кратное повторение операторного выражения, заключенного в фигурные (внешние) скобки, т. е. изменение параметра *i* от 1 до *n.* В качестве счетчика выбираем константу сравнения (*d* + 5), зависящую от параметра *i*; в качестве константы сравнения для проверки логического условия μ_9 запишем в ячейку *d+8* значение этой величины при *i* = *n:*

$$\text{C}, c + n^2 + n + 1, \ c, c + n^2 + n + 1.$$

Логическое условие μ_9 реализуется командами *K*+14 и K+15. Команда *K*+16 обеспечивает остановку машины после конца счета. На этом заканчивается составление программы.

Мы подробно рассмотрели процесс расписывания команд по схеме программы с тем, чтобы подчеркнуть формальный характер этой работы.

Операторный метод программирования имеет следующие достоинства.

1. Обеспечивается возможность формального расписывания команд по схеме программы работниками невысокой квалификации.

2. Обеспечивается возможность независимого программирования некоторых операторов и возможность сравнительно простого объединения полученных участков программы. Сюда же относится удобство возобновления работы по составлению программ после длительного перерыва, что весьма затруднительно при непосредственном программировании.

3. Обеспечивается возможность удобного чтения и проверки программы лицами, не составлявшими эту программу.

3. Принципы работы операторной программирующей программы

Перейдем к рассмотрению автоматического программирования на машине с помощью программирующей программы, основанной на операторном методе. Программирующие программы по своему строению и объему являются чрезвычайно сложными; они содержат несколько тысяч команд. Мы рассмотрим в общих чертах лишь основные принципы построения и работы такой программы.

Порядок работы программирующей программы напоминает порядок ручной работы при расписывании команд

по заданной схеме программы.

Идея, положенная в основу программирующей программы, построенной на операторном методе, состоит в следующем:

а. Весь вычислительный процесс при составлении схемы программы должен быть расчленен на отдельные участки (операторы)* Программирующая программа строит для этих участков отдельные программы, а затем соединяет их в единое целое;

б. Для построения рабочей программы применяются только операторы следующих видов: арифметические, логические, операторы переадресации, операторы восстановления (путем засылки начального вида команд) и засылки (в стандартные ячейки или из стандартных ячеек). Эти операторы называются стандартными. Так как с помощью только стандартных операторов (их команды будет составлять сама машина) не всегда удается построить удовлетворительную рабочую программу, то в нее можно включать также нестандартные операторы, для которых составление команд должно быть произведено вручную.

в. Для обозначения участвующих в вычислительном процессе величин используются условные числа, одни и те же для всей программируемой задачи.

После объединения участков программы в одно целое, получается программа, составленная в условных числах. В дальнейшем с языка условных чисел программа автоматически переводится на язык действительных адресов.

Программирующая программа состоит из отдельных частей — блоков, которые работают независимо друг от друга. Каждый блок производит обработку (расписывание команд и расположение материала в памяти) поочередно для всех операторов одинакового функционального назначения, а затем специальный блок производит расстановку в памяти отдельных частей программы в соответствии с логической схемой.

В состав программирующей программы входят следующие блоки, предназначенные для обработки соответствующих операторов.

A — арифметический блок;
F — блок переадресации;
O — блок восстановления;
P — блок логических условий.
H — блок обработки нестандартных операторов.
Кроме того, в состав программирующей программы еще входят:
W — блок экономии рабочих ячеек;
R — блок расстановки операторов по их порядковым номерам;
D — блок расписывания действительных адресов в командах программируемой программы.

К категории нестандартных операторов относится всякий оператор, который не может быть обработан другими блоками и требует поэтому ручного программирования. Но в ходе эксплуатации программирующей программы выяснилось, что всякую стандартную подпрограмму или часть программы решенной ранее задачи можно использовать как нестандартный оператор. Для этого она должна быть переложена на условные числа новой задачи.

Подготовка схемы программы для ее переработки программирующей программой состоит в следующем:

1) каждый оператор обозначается двумя числами: одно из них указывает на функциональное назначение оператора, а другое является его порядковым номером в данной конкретной схеме программы;

2) все величины, входящие в арифметические формулы, по которым должны вести счет арифметические операторы, обозначаются условными числами;

3) эти формулы кодируются, причем скобки тоже обозначаются определенными условными числами, а для знаков операций используются постоянные коды (например, коды этих операций). В результате каждая арифметическая формула представляется в виде последовательности условных чисел;

4) нестандартные операторы программируются вручную, причем вместо действительных адресов команд пишутся ранее введенные условные числа. Между прочим, как уже говорилось, нестандартными операторами могут являться стандартные подпрограммы или программы и части программ ранее решенных задач, предварительно переложенные на условные числа решаемой задачи;207составляется таблица зависимости всех условных чисел от параметров;

5) производится распределение ячеек памяти для будущей программы;

6) логические условия записываются в виде логических формул и тоже кодируются;

Выполняя описанные выше действия, их оформляют в виде таблиц информации для машины о каждом операторе.

Таблица информации об операторе «озаглавливается» парой чисел, обозначающей данный оператор.

В таблице информации об арифметическом операторе записывается полученный код формул, по которым ведет счет данный оператор.

В таблице информации об операторах переадресации *F* указывается, какие операторы должны ими переадресовываться и как зависят условные числа от переменных параметров.

В таблицу информации об операторах восстановления O_i записывают номера операторов, изменяемые команды которых должны быть ими восстановлены.

В таблицу информации о нестандартном операторе переписывается составленная вручную программа этого оператора.

В таблице информации о логическом операторе записана закодированная логическая формула и номера операторов, к которым должен быть совершен переход при выполнении или невыполнении сложного логического

* В настоящее время разрабатываются программирующие программы, для которых разделение вычислительного процесса на участки производится по другим признакам. Например, для программирующей программы, разработанной в ИТМ и ВТ АН СССР В. М. Курочкиным и А. П. Ершовым, вычислительный, процесс разлагается на циклы вычислений.

условия, описываемого этой формулой. Кроме того, в отдельной таблице приведено содержание элементарных логических условий, входящих в формулу (например, если нужно сравнить две величины, то записана в условных числах команда об их сравнении).

После того, как схема программы представлена в виде таблиц условных чисел, описывающих подробно каждый оператор, эти таблицы перфорируются на перфокарты.

В память машины вводится для обработки информация о группе однородных операторов и соответствующий блок программирующей программы. Работа начинается с обработки логических условий.

Блок Р программирующей программы на основании информации о логическом операторе (логическая формула и содержание элементарных логических условий) составляет группу команд, реализующих этот оператор. Затем вводится в машину арифметический блок *A,* который по заданной информации строит для каждого арифметического оператора его программу в условных числах.

Информация об арифметических действиях, на основании которой составляются команды, образующие арифметические операторы, должна быть составлена так, чтобы порядок действий в ней был строго обусловлен. Это достигается некоторым увеличением числа скобок в формулах. Например, выражение

$$a+b+c + \mathrm{d}(\mathrm{a}+b)$$

перед кодированием должно быть записано так:

$$a + b + c + [d \cdot (a + b)],$$

чтобы при составлении команд оно не было воспринято арифметическим блоком как

$$(a + b + c + d) \cdot (a + b).$$

Арифметический блок просматривает всю информацию с самого начала и, найдя два условных числа, обозначающих числа или величины, между которыми стоит код операции, вырабатывает соответствующую команду. Условные числа и стоящий между ними код операции он после этого стирает. Стирает также и условные числа, обозначающие открытую и закрытую скобки, если упомянутые выше условные числа и код операции заключены в скобки.

На месте стертой информации арифметический блок записывает условное число, обозначающее результат запрограммированной операции.

Выработанная при этом команда записывается в отведенном месте памяти, после чего блок опять начинает просматривать всю информацию с самого начала.

В отличие от программирующей программы, работающей методом сопроводительных величин, арифметический блок не просто формирует команды из их готовых частей, а действительно вырабатывает рациональную программу. В частности, при составлении каждой новой команды арифметический блок просматривает всю дальнейшую информацию. Если он обнаруживает в ней ту операцию над теми же условными числами, которая только что была запрограммирована, то стирает соответствующий участок информации и вписывает на его место условное число, обозначающее результат этой операции. При этом действия $a + b$ и $b + a$ воспринимаются блоком как тождественные (также $a \cdot b$ и $Ь \cdot a$). Этим исключается лишнее повторение команд в программируемой программе. Например, если арифметический блок должен считать по формуле

$$(a+Ь+ c) — (b + a + c) \cdot (a + b + c + d) \rightrightarrows R,$$

то информация об этой формуле может быть представлена в виде последовательности чисел, эквивалентной следующей записи:

$$a + b + c \text{ -}[(b + a + c) \cdot (a + b + c + d)] \rightrightarrows R.$$

Для большей наглядности (которая машине не нужна) мы перепишем последнее выражение с применением скобок различного вида

$$a + b + c — [(b + a + c) \cdot (a + b + c + d)] \rightrightarrows R.$$

Просматривая эту информацию (слева направо), арифметический блок, прежде всего, составит команду для вычисления суммы $a + b,$ в которой результат операции обозначен некоторым условным числом, например r (составит команду $a+b \rightrightarrows$ r), затем просмотрит всю информацию и всюду заменит $a + b$ и $b + a$ этим условным числом. В результате информация примет вид, эквивалентный такой записи,

$$r + c\text{-}[(r + c) . (r + c + d)] \rightrightarrows R.$$

При вторичном просмотре (уже видоизмененной) информации арифметический блок сформирует новую команду r+c $\rightrightarrows$ p (результат операции он обозначит новым условным числом p) и опять видоизменит всю информацию

$$p\text{-}[p.(p+ d)] \rightrightarrows \mathrm{R}.$$

Таким образом, программируемая программа не будет содержать повторных команд для вычисления одних и тех же величин. Арифметический блок сам определит следующий порядок счета

$$1)\ a+b+ c \rightrightarrows p,$$

$$2)\ p\text{-}[p.(p + d)] \rightrightarrows R.$$

После арифметического блока в машину вводится блок нестандартных операторов Н, работа которого

сводится к переписыванию уже составленных (вручную) программ нестандартных операторов в отведенные для них места памяти.

Затем в память вводятся информация об операторах переадресации F_i и блок переадресации *F,* а также общая таблиц зависимостей величин от параметров. Блок *F* составляет в условных числах необходимые команды переадресации.

Наконец, вводится в память информация для работы блока О, составляющего операторы восстановления O_i.

После того как составлены таким образом отдельные программы в условных числах для всех операторов, в работу пускается специальный блок *W* программирующей программы, который просматривает все полученные программы и осуществляет экономию рабочих ячеек.

Затем блок *R* расстанавливает операторы по их порядковым номерам и приводит программируемую программу, записанную в условных числах, в ее окончательный вид.

Последним вводится в память блок *D,* предназначенный для расписывания вместо условных чисел действительных адресов команд. Одновременно в машину для работы блока *D* задается таблица распределения памяти, в которой отдельным условным числам сопоставлены истинные номера ячеек машинной памяти, в которых должны находиться соответствующие величины.

Распределением действительных адресов ячеек заканчивается процесс составления программы, и полученная рабочая программа выводится из машины на перфокарты и на печать для последующего использования.

Следует отметить, что замена ручного программирования автоматическим не устраняет необходимости участия высококвалифицрованных работников в этом деле. Наоборот, повышаются требования к уровню квалификации специалистов. Хотя ручная работа почти полностью исключается, зато возрастает ответственность за составление схемы программы. Схема должна продумываться более детально. При ручной работе схема программы играла роль некоторого ориентира, указывающего общее направление вычислительного процесса. В процессе работы составитель мог вносить уточнения и изменения в программу и даже корректировать ее схему. При автоматическом программировании составленная математиком схема полностью и окончательно определяет всю программу. Ошибка, допущенная в схеме или таблицах, уже не может быть выявлена и исправлена в процессе дальнейшей работы, как это имеет место при ручном программировании, а будет перенесена в окончательную программу. Такие ошибки могут быть обнаружены в процессе практического решения задач. Это повышает роль и ответственность программистов, составляющих логические схемы программ.

Необходимо отметить, что описанный здесь операторный метод автоматического программирования обладает серьезными преимуществами перед всеми прочими. В частности, он позволяет нескольким человекам вести параллельную работу над отдельными кусками одной и той же программы, дает широкие возможности внесения изменений и добавлений в составляемую программу. Нельзя также умолчать о наличии широких перспектив в дальнейшем развитии этого направления автоматического программирования.

Описанные нами выше другие методы автоматического программирования — метод библиотечных подпрограмм, метод символических адресов и метод сопроводительных величин — значительно проще в смысле практической реализации, но обладают существенно меньшими возможностями. Они основываются на тех же двух основных идеях, что и универсальная программирующая программа, а именно: на разделении процесса вычислений на отдельные этапы и использовании условных обозначений. Эти методы охватывают лишь отдельные стороны работы отдельных блоков универсальной программирующей программы. В частности, преобразовательная программа в методе символических адресов выполняет по существу ту же работу, что и блок расписывания действительных адресов в универсальной программирующей программе. По методу сопроводительных величин Х. Рутисхаузера может быть построен специальный блок в универсальной программирующей программе для развертывания отдельных циклов вычислений в полные программы. Библиотечные программы могут удобно применяться совместно с универсальной программирующей программой в качестве нестандартных операторов.

В заключение следует сказать, что методы автоматического программирования в настоящее время усиленно разрабатываются в различных странах. Общая тенденция заключается в том, чтобы в максимально возможной степени упростить и сократить предварительную работу по подготовке задач к решению на машинах.

Возможности техники электронных счетных машин позволяют ожидать в ближайшем будущем значительного прогресса в этой области.

Идеальным будет такое положение, когда задачи в машину будут задаваться непосредственно в виде математических уравнений с некоторыми словесными пояснениями, но без предварительной арифметизации (сведения к вычислительной схеме) и численного кодирования. В принципе возможно также использование электронных цифровых машин и для получения аналитических, формульных решений задач в общем буквенном виде.