



Ученые записки

**РАЗРАБОТКА И ВНЕДРЕНИЕ
АВТОМАТИЗИРОВАННЫХ
СИСТЕМ**

I

Министерство высшего и среднего специального образования
Латвийской ССР

Латвийский ордена Трудового Красного Знамени
государственный университет имени Петра Стучки

Вычислительный центр

Ученые записки

Латвийского государственного университета
имени Петра Стучки

том 242

РАЗРАБОТКА И ВНЕДРЕНИЕ АВТОМАТИЗИРОВАННЫХ СИСТЕМ

Выпуск I



Латвийский государственный университет
Рига 1975

Предлагаемый сборник, составленный из работ сотрудников Вычислительного центра ЛГУ, посвящен как теоретическим вопросам проектирования автоматизированных систем, так и практическим вопросам использования ДЭС/ЕС в процессе технического и рабочего проектирования автоматизированных систем.

По мере накопления материалов по данной тематике предполагается издание следующих сборников.

Сборник рассчитан на студентов физико-математического и экономического факультетов, а также на научных работников и программистов вычислительных центров, занимающихся вопросами проектирования автоматизированных систем.

© Латвийский государственный университет, 1975

Р 30502-140у 75
М 812(II)-75

СИСТЕМА ОТЛАДКИ ПРОГРАММ АСОИ-ПЕНС

А.Я.Абеле, М.В.Витыньш

ОБЩЕЕ ОПИСАНИЕ

1. Назначение

Система отладки программ АСОИ ПЕНС - автоматизированная система, предназначенная для автономной отладки матобеспечения АСОИ ПЕНС на ЕС - ЭВМ типа ЕС-1020 в стадии рабочего проектирования.

Комплексная отладка АСОИ ПЕНС, в том числе и матобеспечения, производится согласно разделу КТС рабочего проекта АСОИ ПЕНС.

Система может быть использована и для автономной отладки других больших систем матобеспечения автоматизированных систем управления.

2. Технические средства

Система отладки использует ЭВМ типа ЕС-1020 со следующей конфигурацией.

Тип устройства	Шифр	Адрес
Процессор с памятью 64 КБ	2020	
Пишущая машинка	7070	01F
Ввод карт	6012	00C
Вывод карт	7010	00D
АЦПУ	7030	00E
Накопитель на МЛ	5012	280+287
Накопитель на сменных МЛ	5052	190+194

Кроме этого, используются устройства подготовки данных на перфокартах типа ЕС-9011.

Система отладки использует операционную систему ДОС/ЕС, библиотеки которой расположены в резидентном файле, под который отведен весь дисковый пакет, постоянно установленный на устройстве 5052 с адресом I90.

В библиотеке абсолютных модулей резидентного файла производится каталогизация всех полученных при отладке программ отлаживаемого матобеспечения потребителя. В целях высвобождения большого объема в библиотеке абсолютных модулей для матобеспечения потребителя произведено перепланирование операционного пакета.

Для СУПЕРВИЗОРА исключена возможность мультипрограммирования. Выполнено стандартное назначение логических устройств всем физическим устройствам.

Создана личная библиотека отлаживаемой системы. На отдельном дисковом пакете, постоянно установленном на устройстве 5052 с адресом I91, размещены личные библиотеки объектных и исходных модулей. На этом же пакете расположены рабочие файлы ДОС-а.

Для рабочих файлов отлаживаемой системы выделен отдельный пакет дисков, постоянно установленный на устройстве 5052 с адресом I92. На устройстве 5052 с адресом I93 установлен пакет дисков для личных файлов, требующих долговременного хранения.

На двух устройствах 5012 с адресами 280 и 281 постоянно установлены рабочие магнитные ленты. На остальных устройствах 5012 по требованию программиста, могут быть установлены личные тома МЛ.

3. Входная информация

3.1. Состав и назначение входной информации

На вход системы отладки поступают следующие документы на бланках (рис.3.1):

- 1) программа,
- 2) изменения к программе,
- 3) данные к программе,
- 4) изменения к данным,
- 5) операторы управления,
- 6) схема отладки.

Информация первых пяти видов перфорируется с бланков и в виде перфокарт каталогизируется в библиотеке системы отладки. При этом информация, помеченная на бланках как изменения к программе (данным), автоматически вызывает коррекцию соответствующих компонентов подбиблиотек перфокарт. Поступившая новая схема отладки заменяет старую, а поступившие новые операторы управления заменяют старые.

Источником всей входной информации является программист. Никакой специальной информации о необходимости выполнения той или иной перфорации, трансляции или запуска программы не требуется. Любой входной документ является приказом к перфорации с последующими корректировкой, трансляцией и запуском программы, к которой он относится, при условии, что все компоненты для этого в библиотеке системы отладки имеются.

Таким образом, на программиста возложена полная ответственность по организации и управлению отладкой его программы.

Система отладки выполняет всю техническую работу, оставляя программисту принципиально-логическую.

3.2. Назначение и содержание входных документов

3.2.1. Программа

Система отладки предусматривает отладку программ, написанных только на языках АССЕМБЛЕРА или PL/I. Программы пишутся на специальных, в зависимости от языка программирования, бланках с соблюдением всех требований языка. Система отладки только требует, чтобы поле идентификации (колонки 73÷80) для каждого оператора программы было заполнено, т.к. по колонкам 77÷80 (номер оператора) производится коррекция программы.

Для более удобной идентификации результатов отладки рекомендуется в качестве первого оператора любой программы иметь комментарий вида:

Z Z Z Z Z Z Z Z, ИВАНОВ, ВЦ ЛГУ
где Z Z Z Z Z Z Z Z -идентификатор программы.

За идентификатором следует фамилия программиста и организация, где он работает. В последующих знаках программист может задать любую свою информацию.

Управляющих карт для трансляции программы от программиста не требуется. Программа после ее перфорации автоматически транслируется, и, если получена результирующая программа, катализируется и, если система отладки может ее укомплектовать согласно "схеме отладки", запускается.

3.2.2. Изменения к программе

Операторы изменений пишутся на тех же специальных бланках, на которых написана программа. Месторасположения оператора изменения определяет его поле идентификации. Никаких специальных указаний, кроме "Удалить от до", не требуется. Система отладки после перфорации изменений производит коррекцию программ на перфокартах и их трансляцию. Результирующие программы каталогизируются и, если система отладки может их укомплектовать согласно "схеме отладки", запускается.

3.2.3. Данные и изменения к данным

Под данными система отладки понимает отладочные данные к программе, при помощи которых программист отлаживает свою программу. Поэтому допускаются частые изменения к данным.

Носителем данных для программы может служить магнитная лента, магнитный диск и перфокарты. Система отладки допускает использование всех типов носителей, однако первоначальный ввод данных предусмотрен только с ПК. Для временного хранения данных программа может использовать рабочие МЛ и рабочий пакет МД, для долговременного - личные тома МЛ и личные участки на пакете личного МД. Каждая программа должна сама себе создать необходимые отладочные файлы на МЛ и МД. Если эти файлы небольшие и требуют частых изменений, целесообразно их хранить на ПК, т.к. в таком виде они удобнее корректируются. Для размещения этих файлов на основном носителе (МЛ, МД), программист в своей отлаживаемой программе вставляет ряд операторов, которые перед использованием этих файлов отлаживаемой программой, переписывают их с ПК на МЛ или МД. После отладки програм-

мы или после выделения основных носителей для постоянного хранения файлов, вставленные операторы удаляются из пакета ПК и программа перетранслируется.

Данные можно писать на любых, пригодных для перфорации, бланках. На бланках, помимо идентификатора самих данных, должен иметься идентификатор программы, к которой эти данные относятся. Номера строк данных, если они необходимы, т.е. предусматривается коррекция данных, следует писать с левой стороны бланка вне поля данных.

Изменения к данным пишутся на тех же бланках, что и данные, и также должны иметь двойную идентификацию. Месторасположение строки изменения определяет ее номер. Никаких специальных указаний, кроме "Удалить от до", не требуется.

Система отладки после перфорации данных или перфорации изменений и коррекций данных на ПК. автоматически производит запуск соответствующей программы, если она каталогизирована в библиотеке абсолютных модулей и, если ее можно укомплектовать согласно "Схеме отладки".

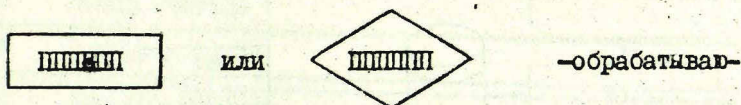
3.2.4. Схема отладки

По каждой программе программист обязан предоставить "схему отладки". На основании схемы отладки оператор-библиотекарь комплектует программу внешними носителями при включении ее в пакет обработки на ЭВМ. Согласно "Схеме отладки" оператор ЭВМ устанавливает внешние носители перед запуском и во время работы программы. Без "Схемы отладки" программа не запускается! Допускается одна схема на несколько последовательно запускаемых программ, при условии, что они всегда будут запускаться в заданной схемой последовательности. Схе-

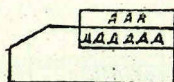
ма идентифицируется по первой или единственной запускаемой программе.

Внизу под схемой, если необходимо, программист может указать, какие действия выполняет оператор ЭВМ на каждое сообщение программы.

При составлении схемы отладки (рис. I) используется следующие обозначения:



шая программа, где IIIIIIIII - идентификатор программы;



^I - пакет данных на перфокартах;

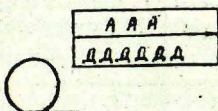
где DDDDDD - номер носителя (для ПК он идентификатор данных).

I, II, III - порядок установки на одном и том же устройстве,

AAA - адрес устройства;



- печатный лист;

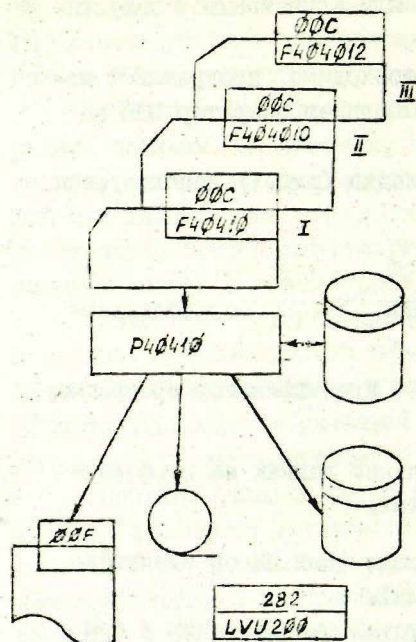


- катушка магнитной ленты;



- пакет магнитных дисков.

СХЕМА ОТЛАДКИ



Имя	F40400	
Тип	I	
Номер	0	
Кол. дор.	10	
Адрес	107	

№ п/п	Сообщение программы	Действие оператора	Ответ оператора
I.	Установить носители	Устанавливает носители согласно этой схеме	"Д"-если устан. "Н"-если не устан.

Рис. I.

К обозначению пакета личных дисков должна быть приложена таблица:

1	Имя файла	
2	Тип участка	
3	Номер участка	
4	Кол-во дорожек	
5	Адрес участка	
6		

Первые четыре реквизита заполняет программист. Адрес участка представляет оператор-библиотекарь. Для выделения внешнего носителя в постоянное пользование (личная МД, личный участок на МД), схема отладки при первом ее поступлении должна быть завизирована руководителем темы.

Поступление новой схемы отладки автоматически аннулирует предыдущую.

3.2.5. Операторы управления

На отдельном бланке, озаглавленном "Операторы управления к программе", программист задает операторы управления, необходимые для работы программы согласно "Схеме отладки". Оператор EXTENT программист задает только для рабочего пакета дисков. Для личного пакета дисков программист указывает в "Схеме отладки" имя файла и количество занимаемых им дорожек.

Заданный операторами управления режим, сохраняется и для последующих запусков программы. Появление нового бланка "Операторы управления" ликвидирует старый.

4. Выходная информация

Вся выходная информация предназначена для программиста. Она состоит из следующих документов:

- 1) бланки отперфорированной информации,
- 2) материалы отладки, полученные с АЦПУ,
- 3) протокол ПМ.

В отдельных случаях программисту возвращается и "Схема отладки". Это свидетельствует о том, что программист или не обеспечил свою программу комплектуемыми данными, или схема составлена непонятно. В таких случаях на "Схеме отладки" сделаны пометки о причине возврата. До поступления исправленной "Схемы", программа запускаться не будет.

5. Внутренняя информация

Всю внутреннюю информацию "Система отладки" по смыслу использования можно подразделить на библиотеку машинных носителей и внутрисистемный учет.

5.1. Библиотека

Библиотека машинных носителей предназначена для хранения программ и данных для их отладки. Исходные тексты на перфокартах хранятся в подбиблиотеке программ (рис.3.3. - II). Программа хранится вместе с операторами управления для трансляции. Это основной вид хранения исходных текстов.

Исходные тексты отлаженных программ, по заявкам программиста, могут каталогизироваться в личной библиотеке исходных модулей системы отладки на магнитном диске. Таким же образом в системную библиотеку абсолютных модулей каталогизируются абсолютные модули программ.

Объектные модули всех, полученных в результате трансляции, программ автоматически каталогизируются в библиотеке объектных модулей.

В целях удаления ненужных программ из библиотеки, производятся периодические инвентаризации, когда путем опроса программистов устанавливаются используемые программы. После этого неиспользуемые программы из библиотеки удаляются.

Данные для отладки хранятся в отдельной подбиблиотеке (рис.3.3. - 33). Удаление программы влечет за собой удаление данных, относящихся к этой программе.

В отдельной подбиблиотеке (рис.3.3. - 66) хранятся перфокарты с операторами отладки. Там имеются заготовленные впрок, постоянные, не меняющиеся от программы к программе, пакеты операторов управления равно как и операторы, укомплектованные по каждой, хранящейся в библиотеке, программе.

Удаление программы влечет за собой удаление ее операторов управления.

В библиотеке хранятся и карты запуско-сервисных программ (разметка ММ, инициализация МД, перезапись и др.), необходимые для обслуживания системы отладки.

5.2. Учетная информация

Учетная информация дает операторам, управляющим работой "Системы отладки", представление о состоянии каждой программы, а также о состоянии всей системы в целом.

Имеются два каталога: каталог операторов управления (рис.3.3. - 60) и каталог схем отладки (рис.3.2. - 50). В них по каждой программе имеется соответственно бланк с операторами управления и схема отладки. При поступлении нового бланка с операторами управления или схемы отладки старые изымаются.

Для внутреннего учета в системе отладки используется карточная система учета.

На каждую учетную единицу (программу) заводится карточка учета (рис.2).

В ней указываются:

- идентификатор программы,
- порядковый номер карточки к данной программе,
- дата перфорации программы,
- номер задачи, к которой программа относится (необязательно).

В карточке учета имеются специальные графы для пометок операторов в процессе отладки. В графе "Перфорация" имеются строки: ОУ, Дан., и Прог., в которых учи-

— влево косая черта,



что 1) информация отперфорирована и изменения внесены;

2) получены перфокарты для каталогизации модуля,

3) по одному из двух заказов произведена перфорация и внесение изменений;



— вертикальная черта,



что 1) в результате трансляции получена (вертикальная черта темного цвета) или не получена (вертикальная черта красного цвета) программа,

2) произведена каталогизация,

3) производился запуск программы (вертикальная черта темного цвета).



Карточки учета хранятся в специальном каталоге карточек учета (рис.3.2. - 13) - ящике с отделениями, помеченными: библиотека, на обработку, перфорация, машина. Нахождение карточки в одном из этих отделений означает, что программа находится в библиотеке без движения, что программа должна быть направлена на обработку, что к программе производится перфорация, что программа в пакете обработки находится на машине.

Технологическая схема системы отладки АСОИ ПЕНС

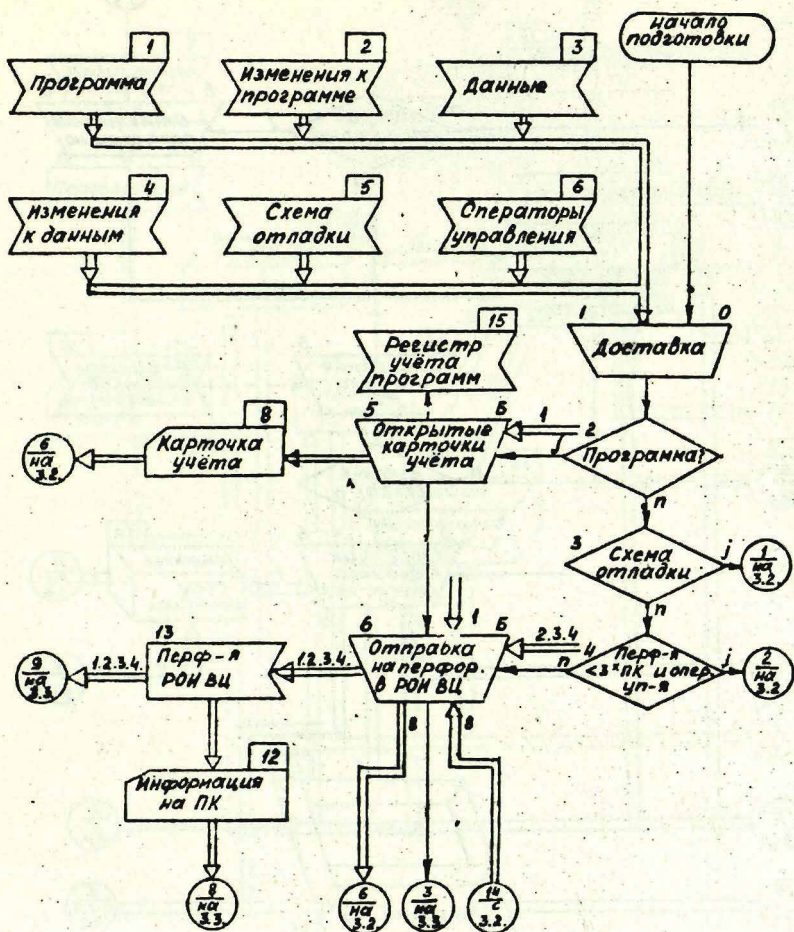
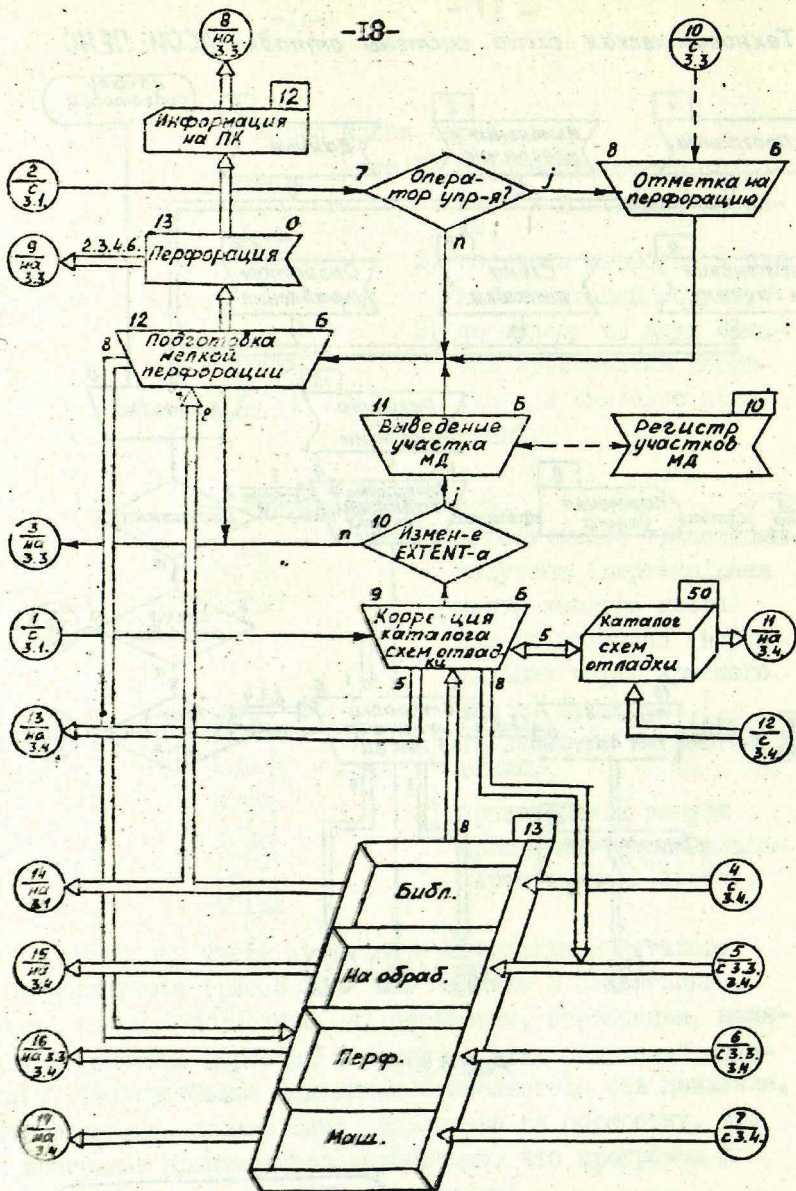


Рис. 3.1.



Puc.3.2.

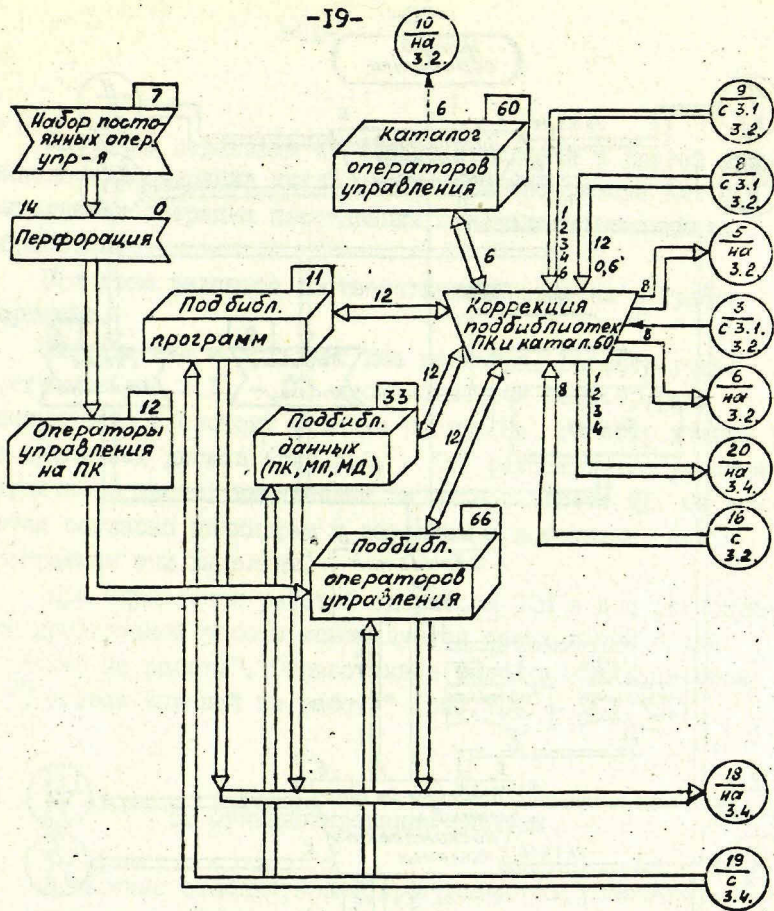
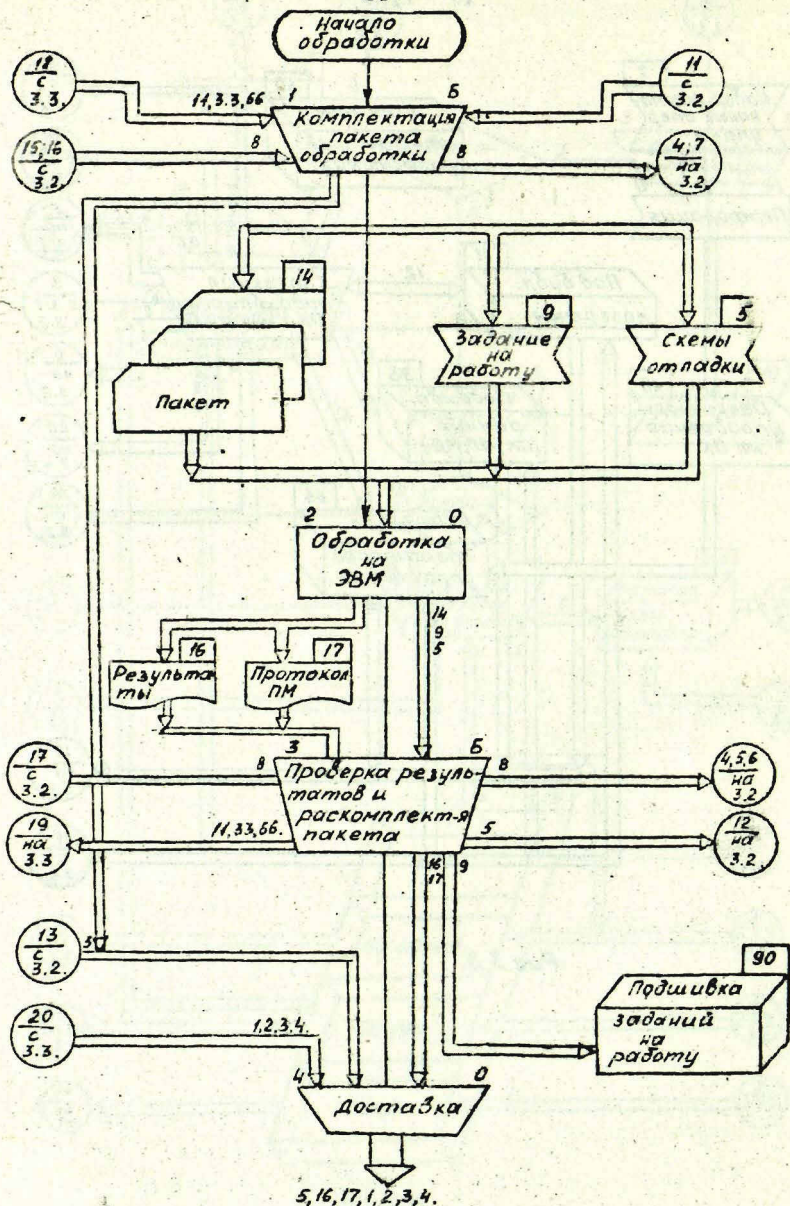


Рис.3.3.



Карточка переходит из одного отделения в другой при выполнении операции цикла подготовки информации или при выполнении операции пакетизации и депакетизации цикла обработки.

При этом делаются соответствующие отметки в графах карточки.

Система еще использует два регистра. Регистр учета программ (рис.3.1. - I5) хронологически регистрирует поступающие в систему отладки программы. Регистр участка магнитных дисков (рис.3.2. - I0) регистрирует участки магнитного диска, выделенные на личном пакете МД. Он ведется согласно цилиндрам и дорожкам с пометками, каким программам они выделены.

При управлении работой оператора ЭВМ и для регистрации проделанной работы используется специальный бланк: "Задание на работу". Отработанные задания накапливаются в "Подшивке заданий на работу" (рис.3.4. - 90).

6. Функционирование системы

В системе отладки, кроме программистов, участвуют I-2 оператора-библиотекаря и необходимое количество операторов ЭВМ. Подразумевается доступным отдел подготовки данных, на который возложены все перфорационные работы, кроме перфорации одиночных карт изменений и операторов управления.

Отладка программ состоит из трех, связанных только расписанием дня и графиком машинного времени, циклов. Цикл работы программиста с программой здесь рассматриваться не будет. Далее следуют описания цикла подготовки информации и цикла обработки на ЭВМ.

Описание циклов ведется со ссылками на "Технологическую схему системы отладки АСОИ ПЕНС" (рис.3.) с указанием номеров операции и носителей. При этом нумерация носителей выполнена справа над обозначением, а нумерация операций (действий) - слева над обозначением. Справа над обозначением проставлены ответственные исполнители: Б - библиотечарь, О - оператор.

6.1. Подготовка информации

Цикл подготовки начинается операцией сбора и доставки (I) бланков входной информации (носители 1,2,3,4, 5,6). После этого по каждому входному документу оператор-библиотечарь проделывает следующую процедуру.

Если входной документ является программой, выполняется операция открытия карточки учета (5). В карточке проставляются идентификатор и отметка о нахождении программы в перфореции. В "Регистр учета программ" (15) заносятся идентификатор, автор, дата поступления и количество перфокарт программы. Карточка учета (8) ставится в отделение "Перфореция" каталога карточек учета (13) и программа отправляется на перфорецию в отдел подготовки данных.

Если входной документ является схемой отладки (5), производится коррекция каталога схем отладки (9), т.е. замена старой схемы новой в каталоге схем отладки (50). Если при этом будет замечено изменение старого или присвоение нового участка на личном пакете магнитных дисков, производится выделение свободного участка МД (II) согласно "Регистру участков МД" (10). После чего подготавливается и перфорируется новый управляющий оператор EXTEXT (операции 12 и 13) для этого участка. Карточка учета (8) данной программы из отделения "Библио-

тека" поступает, в случае отсутствия перфорации в отделение "На обработку", в случае перфорации - в отделение "Перфорация" каталога карточек учета (13).

Если входной документ является бланком операторов управления (6), он сверяется с имеющимся бланком из "Каталога операторов управления" (60). На новом бланке делаются пометки, какие операторы управления надо перфорировать (какие изменились) и производится перфорация (операции 8,12,13).

Если входной документ требует большую перфорацию, он отправляется в отдел подготовки данных (операция 6), если обмен перфорации небольшой, в пределах нескольких карт, то в целях большей оперативности ее выполняет оператор системы отладки (операции 13,14).

В любом случае, при отправке информации на перфорацию, карточка учета изымается из отделения "Библиотека" "Каталога карточек учета" (13), в ней делается отметка о нахождении информации в перфорации и она помещается в отделение "Перфорация".

После всех перфорационных работ производится коррекция (15), т.е. замена старых перфокарт новыми, удаление указанных или вставка новых. После коррекции по каждому носителю делается отметка в карточке учета, что изменения внесены. В случае, когда по данной программе все изменения внесены, карточка учета из отделения "Перфорация" перекладывается в отделение "На обработку" "Каталога карточек учета" (13).

Этим цикл подготовки информации кончается.

6.2. Обработка информации

Цикл обработки информации (рис.3.4.) состоит из операций: комплектации пакета обработки, обработки на ЭВМ, проверки результатов и раскомpletации пакета. Цикл кончается доставкой результатов: распечатки АЦПУ, протокола ПМ, а так же бланков входных документов программистам.

6.2.1. Комплектация пакета обработки

За несколько часов до выделенного машинного времени или в строго установленное время суток библиотечарь берет карточки учета (8) из отделения "На работу" "Каталога карточек учета" (13), сортирует их на те, которые требуют трансляцию и те, которые требуют только запуска программы. Если на карточке учета требуется трансляция и запуск, в текущий пакет обработки программа войдет только на трансляцию.

Сначала пакетируются все транслируемые программы, для чего из "Библиотеки программ" (11) берутся программы на перфокартах, согласно карточкам учета, и объединяются в общий пакет. Соответствующие карточки учета при этом перемещаются в отделение "Машина" "Каталога учебных карточек" и идентификаторы программ записываются в графе "Шифр программы" бланка "Задания на работу", а в графе "Название работы", напротив всех транслируемых программ, пишется "Трансляция".

Потом пакетируются все программы, которые требуют запуска. Для них из "Каталога схем отладки" (50) изымаются схемы отладки, согласно которым они пакетируются. Из "Подбиблиотеки операторов управления" (66) берутся перфокарты операторов управления данной программы, к ним добавляются из "Подбиблиотеки данных" (33) данные на перфокартах, если такие требуются согласно схеме

отладки. Остальные носители данных (МЛ, МД) складываются отдельно.

Карточки учета укомплектованных программ помещаются в отделение "Машина", а карточки программ, которые невозможно укомплектовать, в отделение "Библиотека" каталога карточек учета. При этом "схемы отладки" последних с пометками причин невозможности укомплектования, отправляются программистам.

Каждая запускаемая программа так же записывается в "Задании на работу", а напротив всех идентификаторов запускаемых программ, в графе "Название работы" пишется "работа". В графе "Указания библиотекаря" записываются регистрационные номера тех носителей, кроме перфокарт, которые передаются оператору ЭВМ. "Задание на работу" пишется в двух экземплярах.

6.2.2. Обработка на ЭВМ

Оператору ЭВМ передаются:

- 1) "Задание на работу" (9) на перфокартах,
- 2) Пакет обработки (14),
- 3) "Схема отладки" (5),
- 4) Внешние носители: МЛ, МД.

О получении и соответствии всего пакета заданию, оператор ЭВМ расписывается на одном из экземпляров "Задания на работу".

Оператор ЭВМ выполняет работы строго в установленной в "Задании" последовательности. При невозможности выполнения той или иной работы оператор отмечает в графе "Замечания оператора" "Задания".

Во время работы на ЭВМ оператор руководствуется своими служебными инструкциями. Установку носителей информации он выполняет согласно схемам отладки перед запуском очередной программы. После работы оператор

сдает библиотекарю пакет обработки и результаты (листы АИФУ и полный протокол ПМ) согласно "Заданию на работу".

Учет расхода машинного времени производится по протоколу ПМ.

6.2.3. Проверка результатов и раскомплектация пакета

После получения результатов с ЭЕМ библиотекарь производит раскомплектацию пакета обработки (3). Носители программ на перфокартах и данные размещаются в своих подбиблиотеках, схемы отладки - в каталоге. При этом, по - программно, согласно "Заданию", проверяются полученные результаты на их наличие и читаемость. Карточки учета, по которым все работы выполнены, изымаются из отделения "Машина", на них отмечается выполнение работы, и они перемещаются в отделение "Библиотека" "Каталога карточек учета" (13).

Карточки учета, работа по которым не выполнена вследствие ошибок ЭЕМ, помещаются в отделение "На работу". Там же помещаются и карточки учета, по которым результативно выполнена только трансляция, но которые требуют еще и запуск программы. Карточки учета, по которым в ходе трансляции программа не получена, и которые требуют запуска программы, помещаются в отделение "Библиотека" с пометкой (вертикальная красная черта), что программы в библиотеке объектных модулей нет.

В "Задании на работу" заполняется графа "Причина невыполнения" и "Задание" подписывается для хранения в "Подшивке заданий на работу" (30).

СТРАНИЧНАЯ ОРГАНИЗАЦИЯ ПАМЯТИ НА МАГНИТНЫХ ДИСКАХ

Э.Я. Ботомолс, Г.В. Лучкин

В работе описана система страничной организации памяти на магнитных дисках (SOP), являющейся развитием логической СУБВ операционной системы ДОС/ЕС. Система SOP предназначена для управления накоплением и выборкой записей переменной длины и структуры на магнитных дисках с обеспечением как последовательного, так и прямого доступа к ним.

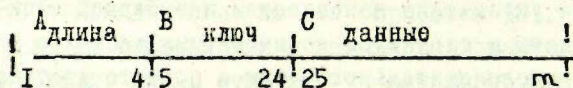
I. Основные понятия SOP

Информация, управляемая системой SOP, располагается на магнитных дисках в файле SOP, который рассматривается как набор физических страниц равной длины (1128 байтов), пронумерованных от 0 до N . В системе SOP физическими страницами называются специально оформленные записи файла с организацией *REGIONAL* (1), причем ключами этих записей являются номера соответствующих физических страниц. Размер файла, т.е. количество физических страниц, определяется пользователем произвольно ($N \leq 17999$) при инициировании файла SOP.

Записи пользователя, размещенные системой на физических страницах, являются строками физических страниц. Внутри физических страниц строки идентифицируются номерами строк, лежащими в пределах от 1 до 63. Физический адрес записи в файле SOP состоит из номера физической страницы и номера строки на ней, доступ к записи по ее физическому адресу заключается в считывании в оператив-

ную память требуемой физической страницы и поиске на ней нужной строки. Строки физической страницы располагаются на ней одна за другой без пропусков. При аннулировании строки происходит смещение последующих строк на освободившееся место с увеличением числа свободных байтов в конце страницы.

Записи пользователя, обрабатываемые системой *5 OR*, имеют следующую структуру, называемую "внешней структурой записи":



А - поле длиной 4 байта, первые два байта которого содержат целое двоичное число, выражающее общую длину записи;

В - поле длиной 20 байтов, содержащее буквенно-цифровой ключ записи;

С - поле длиной $4 \cdot n$ ($n = 0, 1, 2, \dots, 266$), содержащее данные пользователя.

Номер физической страницы, на которой должна быть размещена запись пользователя, определяется с помощью алгоритма рандомизации, применением его к ключу записи. Алгоритм рандомизации должен давать по возможности равномерное распределение записей по всем физическим страницам файла *5 OR*. Используемый в системе *5 OR* алгоритм рандомизации описан в статье "Программа рандомизации на АССЕМБЛЕРЕ и организация ее вызова из PL-процедуры" настоящего сборника. Записи, рандомизация, ключей которых дает равные номера физических страниц, называются синонимами. Синонимы, не уместающиеся на соответствующей физической странице, размещаются на ближайших следующих за ней, имеющих свободные области (последовательность стра-

ниц является циклической, т.е. следующей за последней физической страницей файла считается физическая страница с номером 0). Синонимы одной физической страницы связываются в логическую цепь с помощью адресных ссылок, для этого записи при помещении на физические страницы снабжаются двумя служебными полями: полем собственного адреса и полем адресного сцепления. Поле собственного адреса содержит физический адрес самой записи и идентифицирует ее, а поле сцепления содержит физический адрес записи, следующей за данной в цепи синонимов. Каждое из этих полей имеет длину 4байта. Структура записи, оформленной таким образом, называется "внутренней структурой записи". Вход в цепь задается специальной служебной записью (анкером цепи), расположенной в начале соответствующей физической страницы. Анкерная запись— всегда строка с номером ноль. Каждая физическая страница имеет анкерную запись, расположенную в ее начале, длина анкерной записи — 32 байта. Последняя запись цепи всегда имеет адресную ссылку на анкер этой цепи, т.е. цепь синонимов является замкнутой. При отсутствии записей в цепи синонимов анкер имеет адресную ссылку на себя (пустая цепь). Цепь синонимов, оформленная таким образом, называется логической страницей файла SOP. Анкерная запись не только задает вход в соответствующую (приклепленную) цепь синонимов, но и содержит информацию о физической странице, строкой которой является (например, о свободном месте на физической странице, о свободных номерах строк и др.).

Схематически структура файла системы SOP приведена на рис.1.



Рис. I. Структура файла SOP.

Здесь изображены три логические страницы: нулевая - с записями 0/I, I/I, 2/I и 0/2, первая с записями 2/3 и 2/2 и вторая - пустая.

В общем случае, поиск записи по заданному ключу в системе SOP заключается в рандомизации ключа с получением анкера логической страницы и последовательном просмотре логической страницы путем движения по логической цепи с сравнением ключей.

2. Программное обеспечение SOP

В состав программного обеспечения системы SOP входят четыре процедуры, непосредственно доступные пользователю:

- P405I0, процедура инициирования файла SOP,
- P402II, универсальная процедура обслуживания файла,
- P000ND, процедура поиска записи по ключу,
- P000NS, процедура последовательного просмотра файла SOP.

Эти процедуры написаны на языке PL/I. Кроме того, на языке ASSEMBLER написана служебная процедура рандомизации (рандом-генератор), имеющая имя P000RG.

Следует отметить, что во всех процедурах файл SOP имеет имя F4001 с назначенным логическим устройством SYS019, поэтому в заданиях (JOB), использующих программы обеспечения SOP должны встретиться следующие операторы управления заданиями:

```

// ASSIGN SYS019, <другие параметры>
// DLBL F4001, <другие параметры>
// EXTENT SYS019, <другие параметры>
    
```

Процедура P40510

Процедура P40510 является процедурой инициирования файла SOP. Перед ее выполнением участок (участки) диска, отводимые под файл, должны быть форматизованы с помощью системной программы CLRDISK. Оператор модификации для нее имеет вид:

И — UCL — B = (K=0, D=128).C'*,0Y

Инициирование файла SOP заключается в размещении в начале физических страниц анкерных записей, задающих пустые логические и физические страницы.

Процедура P40510 оформлена, как главная, и может быть выполнена с помощью управляющего оператора:

И — EXEC — P40510.

Процедура P40510 считывает с *SYSPT* одну карту, в первых восьми колонках которой пользователь перфорирует целое десятичное число, равное количеству физических страниц файла SOP, подлежащих инициированию. В конце работы процедуры на *SYSLOG* выдается сообщение

ИНИЦИИРОВАНО л л л л л л л л СТРАНИЦ

Число л л л л л л л л показывает, сколько физических страниц файла инициировано, если оно не совпадает с заданным, то это означает, что в ходе работы процедуры возникла ситуация KEY и выполнение процедуры закончилось досрочно.

Процедура P40211

Процедура P40211 является универсальной процедурой, обслуживающей файл SOP. Процедура обеспечивает выполнение следующих четырех функций по обмену с файлом:

- поиск записи и выдача ее пользователю,

- помещение новой записи в файл,
- изменение (корректирование) хранимой в файле SOP записи,
- исключение (аннулирование) записи из файла SOP.

Предусмотрено выполнение всех четырех указанных функций обмена в двух режимах работы: с рандом-генератором и без него. Режим работы с рандом-генератором является стандартным.

Режим работы процедуры P402II без рандом-генератора допускает явное задание номера логической страницы, к которой требуется применить операцию обмена. Запись, помещенная в файл SOP в режиме работы без рандом-генератора, может быть найдена (для чтения, изменения или исключения) только при работе в этом же режиме. Возможность работы в специальном режиме может использоваться для хранения в файле SOP секретных записей. Для удобного пользования процедура обладает еще тремя дополнительными входами, предназначенными для:

- поиска записи в режиме работы с рандом-генератором (отличается от основного входа меньшим числом параметров),
- открытия файла SOP и настройки процедуры P402II перед выполнением первой операции обмена,
- закрытия файла.

Обращение к процедуре P402II через основную точку входа имеет вид:

CALL P402II (VOBR, KL, ZAP, OTVP, OBL) :

Параметры процедуры:

VOBR - переменная с атрибутом **Fixed (I)**. Задает вид операции обмена с файлом SOP. Может принимать следующие значения:

- 0** - поиск (чтение) записи,
- I** - помещение новой записи в файл,

2 - изменение (корректирование) записи,

3 - исключение (аннулирование) записи из файла.

Другие значения параметра недопустимы.

KL - переменная с атрибутом *CHARACTER* (20).

Задаёт ключ записи, к которой применяется одна из операций обмена с файлом *SOP*.

ZAP - параметр, с помощью которого производится обмен с файлом *SOP*. Для случая, когда *VOBR = 1* или *VOBR = 2* задаёт поле, содержащее запись внешней структуры. В случае, когда *VOBR = 0*, параметр должен задавать поле длины достаточной для размещения запрашиваемой записи. В случае, когда *VOBR = 3* содержимое поля *ZAP* не имеет значения (можно, например, в качестве фактического параметра использовать константу $\emptyset$).

OTVP - переменная с атрибутом *CHARACTER* (2). Задаёт поле ответа, через которое в вызывающую процедуру передается информационный буквенно-цифровой код, сообщающий о результате работы Р402II. Информационный код имеет вид Zn , где Z - буква, задающая класс сообщения, n - цифра, уточняющая его. Буква Z может принимать значения 'У', 'Н', 'Р', что означает, соответственно, 'У' - успешное выполнение заданной операции обмена, 'Н' - отсутствие возможности провести заданную операцию обмена из-за наличия записи с таким ключом в файле *SOP* (при *VOBR = 1*), или из-за отсутствия таковой в файле (при *VOBR = 0, 2, 3*), 'Р' - процедура не может выполняться вследствие ошибок в значениях фактических параметров. Цифра n конкретизирует сообщение. Все возможные информационные коды приведены в таблице I.

OVL - переменная с атрибутом *CHARACTER* (9). Параметр задает режим работы процедуры (с рандом-генератором и без него). Способ его задания можно сформулировать следующим образом: параметр должен задавать символьную строку (константу или переменную), среди первых девяти символов которой должен содержаться по крайней мере один символ "ж". Если "ж" размещена в первом байте поля **OVL**, то тем самым задается режим работы с рандом-генератором, если нет, то задается режим работы без рандом-генератора. В последнем случае поле **OVL** содержит строку вида $\underbrace{nn \dots n}_m \text{ * }$, где $nn \dots n$ - m значное ($1 \leq m \leq 8$) число, возможно содержащее незначащие нули, явно задающее номер логической страницы, к которой требуется применить операцию обмена с файлом **SOP**.

Пример:

OVL = 'ж' - задается режим работы с рандом-генератором,

OVL = '0ж' - операция обмена будет применяться к нулевой логической странице.

Вызовы процедуры **P40211** через дополнительные точки выхода имеют вид:

а) Вызов: *CALL NSIPD (KL, ZAP, OTVP)*;
эквивалентен следующему:

CALL P40211 (0, KL, ZAP, OTVP, '');*

б) Вызов: *CALL NSIPON*;

Такой вызов процедуры **P40211** необходимо осуществить до первого запроса на выполнение операций обмена. При входе в процедуру **P40211** через эту точку происходит открытие файла **SOP** и настройка процедуры для работы, что заключается в считывании в оперативную память нулевой фи-

зической страницы, определения размеров файла и подготовки некоторых рабочих полей.

в) Вызов: *CALL NSIPCL*;

Вызывается процедурой пользователя в момент, начиная с которого больше не будет обращений к файлу *SOP*. Единственное выполняемое действие при вызове процедуры *R402II* через точку входа *NSIPCL* - закрытие файла *SOP*.

Процедура *R402II* рассчитана на пользователя, нуждающегося в различных функциях обмена с файлом *SOP*. Процедура достаточно громоздка, поэтому не рекомендуется ее использовать как процедуру чтения, для этого предназначена специальная, более компактная процедура *R000ND*.

Процедура *R000ND*.

Процедура *R000ND* предназначена для широкого круга пользователей, нуждающихся лишь в получении записей из файла *SOP* по ключу и не занимающихся изменением его. Эта процедура имеет дополнительные входы:

а) дополнительная точка входа *NSIDPN*

Вид обращения:

CALL NSIDPN;

Назначение этой точки входа и выполняемые действия при вызове идентичны точке входа *NSIDPN* процедуры *R402II*.

б) дополнительная точка входа *NSICLS*

Вид обращения:

CALL NSICLS;

Назначение и выполняемые действия те же, что у *NSIPCL* процедуры *R402II*.

Обращение к процедуре *R000ND* через основную точку входа имеет вид:

CALL R000ND (KL, ZAP, OTVP);

Таблица I. Информационные коды ответа
процедуры P402II

код	Значение кода
У 0	Чтение успешно
У 1	Помещение новой успешно
У 3	Корректирование успешно
У 5	Аннулирование успешно
Н 0	Чтение не прошло (записи в 5 OP нет)
Н 1	Помещение новой записи не прошло, в файле 5 OP содержится запись с таким же ключом
Н 2	Для новой записи нет места в файле 5 OP
Н 3	Корректирование не прошло, в файле 5 OP не содержится корректируемая запись
Н 4*	Корректирование не прошло, для новой записи нет места
Н 5	Аннулируемой записи нет в файле 5 OP
P 0	Для $VOBR = 1$ или $VOBR = 2$ значение параметра $K4$ не совпадает с содержимым поля ключа в 2 AP
P 1	Неверно значение $VOBR$ ($VOBR \notin \{0, 1, 2, 3\}$)
P 2	Длина 2 AP больше 1088 или меньше 24
P 3	Длина 2 AP не кратна 4
P 4	Параметр $OB4$ задает номер логической страницы, превосходящей допустимый максимум
P 5	$OB4$ содержит не цифру
P 6	$OB4$ не содержит символ "x" среди первых девяти символов.

*) следует помнить, что при этом старая запись исключается из файла 5 OP, и если это нежелательно, следует ее поместить в файл 5 OP вновь, запомнив перед выполнением операции корректирования.

Значение параметров процедуры описано выше. Пользование процедурой *R000ND* ничем не отличается от пользования дополнительным входом *NSLRD* процедуры *R402II*.

Процедура *R000ND* через поле *OTVP* выдает информационные коды 'Y0' и 'N0'. 'Y0' - в случае успешного поиска записи; 'N0' - в случае неудачного (при отсутствии искомой записи в файле *SOP*).

На рис.2. изображена блок-схема процедуры *R000ND*
Пояснения к блок-схеме:

Блоки 1-3. Рандомизируется ключ искомой записи, проверяется не находится ли в оперативной памяти физическая страница, содержащая анкер логической страницы с вычисленным номером, и, если нет, то она вызывается с диска.

Блок 4. Структура, выделяющая отдельные поля строки физической страницы, накладывается на анкер, делая тем самым доступным для обработки его поле сцепления. Под "текущей записью" понимается строка, на которую наложена описанная структура.

Блоки 5,6. Проверяется, не достигнут ли конец логической страницы. Если достигнут, то формируется информационный код 'N0' и осуществляется выход из процедуры.

Блоки 7,8. По физическому адресу из поля сцепления текущей записи осуществляется поиск записи. Для этого, если нужно, считывается соответствующая физическая страница, и затем все ее строки просматриваются, пока не будет найдена строка с нужным номером. На найденную строку накладывается описанная выше структура.

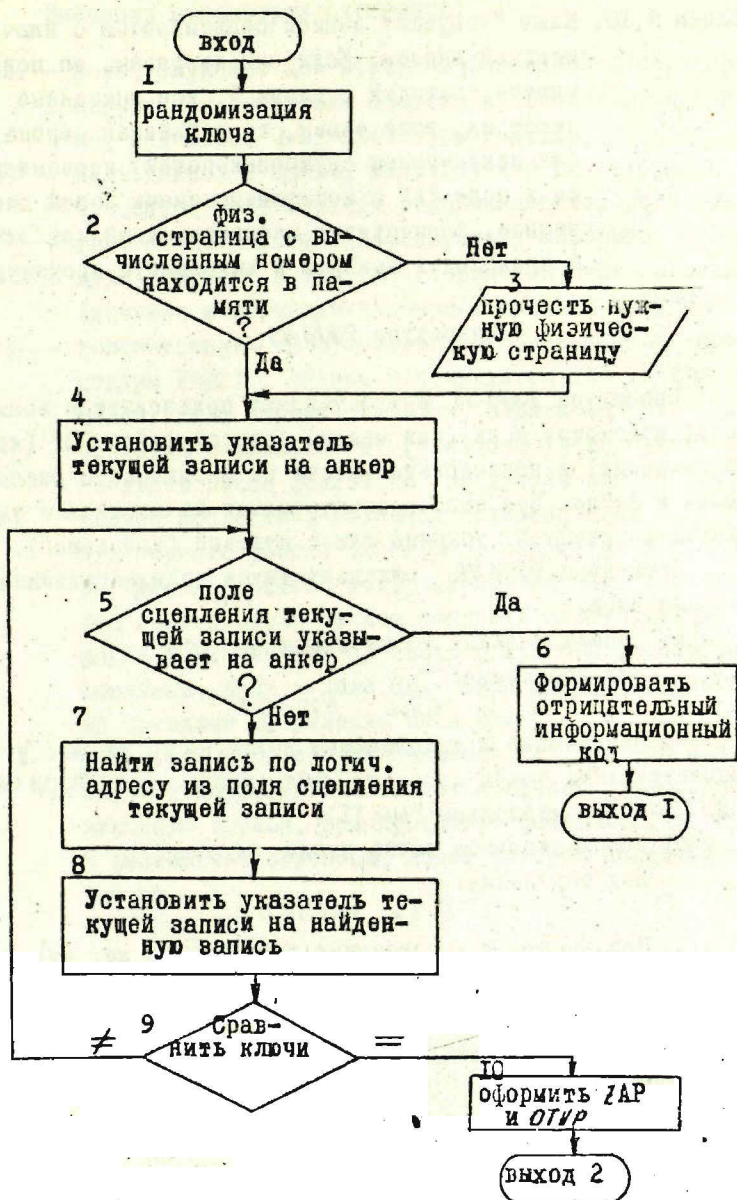


Рис.2. Блок-схема процедуры RRRD ND

Блоки 9,10. Ключ "текущей" записи сравнивается с ключом искомой записи. Если они не равны, то повторяются, начиная с блока 5, все описанные действия, если равны, то найденная строка (за исключением служебных полей) пересылается в поле ZAP с модифицированным полем длины записи, формируется информационный код 'УФ' и происходит возврат в вызывающую процедуру.

Процедура P000NS

Процедура P000NS предоставляет пользователю возможность просмотра всех (или части) записей файла SOP (кроме анкерных) в последовательности их физического расположения в файле. При этом просмотр может начинаться с указанной физической страницы или с нулевой (начальной).

Процедура P000NS обладает двумя дополнительными точками входа:

- а) Дополнительная точка входа NSISON;

Вид обращения:

CALL NSISON;

Назначение и выполняемые действия те же, что у дополнительных точек входа NSIOPN (у процедуры P000ND) и NSIOPN (у процедуры P402II).

- б) Дополнительная точка входа NSISCL

Вид обращения:

CALL NSISCL;

Назначение и выполняемые действия те же, что у дополнительных точек входа NSISCL (у процедуры P000ND) и NSIPCL (у процедуры P402II).

Обращение к процедуре P000NS через основную точку входа имеет вид:

CALL P000NS (ZAP, OTVP, OVL);

Значения параметров следующие:

- Z AP* - поле, служащее для передачи очередной записи процедуры пользователя. Правила пользования им такие же, как полем *Z AP* процедуры *P402II*.
- OTUP* - поле длиной 2 байта, используемое для передачи процедуры пользователя информационного кода ответа (аналогично соответствующему параметру процедуры *P402II*). В таблице 2 приведены все возможные значения информационных кодов процедуры *P000NS*.
- OVL* - соответствует по оформлению параметру *OVL* процедуры *P402II*, однако используется для других целей. Если строка символов, задаваемая *OVL*, начинается с символа "ж" (звездочка), то будет осуществлен поиск физически следующей записи относительно последней, выданной пользователю. Для первого (после *MSISON*) обращения к процедуре *P000NS*, такое содержимое поля *OVL* означает, что пользователь желает получить первую запись файла *SOP*, начиная просмотр с нулевой физической страницы. Если в поле *OVL* записано число (согласно правилам оформления поля *OVL*), т.е. последовательность цифр заканчивающаяся "ж", то это означает, что пользователь желает получить запись, обнаруженную первой, при просмотре файла *SOP*, начатом с физической страницы, номер которой задается этим числом.

Таблица 2. Информационные коды ответа
процедуры *RODD NS*

Код	Значение
У Ø	Запись выдана
И Ø	Достигнут конец файла
P1	Поле OBL содержит "не цифру" до появления первой звездочки
P2	Символ "ж" не содержится среди первых девяти символов поля OBL
P3	Номер физической страницы, явно указанный в OBL, превосходит допустимый максимум.

Заключение

Система SOP была создана как часть общесистемного математического обеспечения разрабатываемой в ВЦ ЛГУ отраслевой автоматизированной системы управления АСОИ-ПЕНС, где служит для управления массивом НСИ (нормативно-справочной информации). Однако система SOP может использоваться для решения широкого круга задач экономического и не экономического характера.

В настоящий момент разрабатывается система *ROD* (рандомизованная организация данных), являющаяся развитием идей, используемых в SOP. Предусматривается реализация таких возможностей, как сегментация записей, введение ключа переменной длины, расширение понятия логической цепи, развитие способов обмена с файлом на диске, применение настраиваемого алгоритма рандомизации. Система *ROD* будет основным средством управления базой данных в системе банка данных *BD*.

СИНТЕЗ И ПРИМЕНЕНИЕ АЛГОРИТМОВ,
ГЕНЕРИРУЮЩИХ ЗАДАНИЯ В ДОС/ЕС

Х.К. Буш

1. Генерация заданий в ДОС/ЕС.

Операционная система ДОС/ЕС предоставляет программисту широкие возможности в составлении пакетов заданий. Совокупность управляющих операторов и директив образует своеобразный и довольно сложный язык управления заданиями. С помощью средств этого языка возможна реализация весьма сложных процессов организации работы ЭВМ как при отладке программ, так и во время эксплуатации готовых программ. Управляющие операторы и директивы дают возможность определять файлы, вызывать выполнение системных и собственных программ, делать разнообразные назначения и переназначения внешних устройств, пользоваться системными и личными библиотеками и т.д. С помощью управляющих карт можно, например, произвести трансляцию алгоритма, находящегося в личной библиотеке исходных модулей, и каталогизировать полученные объектные модули без вывода на перфокарты.

Однако количество управляющих карт в пакете задания даже в случае сравнительно простой обработки исходного модуля может достигнуть нескольких десятков на каждый модуль. Многие управляющие карты содержат название модуля и должны перфорироваться заново для каждого нового модуля, хотя их применение является совершенно стандартным. Для сознательного использования языка управления заданиями необходимы некоторые знания и опыт. Поэтому из огромных потенциальных возможностей ДОС/ЕС часто используются лишь немногие стандартные варианты структуры пакетов заданий. Это приводит к недостаточному использо-

ванию системных и личных библиотек, к непроизводительным затратам машинного времени на повторную совместную трансляцию правильных и исправленных исходных модулей в режиме непосредственного выполнения, к злоупотреблению выводом объектных модулей на перфокарты, к недостаточному использованию возможностей переопределения файлов и переназначения внешних устройств и к другим нежелательным явлениям.

Во время отладки программ в большинстве случаев достаточно иметь несколько стандартных возможностей способов обработки модулей. Во время эксплуатации готовых программ определения файлов, назначения внешних устройств, последовательность вызова фаз также редко меняются.

Напрашивается вывод, что в ДОС/ЕС необходимо ввести что-то аналогичное механизму процедур ОС/ЕС, хотя бы в упрощенной форме. То есть необходимо иметь возможность достаточно просто вызвать выполнение заранее введенных в систему последовательностей управляющих карт. Должна быть предусмотрена возможность модифицирования некоторых карт, например, для указания конкретного имени модуля. И, наконец, сам процесс введения в систему новых последовательностей карт должен быть достаточно простым.

Один из возможных способов решения перечисленных проблем в рамках обычной ДОС/ЕС состоит в том, что во время выполнения некоторой программы на внешнем носителе (например, магнитной ленте) создается последовательность записей длиной 80 байт - образы необходимых карт управления и информации. После конца работы программы системный ввод переназначается на соответствующее внешнее устройство, и начинается выполнение только-что сгенерированного задания. В конце этой последовательности на МЛ должны быть предусмотрены операторы или директивы, обеспечивающие необходимую коммутацию ввода.

Необходимо подчеркнуть, что все действия по перенав-

начению системного ввода, генерации и выполнению заданий выполняются в режиме пакетной обработки, без участия человека - оператора. Лишь в случае неисправности ЭВМ или грубых ошибок программиста может потребоваться вмешательство оператора.

Программы, способные генерировать последовательности управляющих и других карт, легко могут быть написаны на языке ПЛ/1. Назовем их программами 2 уровня.

Можно генерировать шаги заданий, задания или целые потоки заданий.

2. Синтез алгоритмов, генерирующих задания.

Итак, в принципе можно для часто используемых последовательностей управляющих карт написать алгоритмы 2 уровня. Но написание и отладка такого алгоритма само по себе является трудоемким процессом.

Каждый такой алгоритм должен содержать обычные операторы начала главной процедуры, описания переменных и файлов, символьные строки с текстом карт, операторы вывода и т.д. Структура таких алгоритмов почти одинакова, меняются лишь названия, содержимое и количество выводимых образов карт, количество и величина массивов строк. Может несколько варьировать способ модификации карт для конкретного случая применения. Может отсутствовать ввод дополнительной информации (например, если ни одна из выводимых карт не должна быть модифицирована).

Довольно широкий класс алгоритмов 2 уровня можно синтезировать, используя для этой цели специально разработанный алгоритм. Назовем такой алгоритм алгоритмом 3 уровня. Исходной информацией для такого алгоритма служит название будущей программы 2 уровня и предполагаемый вид последовательности карт, которые должна "уметь" генерировать программа 2 уровня, а также некоторая информация о вводе-выводе. На основе этих данных алгоритм 3 уровня

синтезирует текст программы 2 уровня. Возможно (но не обязательно) сделать так, чтобы алгоритм 3 уровня также подготовил бы задание для обработки синтезированного алгоритма. В таком случае после процесса синтеза можно сразу пользоваться готовой программой 2 уровня, каталогизированной в библиотеке абсолютных модулей.

3. Назначение и возможности программы SINTVAD4

Программа 3 уровня SINTVAD4 предназначена для синтеза и обработки программ 2 уровня, способных генерировать заданные последовательности образов карт на магнитной ленте. Программа написана на языке ПЛ/1 и синтезирует алгоритмы 2 уровня также на ПЛ/1. Кроме того, может быть подготовлено задание (на МЛ) для обработки синтезированного алгоритма вплоть до каталогизации отредактированной фазы в библиотеку абсолютных модулей.

Рассматриваемая здесь версия 4, модификация 2 программы SINTVAD4 может работать на ЭВМ ЕС 1020 с объемом оперативной памяти 64 К байт. Используются два устройства МЛ с рабочими лентами и участок диска.

Для запуска процесса синтеза необходимы следующие карты:

```
// ASSGN SYS011,X'280'  
// ASSGN SYS012,X'281'  
// EXEC SINTVAD4  
  < режим синтеза >  
  < префикс и имя модуля >  
  < параграфы синтеза >  
/*  
ASSGN SYSIN,X'281'
```

Карта режима синтеза должна начинаться со слова SINTOPI, за которым могут следовать 9 необязательных параметров, определяющих способ обработки синтези-

рованного алгоритма:

USAT - все задание синтеза каталогизируется в библиотеку исходных модулей, обновляется для нумерации карт и присоединения идентифицирующих букв в позициях 73-76 каждой карты,

UPRINT - производится печать и перфорация обновленного задания синтеза из библиотеки,

SPUNCH - производится печать и перфорация синтезированного исходного модуля из библиотеки исходных модулей (синтезированный алгоритм всегда каталогизируется),

COMPILE - синтезированный алгоритм транслируется,

RCAT - полученные объектные модули каталогизируются в библиотеку объектных модулей,

CATAL - редактируется и каталогизируется фаза, соответствующая синтезированному алгоритму,

RPUNCH - производится перфорация объектного модуля,

CPUNCH - производится перфорация фазы,

#PRINT - в текст синтезированного алгоритма, перед концом главной процедуры, включается вызов внешней процедуры CALL #PRINT('CCCCCCCC');

где фактический параметр CCCCCCCC - имя самого синтезированного алгоритма. Если синтезируется группа алгоритмов, можно в этой процедуре запрограммировать какие-то общие действия, например, печать сообщений, с коррекциями по фактическому параметру - названию конкретного вызываемого алгоритма. Процедура должна быть транслирована и закаталогизирована в библиотеку объектных модулей до начала синтеза.

Параметры режима синтеза разделяются пробелами или запятыми. Порядок их расположения не имеет значения. Любой из параметров или даже все они могут отсутствовать, но имеющиеся должны образовать логически осмысленную последовательность. Например, нельзя требовать

редактирования фазы, если не проводится трансляция. В таких случаях печатается сообщение об ошибке и задание снимается. Карта режима синтеза может вообще отсутствовать, в таком случае принимается, что затребованы все параметры, кроме #PRINT .

Префикс и имя синтезированного модуля используются при каталогизации модуля в библиотеку исходных модулей. Префикс может быть цифрой или буквой, кроме буквы "u". Название (имя) должно состоять из 8 знаков, допускаемых программой БИБЛИОТЕКАРЬ ДОС/ЕС. С таким же именем, но префиксом u каталогизируется задание синтеза. Для главной процедуры и, следовательно, объектного модуля синтезированного алгоритма берется имя, состоящее из двух знаков номера и нечетных букв имени. Для файлового модуля добавляется с начала буква F. При обновлении задания синтеза и исходного модуля в библиотеке исходных модулей в позициях 73-76 каждой карты вносятся нечетные буквы имени. Отредактированная фаза получает полное имя исходного модуля.

Каждый параграф синтеза начинается с карты, содержащей в первой позиции знак \$, а во второй - латинскую букву. Далее должны следовать карты-прообразы тех карт, которые будут синтезироваться на МЛ программой 2 уровня. Параграфов синтеза может быть один или несколько (до 10). В зависимости от параграфов синтеза можно получить два вида синтезированных алгоритмов.

Вид "А". Только один параграф синтеза и употреблена буква А за знаком параграфа. В этом случае синтезированная программа при работе не будет вводить никакую дополнительную информацию. Такие программы полезны для вызова постоянных заданий. При вызове синтезированной фазы карты-прообразы будут воспроизведены на системном устройстве № 12. После этого необходимо переназначить ввод на это же физическое устройство, и начинается выполнение сгенерированного задания.

При написании карт-прообразов должны быть соблюдены некоторые правила. Каждая из этих карт будет вписана в некоторый оператор присваивания строк в синтезированном алгоритме. Поэтому надо заботиться о том, чтобы каждая карта-прообраз после присоединения слева и справа эпострофов образовала бы синтаксически правильное выражение со значением строки символов. Так, например, эпострофы в этих картах должны быть повторены. Еще одна условность: точка в первой позиции карты-прообраза при синтезе будет заменена на наклонную черту. Это позволяет ввести в задание синтеза прообразы карт "конец файла" и "конец задания".

Рассмотрим конкретный пример. С помощью какой-то программы MULTREGN необходимо последовательно обработать три файла на диске. Одно и то же имя файла определяется для разных участков диска. Задание для синтеза программы 2 уровня:

```
SYNTOPT UCAT,UPUNCH,SPUNCH,COMPILE,NCAT,CATAL
X,SYNTHSAT
НА
// DLBL BPLFLI,"1SYS01"
// EXTENT SYS001,011,1520,112,3
// EXEC MULTREGN
// DLBL BPLFLI,"1SYS02"
// EXTENT SYS002,011,1524,04,6
// EXEC MULTREGN
// DLBL BPLFLI,"1SYS03"
// EXTENT SYS003,011,1527,04,9
// EXEC MULTREGN
ASSGN SYSIN,X"00C"
```

Будет синтезирован алгоритм вида "А", и процесс завершится каталогизацией готовой фазы в библиотеку абсолютных модулей. Текст синтезированного алгоритма:


```

*SYNT8;PROCEDURE OPTIONS(MAIN);/*PHASE SYNTTS8T OF SYNTSYS VМ 4.29/
/* JOB GENERATOR MADE 02/01/75 BY SYNTVAD4 PROGRAM */
DECLARE INPROC FILE OUTPUT
ENVIRONMENT (MEDIUM(SYS012,5012) P(00) NOLABEL);
PROG(300) CHARACTER(80); NAME CHARACTER(10);
DECLARE A (10)CHARACTER(60);
OPEN FILE(INPROC);
A ( 1)='// DLBL BPLFLI, 'JJSYS01''
A ( 2)='// EXTENT SYS001,8,1,1520,112,3
A ( 3)='// EXEC MULTREGR
A ( 4)='// DLBL BPLFLI, 'JJSYS02''
A ( 5)='// EXTENT SYS002,8,1,1524,04,0
A ( 6)='// EXEC MULTREGR
A ( 7)='// DLBL BPLFLI, 'JJSYS03''
A ( 8)='// EXTENT SYS003,8,1,1527,04,7
A ( 9)='// EXEC MULTREGR
A (10)='ASSGN SYSIN,X'000''
PUT FILE(INPROC)EDIT(A)(A(0));
CLOSE FILE(INPROC);
END *SYNT8;

```

Для выполнения необходимых действий над файлами достаточно задать

```

// ASSGN SYS012,X'281'
// EXEC SYNTTS8T
ASSGN SYSIN,X'281'

```

Заметим, что независимо от количества карт-преобразов вызов для их выполнения был бы таким же простым. Допускается до 80 карт-преобразов для синтеза одной программы 2 уровня.

Вид "Б". Более чем один параграф синтеза и/или не буква А за знаком параграфа. В этом случае синтезированная программа будет требовать ввод дополнительной информации для модификации преобразов карт. Программы этого типа предназначались для организации обработки

различных программных модулей, поэтому вся вводимая дополнительная информация подразделяется на карты названий модулей и карты текста модулей. Карты названий распознаются по точке во второй позиции карты. Синтезированные программы вида "Б" могут обрабатывать поток карт со структурой название-текст-название-текст и т.д. или со структурой название-название- и т.д., если ввод текста обрабатываемых программ 1 уровня не нужен. В ходе эксплуатации программы выяснилось, что не всегда в карте названия выгодно писать собственно название модуля. В принципе там может быть и любая другая информация, но присутствие точки во второй позиции карты обязательно.

Правила написания прообразов в основном те же, что и для вида "А". Для модификации используется содержимое карты названия программы 1 уровня, которое хранится в 80 - символьной строке INSTR программы 2 уровня, и предполагаемое имя с префиксом (первые 10 символов строки INSTR), хранящееся в переменной NAME программы 2 уровня. В картах-прообразах, вводимых программой 3 уровня, можно предусматривать действия над содержанием строк NAME и INSTR. Синтезированная программа будет выполнять обработку потока данных 1 уровня до возникновения ситуации "конец файла" на вводном устройстве. Начало нового параграфа синтеза указывает, что в этом месте будет помещен текст обрабатываемого алгоритма 1 уровня.

Рассмотрим конкретный пример. Для потока карт со структурой название-исходный модуль-название-... и т.д. необходимо выполнить каталогизацию каждого модуля в библиотеку исходных модулей, нумерацию карт и присоединение первых 4 букв названия в позициях 73-76 каждой карты модуля. Задание для синтеза:


```

SYNTOPT UCAT,UPUNCH,SPUNCH,COMPILE,NOAT,BATAL
Y,CATALS;A
HP
// ASSEN SYS000,X'0000'
// EXEC MAINT
  CATALS1 '||NAME||'
  '||16)||' '||BKEND.
HP
  '||16)||' '||BKEND
  UPDATE '||NAME||'
) END 1.1
  UPDATE '||NAME||'
) PRL 0
  '||171)||' '||SUBSTR(NAME,3,4)||'0000
) END 1.2
..

```

Текст синтезированного алгоритма приведен на следующей странице.

Все синтезированные программы вида "Б" в конце генерированной последовательности карт добавляют "от себя" директиву переключения системного ввода на ввод с перфокарт, карту "конец задания" и специальное задание // JOB SORRY, в котором предусмотрено приведение назначений системных устройств в исходный вид и переключение ввода на ввод с перфокарт. В нормальных условиях выполнения генерированной последовательности дополнительная карта конца задания и специальное задание не выполняются, так как ввод до этого переключается на перфокарты. Однако в случае снятия генерированного задания (точнее, в случае снятия последнего задания в сгенерированном потоке заданий) игнорируются оставшиеся до конца задания карты. После этого начинается выполнение специального задания, которое обеспечивает нормальное продолжение работы.

4. Система обработки программ SINTSYS

Основная цель системы - облегчить процесс составления и отладки программ на машинах ЕС ЭВМ, работающих с ДЭС/ЕС.

```

**CTLI:PROCEDURE OPTIONS(MAIN);**PHASE CATALSIA OF SINTSYS VM 4.2*/
/* JOB GENERATOR MADE P2/01/75 BY SINTVAD4 PROCHAN */
DECLARE SOURCE FILE INPUT
ENVIRONMENT (MEDIUM(SYSTPT,6012) P(001))
(INSTR,NXT) CHARACTER(00); (NEXT,LST)BIT(1); INPROG FILE OUTPUT
ENVIRONMENT (MEDIUM(SYS012,5012) P(00) NOLABEL)
PROG(130) CHARACTER(00); NAME CHARACTER(10)
DECLARE P ( 4) CHARACTER(00)
DECLARE Q ( 0) CHARACTER(00)
ON ENDFILE(SOURCE)GOTO FLOPEN FILE(SOURCE);FILE(INPROG)LIST='0'01
GET EDIT(NXT)(A(00))INXSEG;INSTR=NXT;NAME=NXT;NEXT='0'B;N=0;CO 1=1 TO
300GET FILE(SOURCE)EDIT(NXT)(A(00))IF SUBSTR(INSTR,2,1)='.',THEN DB;
NEXT='1'B;GOTO FORMENDIPROG(1);NEXT=NELIEND;PLIST='1'B;FORMI
P ( 1)='// ASSEN SYS009,X,'000'
P ( 2)='// EXEC MAIN
P ( 3)=' CATALS; 'I'NAME1'
P ( 4)=' '110' '11'BKEND
PUT FILE(INPROG)EDIT(P,(PROG(1) DO
Q ( 1)=' '110' '11'BKEND
Q ( 2)=' UPDATE 'I'NAME1'
Q ( 3)=' END 1:
Q ( 4)=' UPDATE 'I'NAME1'
Q ( 5)=' PRL 0
Q ( 6)=' '1171' '11SUBSTR(NAME,3,4)11'0000
Q ( 7)=' END 1:2
Q ( 8)='//**
PUT FILE(INPROG)EDIT(N)(A(00)1)
IF NEXT A' LST THEN GO TO RX200;
PUT FILE(INPROG)EDIT('ASSEN SYSIN,X,'000'**,
'3'4'// JOB SORRY - CONTINUATION',
'% ON MESSAGE 1122D PRESS RT','ASSEN SYS004,X,'191'**,
'CLOSE SYSPCH,X,'000'1'// LOC',
'ASSEN SYSIN,X,'000'**, '% END'(A(00)1)
CLOSE FILE(INPROG);FILE(SOURCE);
END *CTLI1

```

CTL10000
 CTL10001
 CTL10002
 CTL10003
 CTL10004
 CTL10005
 CTL10006
 CTL10007
 CTL10008
 CTL10009
 CTL10010
 CTL10011
 CTL10012
 CTL10013
 CTL10014
 CTL10015
 CTL10016
 CTL10017
 CTL10018
 CTL10019
 CTL10020
 CTL10021
 CTL10022
 CTL10023
 CTL10024
 CTL10025
 CTL10026
 CTL10027
 CTL10028
 CTL10029
 CTL10030
 CTL10031
 CTL10032
 CTL10033
 CTL10034

Система в описываемой здесь версии 4, модификации 2 состоит из пяти основных и девяти обрабатывающих программ. Кроме того, для работы системы используется несколько измененное ядро супервизора, однако отличия от стандартного варианта супервизора небольшие.

Все 9 обрабатывающих фаз синтезированы с помощью программы **SINTVAD4**, которая также входит в состав системы как одна из пяти основных программ. Каждая из обрабатывающих фаз является программой вида "Б" и предназначена для генерирования потока заданий по обработке программ на любом языке, с которого имеется транслятор на данной машине. Каждая из этих программ генерирует задания по определенной, типовой обработке программ. Таким образом, предлагаются 9 стандартных вариантов обработки.

Имена обрабатывающих программ имеют вид XXXaaae, причем группы букв aaa и ee определяют вид обработки. aaa=CAT : обрабатываемые программы вводятся с перфокарт, причем вводимый поток должен иметь структуру название-текст-название-текст... и т.д. Карта названия должна содержать с первой колонки префикс, точку, имя исходного модуля (от одного до восьми знаков, допускаемых программой БИБЛИОТЕКАРЬ), далее должен следовать один пробел и имя транслятора для этого модуля. Карты названий распознаются по точке во второй позиции. В одном вызове обрабатывающей фазы можно вводить произвольное количество модулей, указывая различные трансляторы. Длина каждого модуля не должна превышать 300 карт. Для обработки каждого модуля генерируется задание, в начале которого предусматривается каталогизация в библиотеку исходных модулей, обновление для нумерации карт модуля, обновление для присоединения идентифицирующих букв в позициях 73-76 каждой карты. Дальнейшая обработка зависит от группы букв ee.

aaa= L1B : обрабатываемые программы уже находятся в библиотеке исходных модулей, и вводимый поток должен иметь структуру название-название-... и т.д. Карта названия составляется так же, как для предыдущего вари-

анта. Обычно с перфокарт вводится лишь начальный текст программ, поскольку средствами программы БИБЛИОТЕКАРЬ можно провести любые изменения в модуле, каталогизированном в библиотеке. При этом можно там же сохранить неизменный вариант. Нумерация карт после исправлений автоматически приводится в порядок. Вообще, пользование библиотеками сильно повышает культуру отладки.

aaa=NGH : эти фазы действуют так же, как и при aaa=LIV, но после конца дальнейшей обработки (определяемой группой ее) вызывается подсистема интерактивной обработки. Эта подсистема позволяет выполнять корректировку программ и составлять произвольные задания работы в режиме диалога с пульта машины. Использование этой подсистемы здесь не будет рассматриваться. Организация интерактивного режима осуществляется программой

NIGHTWRK, второй из пяти основных программ.

ee=TR : исходный модуль с библиотеки с помощью основной фазы SOURCEPR и сервисных программ подготавливается на рабочей ленте, транслируется, полученный объектный модуль с помощью основной фазы FORTMOD и сервисных программ каталогизируется. Последняя из пяти основных фаз, TIMEPR, используется только для печати текущего времени до и после трансляции и еще в некоторых этапах задания.

ee=PI : выполняются все действия, упомянутые в предыдущем пункте, но после этого устанавливается режим // OPTION CATAL и производится редактирование фазы с последующей каталогизацией в библиотеку абсолютных модулей. В состав фазы включается модуль, каталогизированный в предыдущих шагах этого же задания. Однако в данной модификации системы не предусмотрена генерация карт режима меток на лентах и дисках.

eee=EX : выполняются все действия предыдущего пункта, но редактирование проводится в режиме // OPTION LINK и сразу после редактирования вызывается выполнение

программы. Этот вариант обычно полезен для отладки программ. На время выполнения системный ввод данных переключается на ввод с перфокарт.

Перечисленные способы обработки образуют 9 имен обрабатываемых программ.

Рассмотрим конкретный пример. В библиотеке исходных модулей находятся две программы: X.AAAAAAAA и Y.BBBBBBBB. Первая из них написана на ФОРТРАНе-IV, а вторая на ПЛ/I. Необходимо их транслировать и редактировать в виде каталогизированных фаз.

```
// ASSGN SYS012,X'281'
```

```
// EXEC XXXLIBRN
```

```
X.AAAAAAAA FFORTRAN
```

```
Y.BBBBBBBB PL/I
```

```
/*
```

```
A SSCH SYSIN,X'281'
```

Все обрабатываемые фазы и фаза NIGHTWRK после генерирования карт печатают короткое сообщение о виде обработки. Фаза NIGHTWRK также ведет протокол диалога программиста с ЭВМ. Программа SINTVAD4 во время синтеза печатает подробный отчет.

5. Использование системы SINTSYS

Самым важным свойством описанной системы является то, что это открытая система: в ее составе имеются средства, обеспечивающие несложный процесс синтеза всевозможных программ 2 уровня, не обязательно связанных с обработкой программ.

В целях облегчения первоначального запуска системы разработана специальная лента, содержащая запись всей информации для генерации и проверки системы.

Система написана на языке ПЛ/I, подмножество ДС/ЕС /2/, /4/. Макрокоманды генерации ядра супервизора написаны на языке АССЕМБЛЕР /2/, /5/. Первоначальная версия одного из алгоритмов обработки /1/ проверялась

на ЕС 1020 с ДОС/ЕС 1.0, а описанная здесь модификация - на ЕС 1020 с ДОС/ЕС 1.3.

Для оценки затрат машинного времени при обработке приведем данные об обработке программы на ПЛ/1, состоящей из 230 операторов (294 перфокарты), из библиотеки исходных модулей:

генерация задания и подготовка модуля	I мин.40 сек.
трансляция	0 мин.24 сек.
каталогизация объектного и файлового м.	I мин.17 сек.
Редактирование и каталогизация фазы	I мин 08 сек.

Для сравнения укажем, что только на перфорацию объектного и файлового модуля с помощью перфоратора ЕС 7010 при объеме колоды 371 карта необходимо около 3 мин.45 сек, а для ввода в устройства ЕС 6012 этой же колоды во время каталогизации необходимо еще 45 сек. /3/.

ЛИТЕРАТУРА

1. Буш Х.К. Алгоритм для автоматизированной обработки подпрограмм на ПЛ/1 в системе ДОС/ЕС, Государственный фонд алгоритмов и программ, № П000760, 1974.
2. Джермейн К. Программирование на IBM/360. М., 1973.
3. Единая система ЭВМ, под ред. А.М. Ларионова. М., 1974.
4. Килов Х.И. Подмножество ПЛ/1, Рига, 1974.
5. Стебли Д. Логическое программирование в Системе/360. М., 1974.

Организация ввода и накопления данных в АСОИ

Л.М. Васерман

1. Некоторые классификационные признаки АСОИ.

В настоящей статье описывается технологический процесс ввода и накопления данных в автоматизированной системе обработки информации (АСОИ). Предполагается, что накапливаемые данные должны быть организованы в виде файла на магнитных лентах; общий объем постоянно-хранимых данных в рассматриваемой АСОИ составляет 50-500 млн. байт. Таким образом, речь будет идти о технологическом процессе коррекции существенного большого, "главного" файла АСОИ на магнитных лентах, т.е. файла с последовательным доступом к данным. Очевидно, указанный объем этого файла практически исключает возможность его сортировки, поэтому технологический процесс должен быть построен на принципе корректуры — слияния главного файла с некоторым вспомогательным файлом. С другой стороны, следует учитывать основные свойства входного потока, т.е. свойства данных, предназначенных для ввода и накопления в АСОИ. В частности, для описываемого здесь технологического процесса определяющими являются следующие свойства вводимых данных:

1) общий объем данных составляет в год 10-20% от объема постоянно хранимой информации; однако распределение этого объема по дням является неравномерным, возможно увеличение дневного потока в 5-10 раз против среднего, так что в день может вводиться от 100 до 1000 первичных документов;

2) первичные документы АСОИ отличаются большим разнообразием форм, автоматический контроль данных возможен лишь в узких пределах; в то же время технология должна обеспечить высокую достоверность хранимых данных — таким обра-

зом, визуальный контроль ввода должен рассматриваться как неизбежный, вообще говоря, этап технологического процесса;

3) требования к оперативности обработки данных выражаются в том, что срок от момента поступления в АСОИ первичного документа до внесения данных в файл должен составлять не более недели.

Техническая база АСОИ, кроме уже указанного ограничения в выборе магнитных лент в качестве физического носителя главного файла, не предусматривает какие-либо другие средства сбора входных данных, кроме бумажных первичных документов и перфокарт. Однако в качестве носителей небольших файлов допускается использование сменных магнитных дисков.

Наконец, отметим, что технологический процесс ввода и накопления данных должен обеспечить возможность удобного восстановления главного файла.

Программное обеспечение технологического процесса строится на основе операционной системы ДОС/ЕС.

Все вышеперечисленные условия, которым должен удовлетворять технологический процесс ввода и накопления данных, одновременно выделяют некоторый класс АСОИ. Представителями этого класса являются, например, системы коммунального обслуживания, системы учета населения, системы обработки информации в сфере пенсионного обеспечения и т. п., даже если в качестве основного носителя хранимых данных выбираются не магнитные ленты, а постоянные диски.

В качестве основного критерия при выборе из множества технологических процессов, удовлетворяющих перечисленным требованиям, мы примем производительность процесса, понимая под этим отношение объема накопленных в главном файле данных (информации) к общему времени работы технологического процесса, потребовавшемуся для этого накопления.

Известно, что наивысшая производительность процесса ввода достигается при пакетном режиме ввода. Этот режим

заключается в том, что данные в большом количестве предварительно загружаются на быстрое периферийное устройство, и затем без вмешательства оператора происходит их ввод и обработка до полного исчерпания данных. Поскольку процесс собственно ввода данных в ЭВМ является центральным этапом рассматриваемого технологического процесса, желательно построить весь технологический процесс так, чтобы обеспечить на центральном этапе именно пакетный режим ввода данных. Тем самым при прочих равных, или почти равных условиях будет обеспечена максимальная производительность всего технологического процесса.

Итак, описываемый далее технологический процесс можно кратко определить, как процесс корректировки большого информационного файла в режиме пакетного ввода данных.

2. Основные понятия

Пакетом ввода называется совокупность перфокарт (перфоносителей) на одно решение задачи ввода. Здесь под решением задачи ввода понимается реализация некоторой последовательности программных процедур, в результате чего данные из пакета ввода будут перенесены на магнитный носитель - ленту или диск.

Пакет ввода состоит из входных файлов, состоящих, в свою очередь, из комплектов перфокарт. Обычно пакет ввода состоит из одного файла; несколько файлов допускается, если входные файлы определяются в нескольких программах задачи ввода.

Комплект перфокарт представляет собой совокупность перфокарт, соответствующих одному комплекту входных документов. На каждой перфокарте комплекта располагаются данные двух видов: технологические и логические. Технологические данные служат для идентификации перфокарт комплекта, связи комплекта перфокарт с комплектом входных документов и связи с данными на магнитном носителе. С помощью

этих данных организуется технологический процесс контроля ввода и коррекции данных с целью устранения ошибок коррекции и ввода. Логические данные, или собственно данные - это та информация, которая должна быть накоплена или использована для коррекции ранее накопленной информации главного файла.

Отметим, что идентификация данных имеет два важных аспекта: следует убедиться, что данные относятся к обрабатываемому комплекту перфокарт, а с другой стороны, следует опознать логические данные. Последнее достигается применением макетной структуры перфокарт.

Комплект входных документов - эта определенным образом оформленная совокупность входных документов. Входной документ представляет собой унифицированную форму, подготовленную и приспособленную для перфорации. Содержание входного документа соответствует содержанию первичного документа, однако некоторые описательные реквизиты последнего могут не отражаться во входном документе, другие - заменены шифрами.

Совокупность форм входных документов считается точно определенной, в то время как множество первичных документов не считается фиксированным.

3. Схема технологического процесса.

Рассмотрим теперь в целом схему технологического процесса сбора, ввода и обработки данных в пакетном режиме, считая конечной целью этого процесса корректировку главного файла.

Схема технологического процесса изображена на рис. I.

Технологический процесс начинается с обработки первичного документа; обработка заключается в создании входного документа и объединении входных документов в комплект входных документов (КВД). Комплект входных документов поступа-

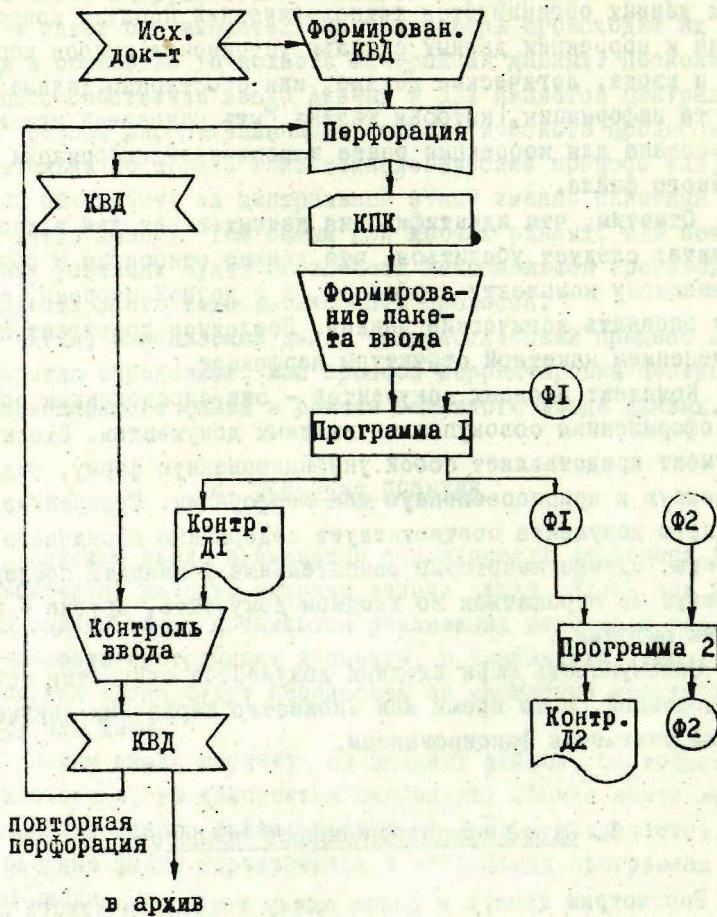


Рис. 1. Схема технологического процесса.

ет на перфорацию, в результате чего создается комплект перфокарт (КПК). Комплекты перфокарт собираются в пакет ввода для обработки его программой I. В результате этого создается новое поколение файла ФІ - файла документов. Каждая запись файла ФІ представляет собой копию входного документа; кроме того, запись содержит некоторые технологические реквизиты для управления процессом ввода данных. Сортированность файла соответствует расположению входных документов в комплекте входных документов, следовательно, запись может быть при необходимости откорректирована информацией, несущей в качестве технологических реквизитов тот же шифр комплекта и тот же шифр документа.

Одновременно с созданием нового поколения файла ФІ задача выдает комплект контрольных табулограмм ДІ.

Контрольные табулограммы используются для исправления ошибок перфорации и ввода в ЭВМ путем визуальной сверки табулограмм с входными документами. Поскольку эта работа весьма трудоемка, программа I проводит возможно более полный синтаксический контроль данных и диагностику ошибок. Здесь могут быть применены всевозможные и весьма сложные методы распознавания данных и их синтаксического анализа, в частности, использование заранее заготовленных описаний форм входных документов.

Контроль ввода с использованием ДІ завершается передачей обрабатываемого комплекта входных документов на повторную выборочную перфорацию отдельных документов или их реквизитов. После выборочной перфорации снова создается комплект перфокарт, входящих в очередной пакет ввода задачи I. Затем снова производится сверка контрольных табулограмм с входными документами, и если требуется, комплект опять отдается на повторную перфорацию.

Описанный процесс повторяется до тех пор, пока контроль ввода не подтверждает точного соответствия накопленных в файле ФІ данных входным документам обрабатываемого

комплекта. После этого выполняется программа 2, цель которой - непосредственная коррекция "главного" файла Ф2. При этом производится "раскомплектация" данных, поскольку сортированность файла Ф2 обычно отличается от сортированности файла Ф1. В самом деле, как отмечалось, сортированность файла Ф1 соответствует структуре комплекта входных документов, т.е. вообще говоря, случайной комбинации данных, в то время как организация файла Ф1 должна соответствовать целевым функциям АСОИ.

Иными словами, структура и состав файла Ф1 определяются потоком входных документов, а структура и состав файла Ф2 - потоком выходных документов АСОИ; как правило, не существует однозначного соответствия между этими потоками в сколь-нибудь сложных АСОИ.

4. Управление технологией

Из изложенного ясно, что описанный технологический процесс в значительной степени автоматический, он не требует прямого вмешательства как на отдельных этапах, так и для перехода от одного этапа к другому. Все этапы технологического процесса не должны быть синхронизированы, все операции - ручные и автоматические - являются унифицированными. Остановимся подробнее на стыковке между этапом окончания контроля ввода комплекта входных документов и этапом обработки комплекта программой 2. Здесь используется механизм дат, сущность которого заключается в следующем.

Каждая запись файла Ф1 имеет отметку о дате ее создания и отметку о готовности комплекта к дальнейшей обработке. В программе 2 рассматриваются только те записи файла Ф1, дата создания которых больше даты создания входного файла Ф2, остальные записи файла Ф1 игнорируются. С

другой стороны, анализируется дата "старой" записи входного файла Ф1, и если с момента готовности этой записи прошел установленный срок, запись "выбрасывается", т.е. не переносится в выходное поколение файла Ф1.

Благодаря такому механизму обеспечивается периодическое самоочищение файла Ф1 и блокируется повторное использование одной и той же записи для корректировки файла Ф2.

5. Обсуждение

Рассмотрение изложенного технологического процесса в свете тех классификационных признаков АСОИ, которые были отмечены в п. I, позволяет сделать следующие выводы.

I. Технологический процесс не требует частого обращения к файлу Ф2, в то же время обеспечивает массовую коррекцию записей этого файла при одном выполнении программы 2. Это обстоятельство является очень важным для существенно больших файлов, занимающих несколько (десятки) томов магнитных лент.

2. Ввод данных из одного комплекта входных документов не требует коррекции комплекта перфокарт при обнаружении ошибок перфорации, а сводится к многократному пропуску отдельных комплектов перфокарт. Очевидно, что использование режима коррекции комплекта перфокарт при постоянной вероятности ошибок перфорации и ввода для достаточного большого комплекта может привести к "зацикливанию": никогда не удастся ввести правильно комплект входных документов.

3. Разнообразие форм входных документов никак не отражается в структуре технологического процесса, что позволяет полностью унифицировать все операции по обработке комплекта, независимо от его конкретного содержания.

4. Использование принципа комплектации и пакетирования существенно упрощает взаимодействие между функциональными службами и позволяет легко проследивать движение данных по процессу.

5. Технологический процесс построен на принципе смены поколений файлов, обеспечивающем простоту и единообразие процедур восстановления информации. В самом деле, при порче одного из поколений файла Ф1 достаточно выполнить программу 1, взяв предыдущее поколение файла Ф1 в качестве входного и собрав в пакет ввода все комплекты перфокарт, полученные при всех сеансах перфорации одного и того же комплекта входных документов. При этом легко построить программу 1 так, чтобы последовательность комплектов в пакете не играла никакой роли.

При порче поколения файла Ф2 достаточно выполнить программу 2 с предыдущим поколением этого файла: механизм дат обеспечивает автоматическое создание полного поколения файла Ф2.

Таким образом, в целом можно считать, что описанная технология решает проблему организации эффективного ввода и накопления данных для описанного в п.1 класса АСОМ.

Автор выражает глубокую признательность Д.Н.Поспелову и Г.В.Лучкину за плодотворное обсуждение основных положений настоящей статьи.

ОСОБЕННОСТИ РАБОТЫ С ИНДЕКСНО-ПОСЛЕДОВАТЕЛЬНЫМИ ФАЙЛАМИ В ДОС/ЕС

С.А.Иванова

І. Введение

Файлы в системе ДОС/ЕС по способу организации данных, которые их составляют, делятся на три категории:

- последовательный файл;
- региональный файл;
- индексно-последовательный файл.

К записям данных файла, организованного первым и вторым способом, осуществляется соответственно последовательный метод доступа и прямой (произвольный). Третий способ организации - индексно-последовательный, допускает оба метода доступа к данным. Такой файл может быть создан только на устройстве прямого доступа - пакете дисков.

Индексно-последовательная организация файла предполагает наличие в записях данных некоторой вспомогательной информации, называемой ключом. При первоначальной загрузке файл создается как последовательный из предварительно рас-
сортированных в порядке возрастания ключа записей. Одновременно с этим система создает в пределах файла особую информационную надстройку, называемую индексами. Эти индексы и обеспечивают быструю обработку записей файла при прямом методе доступа. Кроме того, по желанию пользователя, может быть предусмотрена отдельная область файла (область переполнения), используемая при добавлении новых записей, что позволяет обходиться без реорганизации файла. Однако, периодически ее все-таки следует производить с тем, чтобы упорядочить записи, попавшие в область переполнения и сократить время поиска этих записей.

Как уже отмечалось выше, индексно-последовательный способ организации допускает последовательный метод доступа к данным, что является удобным при просмотре всего файла или некоторого его участка. Прямой метод доступа обеспечивает удобную выборку, обновление и добавление записей в файл в произвольном порядке.

2. Формат записи данных индексно-последовательного файла

Вначале остановимся подробнее на формате записей данных, из которых создается файл. Как уже отмечалось, записи данных должны сопровождаться управляющей информацией, называемой ключом записи. С точки зрения пользователя, ключ - это указатель, позволяющий выбрать из файла или поместить в него нужную запись. И, как правило, в качестве значения ключа выбирается некоторая информация такого рода, как, например, номер платежа, или порядковый номер в списке инвентаризации и т.п., входящие в состав данных. Для системы же ключи записей содержат информацию, необходимую при построении индексов, позволяющих осуществить прямой метод доступа к записям файла. Значение ключа система размещает в специальной области ключа, данные в области данных. Кроме того, имеется еще некоторая служебная информация. Таким образом, внутренний формат записи индексно-последовательного файла имеет вид:

Маркер адреса	Обл-ть счетчика	Обл-ть ключа	Область данных
------------------	--------------------	-----------------	----------------

Рис. I. Формат записи

При первоначальном создании файла, загружаемые записи должны быть расположены в порядке возрастания ключей (упорядоченность понимается в смысле алфавита ЕС). Ключи

всех записей должны быть одинаковой длины и содержать не более 255 знаков. Во время загрузки одновременно с помещением записей в файл система строит индекс. Записи допускаются только фиксированной длины и могут быть сброкированными и несброкированными. То есть в одну физическую запись могут входить несколько логических записей или соответственно одна логическая запись. При создании файла в целях экономии памяти значение ключа можно не вводить в состав области данных, так как это значение все равно будет содержаться в области ключа. Однако для сброкированных записей ключ каждой логической записи должен обязательно содержаться внутри нее, так как в область ключа помещается значение ключа последней записи блока. и ключи всех записей должны располагаться с одной и той же позиции.

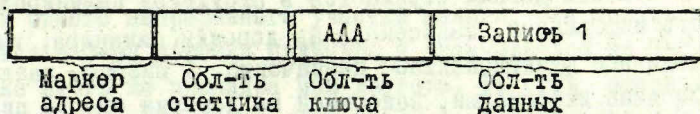


Рис.2. Формат несброкированной записи

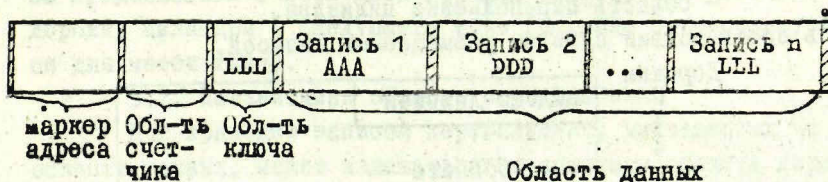


Рис.3. Формат сброкированных записей

При блокировании записей более эффективно используется память за счет экономии места для служебной информации. Кроме того, при последовательной обработке экономится время обмена. Однако, при произвольной обработке сброкированных записей время затрачивается на передачу в память целого блока.

3. Структура индексно-последовательного файла

В целом индексно-последовательный файл, организованный на пакете дисков, может состоять из трех областей:

- индексной области,
- основной области данных,
- независимой области переполнения.

Первые две области должны быть обязательно, а третья создается по желанию пользователя. Участки памяти, занимаемые каждой из областей, описываются своим управляющим оператором EXTENT.

3.1. Основная область данных.

Основная область данных может занимать один или несколько цилиндров пакета дисков. При необходимости, она может располагаться на нескольких пакетах. Состав основной области данных отражается в структуре цилиндров. То есть информация, занесенная на дорожки цилиндра, имеет различное функциональное назначение. А именно, различают три типа информации, заносимой на дорожки одного цилиндра:

- индекс дорожек,
- область данных,
- область переполнения цилиндра,

в соответствии с видом размещаемых записей.

Дорожки

0
1
2
3
4
5
6
7
8
9

индекс дорожек
Область данных
Область переполнения цилиндра

Рис.4. Цилиндр основной области данных

Область данных - это участок памяти, куда помещаются записи данных при создании файла.

Индекс дорожек - часть информационной надстройки файла, упоминавшейся выше - формируется системой по мере загрузки области данных и отражает размещение записей по дорожкам этой области. (Подробнее см. ДОС/ЕС, кн.28 ЕЮ.132.032 Д2).

По желанию пользователя для каждого цилиндра может быть определена область переполнения цилиндра, которая занимает целое число дорожек. Эта область резервируется для записей переполнения, т.е. тех записей, которые вытесняются с дорожек области данных при добавлении новых записей в файл.

При размещении на пакете дисков основной области данных индексно-последовательного файла необходимо соблюдать следующее правило: эта область должна содержаться внутри одного непрерывного участка памяти. Она должна начинаться с 0-й дорожки цилиндра и оканчиваться на 9-й дорожке этого же цилиндра или другого. То есть основная область данных не может начинаться или заканчиваться посреди цилиндра. Для многотомного файла основная область должна продолжаться с последней дорожки одного тома на 0-ую дорожку цилиндра I следующего тома (цилиндр 0 резервируется для меток тома).

3.2. Независимая область переполнения.

Для хранения записей переполнения, вытесненных из области данных, может задаваться независимая область переполнения, под которую отводится непрерывный участок памяти из нескольких цилиндров. Такая область может использоваться для хранения всех записей переполнения или только тех, которые не разместились в областях переполнения цилиндров основной области данных. Такое совместное использование двух типов областей переполнения часто используется.

Записи в областях переполнения отличаются по формату от записей области данных. Во-первых, эти записи всегда несблокированные независимо от того, какими были основные записи файла. Во-вторых, к ним добавляется служебное поле, называемое полем связи, для указания на следующую запись в цепочке переполнения, относящуюся к этой же дорожке области данных.

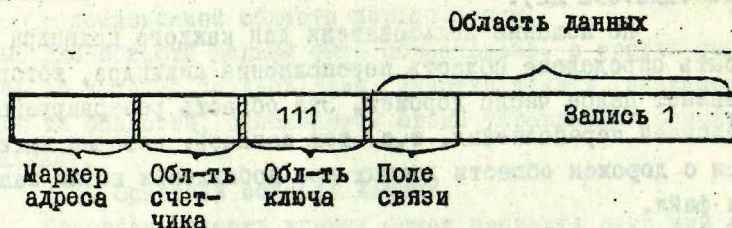


Рис.5. Формат записи переполнения

Независимая область переполнения задается по желанию пользователя посредством оператора `EXTENT`.

Эта область может находиться на том же томе, что и основная область данных или на другом, но в пределах одного тома.

3.3. Индексная область

В этой области система формирует индексы более высокого уровня, чем индекс дорожек. Здесь размещается индекс цилиндров, содержащий информацию о расположении записей данных по цилиндрам файла. По желанию пользователя в этой области может быть создан индекс еще более высокого уровня - главный индекс, отражающий заполненность индекса цилиндров.

Для каждого из этих индексов описывается отдельно участок памяти посредством своего оператора `EXTENT`. Участок главного индекса должен непосредственно предшествовать участку индекса цилиндров. Эти участки должны быть непрерывными и находиться на одном томе.

Таким образом, имеется три типа индексов разного уровня:

- индекс дорожек,
- индекс цилиндров,
- главный индекс.

Первые два типа индексов всегда должны входить в состав файла. Главный индекс может быть создан по желанию пользователя, что рекомендуется делать только для очень больших файлов (для которых участок индекса цилиндров занимает более 4-х дорожек).

Каждый из индексов представляет собой цепочку записей. Отдельная запись содержит информацию об одной дорожке цилиндра или одном цилиндре. Структура записи следующая:

- область ключа, содержащая ключ последней записи на дорожке или цилиндре;
- область данных, содержащая адрес дорожки

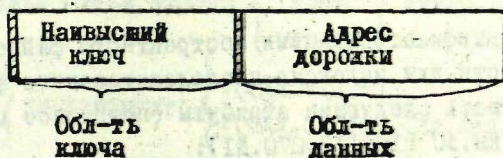


Рис.6. Формат индексной записи

Различие уровней индексов означает следующее: когда по значению ключа запрашивается или помещается в файл некоторая запись, то для определения ее местонахождения сначала просматривается главный индекс (если он был построен), затем индекс цилиндров, индекс дорожек и, наконец, дорожка, на которой должна располагаться эта запись. Таким образом, информация, содержащаяся в индексах, ускоряет просмотр, а использование индексов разных уровней позволяет сузить просмотр посредством указания низшего индекса, область действия которого содержит значение ключа обрабатываемой записи. (Подробнее об индексах см. ДОС/ЕС. кн.28)

ЕЮ.132.032 Д2).

4. Описание индексно-последовательного файла на языке PL/1

В предыдущих разделах была кратко описана внутренняя структура индексно-последовательного файла, организованного на пакете дисков. Теперь будут описаны средства языка PL/1, позволяющие создавать и эксплуатировать файлы этого типа. Вначале рассмотрим описание индексно-последовательного файла, которое необходимо в программе пользователя.

Как известно, вся информация о конкретном файле на языке PL/1 задается для операционной системы как последовательность атрибутов и режимов в операторе описания DECLARE, имеющим формат:

DECLARE<идентификатор>FILE<атрибуты>[,<идентиф.>FILE
<атриб.>,...]

где <идентификатор> - имя абстрактного файла (≤ 6 знаков). В частности для индексно-последовательного файла необходимо указывать следующие атрибуты (подробное описание см. ДОС/ЕС. кн.30 ЕЮ.132.070.Д1):

- 1) RECORD - способ обработки файла;
- 2) INPUT/OUTPUT/UPDATE - функции файла, соответственно вводного, выводного и обновляемого (т.е. записи извлекаются для обработки и помещаются обратно, или добавляются в файл);
- 3) SEQUENTIAL/DIRECT - метод доступа последовательный и прямой соответственно (по умолчанию SEQUENTIAL);
- 4) KEYED - доступ к записям по значению ключа.

В составе атрибута ENVIRONMENT должны указываться следующие режимы:

- а) MEDIUM(SYSxxx,nnnn) - устройство;
- б) F(<длина записи>) или F(<длина блока>,<длина записи>)

c) INDEXED

d) KEYLENGTH(n) ($1 \leq n \leq 255$) - длина ключа

e) EXTENTNUMBER(n) ($2 \leq n \leq 255$) - число участков, занимаемых файлом

f) OFLTRACKS(n) ($0 \leq n \leq 8$) - число дорожек переполнения цилиндра

Кроме того, еще могут быть указаны режимы INDEXMULTIPLE - для создания главного индекса, и KEYLOC(n) - для блокированных записей, указывающий, с какого байта записи располагается ключ. Все атрибуты и режимы оператора DECLARE сведены в табл. I.

Таким образом, записать формат оператора DECLARE для индексно-последовательного файла можно в следующем виде (понимая атрибуты в скобках {}, как альтернативные, а в [], как аддитивные).

$$\text{DECLARE} \langle \text{имя файла} \rangle \text{ FILE RECORD } \left\{ \begin{array}{l} \text{OUTPUT} \\ \text{INPUT} \\ \text{UPDATE} \end{array} \right\} \text{ KEYED}$$

$$\left\{ \begin{array}{l} [\text{SEQUENTIAL}] \\ \text{DIRECT} \end{array} \right\} \text{ ENVIRONMENT} \left(\left\{ \begin{array}{l} F(\langle \text{дл. зап.} \rangle) \\ F(\langle \text{дл. блока}, \langle \text{дл. зап.} \rangle) \end{array} \right\} \right)$$

MEDIUM(SYSxxx,nnnn) INDEXED EXTENTNUMBER(n)
KEYLENGTH(n) [OFLTRACKS(n)] [INDEXMULTIPLE] [KEYLOC(n)])

Следует отметить, что после атрибута FILE порядок остальных атрибутов и режимов может быть произвольным.

Пример I.

```
DECLARE DISK FILE RECORD KEYED OUTPUT ENVIRONMENT
  (F(240,80) INDEXED KEYLENGTH(3) KEYLOC(1)
  EXTENTNUMBER(2) MEDIUM (SYS004,5052))
```

Описан выводной индексно-последовательный файл с именем DISK, занимающий два участка памяти, состоящий из сблокированных по три логических записей длиной 80 байтов. Ключ, длиной в три байта, располагается с I-го байта записи.

Таблица I. Атрибуты и режимы оператора DECLARE
для индексно-последовательного файла

Обозначения: X - обязательно указание

S - присутствие по умолчанию

A - выбор из указанных возможностей

O - возможно применение

Атрибуты и режимы	Файл с последовательным доступом			Файл с прямым доступом	
	Вводной	Выводной	Обновля- емый	Вводной	Обновля- емый
<имя файла> FILE	X	X	X	X	X
RECORD	X	X	X	X	X
INPUT	X			X	
OUTPUT		X			
UPDATE			X		X
SEQUENTIAL	S	S	S		
DIRECT				X	X
KEYED	X	X	X	X	X
BUFFERED	S	S	S		
ENVIRONMENT	X	X	X	X	X
MEDIUM(	X	X	X	X	X
SYSnnnn, dddd	X	X	X	X	X
F(<дл.блока>)	A	A	A	A	A
F(<дл.блока>, <дл.зап.>)	A	A	A	A	A
VERIFY		O	O		O
INDEXED	X	X	X	X	X

KEYLENGTH	X	X	X	X	X
EXTENTNUMBER	X	X	X	X	X
INDEXMULTIPLE	0	0	0	0	0
OFLTRACKS		0			0
KEYLOC	0	0	0	0	0

5. Операторы программы "УПРАВЛЕНИЕ ЗАДАНИЯМИ" для индексно-последовательного файла

В этом разделе будут перечислены управляющие операторы, которые необходимы при работе с индексно-последовательным файлом, и указан их порядок. Подробное описание функций и формата этих операторов приводится в материалах ДОС/ЕС кн.6 ЕЮ.132.017 Д1.

Для индексно-последовательного файла первым специальным оператором перед вызовом РЕДАКТОРА должен указываться оператор резервирования памяти для обработки меток файла

//_ LBUTYP _ NSD(nn)

где nn - соответствует количеству участков, занимаемых файлом.

Затем следует оператор вызова РЕДАКТОРА СВЯЗЕЙ

//_ EXES ± LINKEDT

после которого указывается оператор, сообщающий информацию для меток файла.

//_ DLVL _ <имя файла> , ['<идентификатор файла>'] , [<дата>] , <код>

Код для индексно-последовательного файла может иметь два вида: ISC - для создания файла, ISE - для обработки файла.

Непосредственно за этим должны следовать операторы, описывающие участки файла.

//_ EXTENT _ [SYSxxx] , [<регистр.номер тома>] , <тип> , <номер уч-ка> , <адрес уч-ка> , <кол-во дорожек>

<тип> - цифры, определяющие тип участка: 4 - участок для индексной области, 1 - для размещения данных, 2 - для независимой области переполнения.

<номер участка> - десятичное число от 0 до 255, определяющее порядковый номер участка. Участок главного индекса должен иметь номер 0. Если файл не имеет главного индекса, то первый участок файла, занимаемый индексом цилиндров, должен иметь номер 1.

<адрес участка> - от 1 до 5 цифр, определяющих относительный номер первой дорожки участка, вычисляемый по формуле

$$10 \times \text{Нцил.} + \text{Ндор.}$$

Порядок следования операторов EXTENT такой: первым указывается EXTENT главного индекса (если он строится), затем EXTENT индекса цилиндров и последними - EXTENT основной области данных и, возможно, независимой области переполнения.

Пример 2.

Задание на трансляцию, редактирование и выполнение программы, обрабатывающей индексно-последовательный файл может быть составлено следующим образом:

```
// JOB PRIMER
```

```
// OPTION LINK
```

```
// ASSIGN SYS004,X'191'
```

```
// EXEC PL/I
```

[Программа]

```
/x
```

```
// LBLTYP NSD(02)
```

```
// EXEC LINKEDT
```

```
// DLBL DISK,,0,ISE
```

```
// EXTENT SYS004,MSTAT,4,1,850,10
```

```
// EXTENT,,1,2,860,10
```

// ЕХЕС

[Данные]

/ж

/2

Файл расположен на одном томе, имеющем регистрационный номер МСТАТ, занимает два участка: один цилиндр под индекс цилиндров и один - под основную область данных.

6. Операции с индексно-последовательным файлом

Операции, которые проводятся с индексно-последовательными файлами средствами языка PL/1, можно разделить на пять видов:

- создание файла;
- расширение файла;
- чтение записей из файла;
- добавление записей в файл;
- обновление записей файла.

Каждый из видов операций требует описания файла с теми или иными атрибутами и применения соответствующих операторов чтения или записи в программе пользователя. К тому же следует заметить, что в программе необходимо, по крайней мере один раз, открыть файл в начале обработки и закрыть в конце. Далее рассмотрим подробнее каждый из видов операций с файлом.

6.1. Создание файла

Как уже отмечалось раньше, индексно-последовательный файл создается на пакете дисков на отведенных участках из предварительно рассортированных в порядке возрастания значения ключа записей. И создается он как последовательный файл, т.е. с атрибутом SEQUENTIAL. Очевидно, что при создании файл на пакете дисков является выходным, поэтому в описании указывается атрибут OUTPUT. Непосредственно зане-

сение каждой записи в файл осуществляется оператором
 WRITE FILE(<имя файла>)FROM(<идентиф.>)KEYFROM(<выражение>)
 где <идентификатор> - имя простой переменной, имя массива или старшей структуры (имена с атрибутами DEFINED не допускаются);
 <выражение> - скалярное выражение, значение которого должно преобразовываться в строку знаков.

действие оператора - передача содержимого указанной переменной или структуры в область данных записи файла и значения выражения в область ключа записи. Таким образом формируется одна запись файла и одновременно с этим системой строятся индексы.

Кроме того, следует отметить, что в управляющем операторе DLVL должен быть указан тип - ISC.

Для создания файла пользователем составляется на PL/1 отдельная программа.

Пример 3.

На пакете дисков создается индексно-последовательный файл под именем DISK. Вводный файл (KARTE) находится на картах, и его записи упорядочены. Ключ каждой записи содержит 3 символа.

```

PROG1: PROCEDURE OPTIONS (MAIN);
  DECLARE KARTE FILE RECORD INPUT ENVIRONMENT
    (F(80) MEDIUM (SYSIPT,6012) CONSECUTIVE);
  DECLARE DISK FILE RECORD KEYED OUTPUT ENVIRONMENT
    (F(80) INDEXED KEYLENGTH(3) EXTENTNUMBER(2)
    OFLTRACKS(2) MEDIUM (SYS004,5052));
  DECLARE 1 SZAP,2(KLUCH CHARACTER(3),INF CHARACTER
    (77));
  ON ENDFILE (KARTE) GO TO A1;
  OPEN FILE (KARTE), FILE (DISK);
A2:  READ FILE (KARTE) INTO (SZAP);
  WRITE FILE (DISK) FROM (SZAP) KEYFROM (KLUCH);
  GO TO A2;
```

```
A1:  CLOSE FILE (DISK);
      END PROG1;
```

Заметим, в частности, что оператор `DIVL` программы "УПРАВЛЕНИЕ ЗАДАНИЯМИ" имеет вид:

```
// DIVL DISK,, 0,ISC
```

6.2. Расширение файла

Под расширением файла понимается помещение новых записей в конец области данных (если она не была заполнена до конца при создании файла) или расширение границ основной области данных и индекса цилиндров уже созданного файла и помещение новых записей. Причем для этой операции новые записи, как и при создании файла, должны быть упорядочены и ключ первой из новых записей больше ключа последней записи файла. Для помещения новых записей в конец области данных файла может быть использована та же программа, что и для создания файла. Но при этом в управляющем операторе `DIVL` должен указываться код - `ISE`. Изменить верхнюю границу основной области данных и области индекса цилиндра можно, указав новые параметры в соответствующих операторах `EXTENT` (для главного индекса это невозможно). Тем самым увеличивается файл.

6.3. Чтение записей

Чтение записей из индексно-последовательного файла осуществляется двумя методами - последовательным (атрибут `SEQUENTIAL`) и прямым (атрибут `DIRECT`). Если программа осуществляет только извлечение записей, то файл может быть описан как вводной, т.е. с атрибутом `INPUT`.

Соответственно методу доступа к записям при чтении используется оператор `READ` того или иного формата. Для последовательного извлечения записей может быть использован любой из трех форматов в зависимости от нужд пользователя.

Формат первый:

```
READ FILE(<имя файла>)INTO(<идентиф.>)KEY(<выражение>)
```


(Значения параметров описаны в предыдущем параграфе).

Действие оператора: передача информации из области данных записи в память; значение выражения указывает ключ записи, которая должна быть прочитана.

формат второй:

READ FILE(<имя файла>) INTO(<идентификатор>)

Этот оператор может использоваться для прочтения последовательно всего файла без указания значения ключей (каждый оператор читает только одну запись). Если необходимо прочитать последовательно только часть файла, то ключ начальной записи указывается в режиме KEY оператора первого формата, и эта запись читается этим оператором, а остальные записи могут быть прочитаны посредством оператора READ второго формата.

Формат третий:

READ FILE(<имя файла>) INTO(<идентиф.>) KEYTO(<идентиф.>)

Действие оператора: передается содержимое как области данных, так и области ключа записи в основную память, отведенную переменным, указанным соответственно в режимах INTO и KEYTO.

Для прямого метода доступа применяется только первый формат оператора READ.

В операторе DLVL должен быть указан код - ISE.

Пример 4.

Читаются последовательно и печатаются записи файла (DISK), созданного в примере 3, в соответствии со значением ключей, которые следуют в возрастающем порядке, но не обязательно все подряд. Эти значения ключей (KL) пробиты и вводятся с п/к и занимают первые три позиции из каждой десяти на одной п/к. Признаком конца этого набора п/к служит последовательность символов ***.

```

PROG2: PROCEDURE OPTIONS(MAIN);
  DECLARE DISK FILE RECORD KEYED INPUT ENVIRONMENT
  (MEDIUM(SYS004,5052)F(80)INDEXED KEYLENGTH(3)
  EXTENTNUMBER(2));
  DECLARE INFOR CHARACTER(80),KL CHARACTER(3);
  OPEN FILE (DISK);
A1:   GET EDIT (KL) (A(10));
      IF KL='END' THEN GO TO EOP;
      READ FILE (DISK) INTO (INFOR) KEY (KL);
      PUT EDIT (INFOR) (SKIP,A(80)); GO TO A1;
EOP:  CLOSE FILE (DISK);
      END PROG2;

```

6.4. Добавление записей.

После того, как индексно-последовательный файл создан, может возникнуть необходимость во внесении новых записей, значения ключей которых попадают между ключами записей существующего файла. В этом случае система обеспечивает помещение новой записи в соответствующее место файла согласно значению ключа. Определение местоположения этой записи производится следующим образом.

Вначале просматриваются индексы, начиная с индекса высшего уровня. Просмотр индекса дорожек определяет ту дорожку, на которую нужно поместить добавляемую запись. А именно, запись вставляется на дорожку, у которой последний ключ выше ключа добавляемой записи, и тогда последняя запись дорожки вытесняется в область переполнения. Может оказаться, что по значению ключа новая запись оказывается как бы между дорожками, в этом случае она помещается в область переполнения, но содержит в поле связи признак принадлежности к первой из этих дорожек.

Выполнение этой операции сопровождается корректировкой информации в соответствующих индексах. Подробное описание внутреннего механизма этой процедуры приведено в материалах ДОС/ЕС. кн.28.ЕЮ.132.032 Д2. и кн.30.ЕЮ.132.070 Д1.

Операция добавления записей в созданный файл осуществляется прямым методом доступа (т.е. указываются атрибуты DIRECT и UPDATE) посредством оператора
`WRITE FILE(<имя файла>)FROM(<идентиф.>)KEYFROM(<выражение>)`
 Один такой оператор выполняет вставку одной записи. Если окажется, что ключ добавляемой записи старше самого большого ключа файла, то запись помещается или в конец области данных, если она неполная, или же в область переполнения.

6.5. Обновление записей.

Для обновления записей файла, т.е. замены старых записей новыми, содержащими некоторую новую информацию в области данных, могут быть использованы оба метода доступа.

При необходимости заменить большое количество записей без добавления новых наиболее эффективен последовательный метод доступа (атрибуты UPDATE и SEQUENTIAL). При этом новые записи должны располагаться в логически возрастающей последовательности ключей. Перед заменой запись обязательно должна быть извлечена из файла, т.е. прочитана посредством оператора READ (с режимом KEY или без него), а затем, после изменения, внесена обратно в файл с помощью оператора

`REWRITE FILE(<имя файла>)FROM(<идентификатор>).`

Пример 5.

Пусть в файле со значением ключей записей 010, 020, ..., 260 нужно полностью заменить записи с ключами от 060 до 190. Новые записи вводятся с карт.

`PROG3: PROCEDURE OPTIONS(MAIN);`

`DECLARE DISK FILE RECORD KEYED UPDATE ENVIRONMENT
 (P(80)INDEXED KEYLENGTH(3) EXTENTNUMBER(2)
 MEDIUM(SYS004,5052));`

`DECLARE ZAP CHARACTER(80), KLUCH CHARACTER(3);
 OPEN FILE(DISK);`

`READ FILE(DISK) INTO(ZAP) KEY('060');`

```
A1: GET EDIT(ZAP) (A(80));
    REWRITE FILE(DISK) FROM(ZAP);
    READ FILE(DISK) INTO(ZAP) KEYTO(KLUCH);
    IF KLUCH='200' THEN GO TO BOP;
    GO TO A1;
BOP: CLOSE FILE(DISK);
    END PROG3;
```

при обновлении небольшого числа записей более эффективен прямой метод доступа (атрибуты UPDATE, DIRECT), позволяющий ускорить процесс обработки путем произвольного извлечения записи. В этом случае имеются две возможности проведения обработки записей.

Первая — это полное изменение информации в области данных записи. Эту операцию можно произвести посредством оператора REWRITE следующего формата

```
REWRITE FILE(<имя файла>)FROM(<идентиф.>)KEY(<выражение>)
```

Вторая — когда нужно частично изменить информацию в записи. Очевидно, что в этом случае обновляемая запись должна быть вначале прочитана, затем она изменяется и помещается на прежнее место. Эта операция производится с использованием совместно двух операторов следующего формата

```
READ FILE(<имя файла>)INTO(<идентиф.>)KEY(<выражение>)
REWRITE FILE(<имя файла>)FROM(<идентиф.>)KEY(<выражение>)
```

Пример 6.

В файле изменяются некоторые записи, значения ключей которых вводятся с карт. До ключа '200' записи изменяются полностью, при этом новая информация расположена на тех же картах. Начиная с записи с ключом '200' информация изменяется частично (на место 21, 22, 23 символов ставятся и остальные до 80-го заменяются пробелами).

Признаком конца обработки служит последовательность символов 'жжж'.


```

PROG4: PROCEDURE OPTIONS(MAIN);
  DECLARE DISK FILE RECORD KEYED DIRECT UPDATE
    ENVIRONMENT (F(80)) INDEXED KEYLENGTH(3)
    ENTENTNUMBER(2) MEDIUM(SYS#04,5052));
  DECLARE ZAP CHARACTER(80), KLUCH CHARACTER(3),
    R CHARACTER(20));
  OPEN FILE(DISK);
A1:   GET EDIT(ZAP) (A(80));
      KLUCH=ZAP;
      IF KLUCH='200' THEN GO TO A2;
      REWRITE FILE(DISK) FROM(ZAP) KEY(KLUCH);
      GO TO A1;
A2:   READ FILE(DISK) INTO(ZAP) KEY(KLUCH);
      R=ZAP;
      ZAP=R // 'ЖЖЖ';
      REWRITE FILE(DISK) FROM(ZAP) KEY(KLUCH);
      GET EDIT(ZAP) (A(80));
      KLUCH=ZAP;
      IF KLUCH='ЖЖЖ' THEN GO TO EOP;
      GO TO A2;
EOP:  CLOSE FILE(DISK);
      END PROG4;

```

Все положения относительно операций с файлами, изложенные в параграфе 6, сведены в таблицу 2, которая описывает все допустимые сочетания атрибутов описания и операторов обмена.

**Таблица 2. Атрибуты описания и операторы обмена для операций
с индексно-последовательным файлом**

Операция	Атрибуты оп-ра DECLARE	Оператор обмена	Оператор DLBL
1. Сов- дание	OUTPUT SEQUENTIAL }	WRITE FILE(<имя ф.>)FROM(<идент.>)KEYFROM(<выраж.>)	//DLBL,...,ISC
2. Чте- ние	a) INPUT SEQUENTIAL } b) INPUT DIRECT }	READ FILE(<имя ф.>)INTO(<идент.>)KEYTO(<идент.>) READ FILE(<имя ф.>)INTO(<идент.>)KEY (<выраж.>) READ FILE(<имя ф.>)INTO(<идент.>), READ FILE(<имя ф.>)INTO(<идент.>)KEY(<выраж.>)	//DLBL,...,ISE
3. Распи- сание	OUTPUT SEQUENTIAL }	WRITE FILE(<имя ф.>)FROM(<идент.>)KEYFROM(<выраж.> ,	//DLBL,...,ISE
4. Добав- ление	UPDATE DIRECT }	WRITE FILE(<имя ф.>)FROM(<идент.>)KEYFROM(<выраж.>)	//DLBL,...,ISE
5. Обнов- ление	a) UPDATE SEQUENTIAL } b) UPDATE DIRECT }	READ FILE(<имя ф.>)INTO(<идент.>)KEY(<выраж.>) READ FILE(<имя ф.>)INTO(<идент.>) REWRITE FILE(<имя ф.>)FROM(<идент.>) REWRITE FILE(<имя ф.>)FROM(<идент.>)KEY(<выраж.>) READ FILE(<имя ф.>)INTO(<идент.>)KEY(<выраж.>) REWRITE FILE(<имя ф.>)FROM(<идент.>)KEY(<выраж.>)	//DLBL,...,ISE

7. Реорганизация файла

Как уже отмечалось, индексно-последовательная организация представляет удобную возможность добавлять новые записи в файл без его перезаписи. Однако, периодически все-таки реорганизация должна производиться по следующим причинам: добавления увеличили время поиска записей; в основной области данных накопилось много ненужных записей; заполнилась область переполнения.

Периодичность реорганизации зависит от изменчивости файла, от требований пользователя относительно времени доступа. Реорганизовать файл можно двумя способами. Первый - файл последовательно переписывается на другой участок или на другой носитель для того, чтобы, воспользовавшись им как вводным, заново создать файл на первоначальном участке. Второй - файл реорганизуется и помещается на другом участке диска. Информация, на основе которой принимается решение о реорганизации файла, накапливается операционной системой в метке тома файла.

Следует отметить, что ДОС/ЕС не имеет средств для пометки и удаления ненужных записей из файла. Поэтому пользователь должен производить эту операцию своими программными средствами. Помеченные записи физически могут быть удалены из файла только при его реорганизации.

8. Заключение

В завершение можно отметить, что индексно-последовательная организация файла данных является довольно удобной, так как допускает два метода доступа к данным - прямой и последовательный. Кроме того, выполнение всех операций с записями файла обеспечивается системой программирования, что значительно упрощает его эксплуатацию пользователем.

РАСШИРЕНИЕ ВОЗМОЖНОСТЕЙ ДИСКОВОЙ
ОПЕРАЦИОННОЙ СИСТЕМЫ ПРИ ПРОГРАММИРОВАНИИ
НА PL /I И АССЕМБЛЕРЕ

Х.И. Килов

I. Обработка стандартных ленточных меток
в PL -программах

Как известно, в подмножестве PL /I [I] невозможно непосредственная обработка стандартных меток файла на магнитной ленте. Проверку таких меток производит системная программа открытия файла, находящаяся в транзитной области и недоступная программисту, и поэтому он в проблемной программе лишен возможности непосредственного доступа к содержимому меточных полей и не может принять решение, отличное от стандартного.

Тем не менее, это может быть сделано с помощью предлагаемой подпрограммы на PL /I. Именно, стандартные метки тома VOL1 и файла HDR1 могут быть считаны как первые две записи длиной 80 байтов (вспомогательного) ленточного файла без меток. Затем вспомогательный файл без меток закрывается, что вызовет автоматическую перемотку ленты на начало (тома и) обрабатываемого файла. Полученная метка файла может быть проверена программистом в проблемной программе, после чего обрабатываемый файл может, если результаты проверки удовлетворяют программиста, открываться и обрабатываться, как обычно.

Описываемая процедура FILAB с параметром - строкой символов длины 80, в которую помещается меточная запись, и реализует указанную возможность. Проблемный программист должен лишь вызвать эту процедуру перед открытием обрабатываемого файла, установив предварительно ленту на начало. При этом необходимо учитывать, что номер

Замечание. При открытии файла на МЛ с помощью оператора OPEN проверяется дата в управляющем операторе // TLBL , если она указана. Эта дата должна совпадать с календарной датой создания файла, иначе на пишущей машинке отпечатается сообщение номер 4133D о неверной метке HDR1 , на которое, впрочем, оператор может ответить IGNORE.

1. Килов Х.И. Подмножество РЛ/I. Справочное руководство по программированию. Рига, 1975, 16 с.

```

      TLY:PROCEDURE OPTIONS(MAIN);
      DECLARE C CHARACTER(80); FILE OUTPUT RECORD ENVIRONMENT
      (MEDIUM(SYS005),S012) F(80);
      OPEN FILE(T);C(18)'ABCDE67890';WRITE FILE(T)FROM(C);
      PUT EDIT('RECORD WRITTEN ',C(11))CLOSE FILE(T);
      END TLY;

```

```
// ASSIGN SYS005,X'281'
// TLGL T,'AN INTERESTING',75/175
```

RECORD WRITTEN ABCDE67890ABCDE6789WABCDUE67890ABCDE67890ABCDE67890ABCDE67890

Проверка метки и считывание файла T:

```

FILAB:PROCEDURE(T);
  DECLARE(S,T) CHARACTER(80);
  A31416 FILE RECORD INPUT ENVIRONMENT(MEDIUM(SYS005,5012)F(50)NOLABEL);
  OPEN FILE(A31416) READ FILE(A31416) INTO(S); READ FILE(A31416) INTO(S);
  CLOSE FILE(A31416);
  T=S;
END FILAB;

```

```

TRY:PROCEDURE OPTIONS(MAIN);
  DECLARE S CHARACTER(80); T FILE INPUT RECORD ENVIRONMENT
    (MEDIUM(SYS005,5012)F(80));
  CALL FILAB(S) PUT EDIT('LABEL= ',S)(A);
  OPEN FILE(T);
  READ FILE(T) INTO(S); PUT SKIP EDIT('RECORD READ= ',S)(A); CLOSE FILE(T);
END TRY;

```

// TLOC T

```

LABEL= MORIAN INTERESTING 11111100010001000101 75000 7517500000000000/20
RECORD READ= ABCDE67090ABCUE67090ABCDE67090ABCDE67090ABCUE67090ABCUE67090

```


2. Программа рандомизации на ассемблере и организация ее вызова из PL -процедуры

Алгоритмы рандомизации реализуются в подмножестве PL/1 [1], как правило, с трудом, так как требуют манипуляций с битами внутри машинного слова. В то же время ассемблер позволяет достаточно просто написать нужные подпрограммы. В работе приводится одна из возможных подпрограмм рандомизации и рассматривается ее вызов из PL-процедуры.

Выбранный алгоритм рандомизации соответствует приведенному для GE-400 в [2]. Происходит преобразование строки символов (поля данных) FIELD длины $LENGTH \leq 20$ в номер страницы (записи в файле) RESULT (в файле содержится N страниц). Поскольку номер страницы будет использован в качестве ключа, он выдается в десятичном распакованном виде (атрибут PICTURE '(8)9' в PL -процедуре). Конкретная реализация алгоритма видна из комментариев в подпрограмме на ассемблере.

Организация вызова подпрограммы на ассемблере из PL -процедуры рассматривается в руководстве для программиста по использованию PL/1. Но, как выяснилось, руководство это не свободно от недостатков. Так, "пролог" подпрограммы на ассемблере в некоторых случаях может содержать лишь одну (макро) команду, и тогда все остальные указания в руководстве команды не нужны. В этой ситуации в качестве регистра базы может использоваться регистр I5, в который вызывающая PL - процедура помещает адрес точки входа подпрограммы на ассемблере. Поэтому в загрузке в (другой) регистр базы содержимого регистра I5 нет необходимости. Далее, хотя руководство и требует, чтобы под-

программа на ассемблере имела свою DSA длиной, по крайней мере, 8 байтов, — на самом деле можно обойтись и без такой DSA. Для подпрограммы, в которой не используются другие подпрограммы и (стандартные) файлы, не происходят прерывания, и не используется регистр I3, DSA можно не создавать. Таким образом, для реализации "пролога" рассматриваемой подпрограммы на ассемблере и возврата из нее достаточно использовать лишь макрокоманды SAVE и RETURN.

ЛИТЕРАТУРА

1. Килов Х.И. Подмножество PL/1. Рига, 1974, 59 с.
2. GE-400. Series IDS COBOL. (Ref. 18.30.207A). General Electric Information Systems, 1968, pp.21-22, 113-114.

```

NATE1PROCEDURE OPTIONS(MAIN)1
DECLARE RESULT PICTURE'(8)9',FIELD CHARACTER(20)1
N#1001FIELD#94A123,7BAG'1
PUT EDIT1' FIELD#',FIELD,' N#',N11A,A,A1F(4)11
DO LENGTH#20 TO 1 BY -61
CALL RANDOM(FIELD,LENGTH,N,RESULT)1
PUT SKIP EDIT1' LENGTH#',LENGTH,' RESULT#',RESULT1(A,F(3),A,F(4))1
CALL DYNDUMP(RESULT)1
END1
END NATE1

```


STMT	SOURCE	STATEMENT	
1	RANDOM	START-8	
2		PRINT NUGEN	
3		USING =1:5	
4		SAVE (1:4,1:2)	
5		L 2:8(1)	FIELD ADDR IN R2
6		MVC FIELD(8(1:2)	
7		L 2:4(1)	LENGTH ADDR IN R2
8		MVC LENGTH(8(1:2)	
9		L 2:LENGTH	
10		BCTR 2:8	
11		ST 2:LENGTH	
12		L 2:8(1)	N ADDR IN R2
13		MVC N(8(1:2)	
14		SH 7:7	RESULT IN R7
15		LM 3:5,INIT	INDEX IN R3
16	LOOP	SH 6:6	
17		IC 6:FIELD(13)	
18		ALR 7:6	
19		LR 6:7	ROTATE
20		SHDL 6:12	RIGHT 12 BITS
21		N 6:RXL4'1'	TESTING
22		BO 8:NOCOMP	LAST BIT
23		X 7:BX'FFFFFFFF'	COMPLEMENT
24	NOCOMP	DXLE 3:4,LOOP	
25		LTR 7:7	
26		BP DIVOIV	
27		X 7:BX'FFFFFFFF'	
28	DIVDIV	C 7:N	
29		BL NODIV	
30		SR 6:6	DIVIDE
31		D 6:N	BY N
32		LR 7:6	
33	NODIV	CVD 7:RES	
34		UNPK RESULT,RES	
35		RVZ RESULT(7(1):BX'FF'	
36		L 2:12(1)	RESULT ADDR IN R2
37		MVC 8(8(1:2):RESULT	RESULT IN ITS PLACE
38		RETURN (1:4,1:2)	
39		DS 80	
40	RESULT	DS 0	
41	RES	DS 0	
42	FIELD	DS 6L28	
43	N	DS 8	
44	INIT	DC F'0'	
45		DC F'1'	
46	LENGTH	DS 8	
47		END	
48		DXL4'1'	
49		DX'FFFFFFFF'	
50		DX'FF'	

// EXEC

FIELD#94A12378AC I N# 100
LENGTH# 20, RESULT# 59
PL/I DYN DUMP
001E30 P0F0F0F0 P0F0F3F0

LENGTH# 14, RESULT# 40
PL/I DYN DUMP
001E30 P0F0F0F0 P0F0F4F0

LENGTH# 8, RESULT# 91
PL/I DYN DUMP
001E30 P0F0F0F0 P0F0F9F1

LENGTH# 2, RESULT# 88
PL/I DYN DUMP
001E30 P0F0F0F0 P0F0F8F0

3. Условное выполнение шагов задания в Дисковой операционной системе

Язык управления заданиями в Дисковой операционной системе не позволяет осуществлять непосредственную связь между различными заданиями или различными шагами одного и того же задания. Такая связь требуется, например, для того, чтобы выполнить тот или иной шаг задания в зависимости от того, удовлетворяется ли некоторое (заданное заранее) условие. Для небольших программ в этом нет необходимости, т.к. все выполняемые подпрограммы могут одновременно размещаться в памяти. Однако, для больших и сложных задач, решение которых требует выполнения нескольких шагов задания, без такой связи не обойтись.

Для реализации программным образом связи между шагами задания используется область связи супервизора, (абсолютный) адрес которой находится в байтах с шестнадцатиричными адресами I4-I7. В области связи имеются байты, информация в которые может быть определенным образом записана. К таким байтам относится байт программных переключателей UPSI, содержимое которого очищается в начале задания.

Извлечь информацию из ячейки с определенным абсолютным адресом ВРЛ - программе можно с помощью использования базированной переменной, указателем которой является нужный адрес. В рассматриваемом случае такой указатель требуется сформировать из содержимого четырех байтов с адресами I4-I7, для чего используется тот же прием (см. процедуру STEP1). Указанный способ, однако, непригоден для занесения информации в область связи,

т.к. при попытке записи происходит прерывание по защите памяти. Чтобы это обойти, требуется написать подпрограмму на ассемблере с использованием макрокоманды `MVSOM` (см. подпрограмму `PUTSOM`). Можно использовать подпрограмму на ассемблере и для извлечения информации из области связи (см. подпрограмму `GETSOM`, вызываемую процедурой `STEP2`).

Ясно, что для проверки определенной комбинации битов в байте `UPSI` достаточно наложить на этот байт (с помощью `DEFINED`) битовую переменную длины 8, и посредством логического умножения затем проверить нужные биты, как и сделано в процедуре `STEP1` для последнего и в процедуре `STEP2` для предпоследнего бита.

Вызов подпрограмм на ассемблере из `PL` - процедур во всех случаях реализован существенно проще, чем указано в руководстве для программиста по использованию `PL/1`. Подробнее этот вызов рассмотрен выше, во второй статье из этого цикла.

```
ROCOM1PROCEDURE OPTIONS(MAIN)
  DECLARE C CHARACTER(1)
  GET EDIT(C)(A(1))PUT EDIT(' UPSI JUST READ: ',C)(A)
  CALL PUTROM1
END ROCOM1
```



```

1 PUTCOM      START
2             PRINT NOGEN
3             USING 0,13
4             SAVE 1(4,12)
5             L     2(0,11)          BYTE ADDR IN R2
6             MVC   BYTE,0(2)
7             MYCOM 23,1,BYTE
8             RETURN 1(4,12)
9
10            BYTE 03      0
11
12            END
13
14            DA(BYTE)

```

// EXEC

UP5: JUST READ: 1

```

STEP1 PROCEDURE OPTIONS(MAIN);
  DECLARE A(24) CHARACTER(1) BASED(P), P POINTER;
  UPSIC CHARACTER(1), UPSIB BIT(8) DEFINED UPSIO;
  C CHARACTER(4), B BIT(32) DEFINED O;
  UNSPEC(P) = '0'B; C(A(21) || A(22) || A(23) || A(24)) UNSPEC(P) = '0';
  /* COMAMEA ADDRESS = DEC, 20 */
  UPS(C(A(24))) /* UPS: ADDR = COMAMEA+23 */
  IF UPSIB & '00000001'B THEN PUT 20(1) ('STEP1 EXECUTED')(A);
  /* L A S T   B I T */ ELSE GOTO END1;
END1 END STEP1;

```

// EXEC

STEP1 EXECUTED

```

STEP2:PROCEDURE OPTIONS(MAIN)
  DECLARE UPSIO CHARACTER(1), UPSID BIT(8) DEFINED UPSIO
  CALL GETCOM(UPSIO) /* ALTERNATIVE * USING ASSEMBLER ROUTINE */
  IF UPSID & '00000010'B THEN PUT EDIT(' STEP2 EXECUTED')|A|
  /* LAST BIT BUT ONE */ ELSE GOTO END2
END2:END STEP2

```

```

1  GETCOM  START
2          PRINT NOGEN
3          USING *118
4          SAVE  (14,12)
5          L     2,0(1)
6          COMRG
7          MVC   0(1,2),25(1)
8          RETURN (14,12)
9          END

```


4. Программа перекодировки дат

При программировании на PL/1 [1] календарная дата, получаемая с помощью встроенной функции DATE, имеет вид строки из 6 символов: YYMMDD. Однако, дата, представленная в таком виде, непригодна для проведения вычислений с ней, что важно, например, при определении срока годности информации и т.п. В связи с этим возникает необходимость в перекодировке даты из рассмотренного вида в число - порядковый номер дня, начиная, например, с 1 января 1901 года, и в обратной перекодировке - из числа в упоминавшуюся строку из 6 символов.

Этой цели и служат две написанные на ассемблере [2] подпрограммы - PDDDDYD и PDDDDYU, первая из которых переводит дату из вида YYMMDD (с атрибутом PICTURE'(6)9') в двоичное число DDDDD, занимающее 4 байта в памяти, а вторая осуществляет обратную перекодировку. Каждая из подпрограмм имеет два параметра; первый из них является аргументом, второй - результатом.

Применявшийся алгоритм виден из текста и комментариев к подпрограммам. Для экономии места всюду, где возможно, действия производились с полусловами. Указанные подпрограммы производят также проверку исходных данных и при обнаружении синтаксических и/или семантических ошибок выдают в результате нуль.

ЛИТЕРАТУРА

1. Килев У.И. Подмножество PL/1. Справочное руководство по программированию. Рига, 1975. 16 с.
2. Chapin N. 360/370 Programing in Assembly Language. Second Edition. McGraw-Hill Book Co., 1973, 580 pp.

```

1 PRINT ROGEN
2 PPSBYD START 0 CONVERTS PYMMDD (MUST BE P10'1619')
3 USING =.15 INTO FIX BIN DDDDD
4 SAVE (14,12) PASSED SINCE JAN 1,1901
5 L 2:0:11
6 RVC YMO:0:12
7 L Y:4:11
8 RVI DAYS+5,X'10'
9 XC WORK,WORK
10 MY2 WORK,YMO
11 CLC WORK,=CL6'000000'
12 BNE INVALID NON-NUMERIC
13 LH 6:0H'4'
14 IC 7:YMO(6) DIGIT 1
15 N 7:0F'15'
16 CH 7:0H'10'
17 BNL INVALID
18 IC 8:YMO+1(6) DIGIT 2
19 N 8:0F'15'
20 MH 7:0H'10'
21 AR 7:0
22 BZ INVALID ZERO1
23 STM 7:YMO(6)
24 SH 6:0H'2'
25 BNL LOOP
26 LH 6:YMO
27 CH 4:YMO+2 YEAR & RESULT
28 LH 5:YMO+4 MONTH
29 CH 4:0H'13' DAY
30 BNL INVALID
31 BCTR 3:0
32 MH 6:0H'1461'
33 SRA 6:2
34 NI YMO+1,X'03' DAYS PASSED TILL JAN 1
35 BNZ NOTLEAP LEAP?
36 RVI DAYS+5,X'10' LEAP - FEB 29
37 CH 4:0H'2' AFTER FEB 29?
38 BNH NOTLEAP
39 AH 6:0H'1'
40 NOTLEAP BCTR 4:0
41 SLA 4:2 SUBTRACT 1 = MONTH AS INDEX
42 CH 5:DAYS(4) NO OF DAYS > ...?
43 BH INVALID
44 AH 6:DAYS*2(4) DAYS PASSED SINCE 1ST OF ...
45 AR 6:3 DATE-TODAY
46 THATSALL ST 6:0(4)
47 INVALID SR 6:0
48 B THATSALL
49 WORK DS CL6
50 YMD OS CL6
51 DAYS OS 2F
52 ENU OC H'31,010,31,31,30,30,30,31,120,30,15,31,101,31,212,30,1
53 =F'15'
54 =CL6'000000'
55 =H'4'
56 =H'10'
57 =H'2'
58 =H'13'
59 =H'1461'
60 =H'1'

```


1		PRINT	NOHEN	
2	POUNDS	START	0	CONVERTS FIX BIN DDDDD
3		USING	0,15	INTO YYMMDD (PIC '(6)9')
4		SAVE	(14,12)	PASSED SINCE JAN 1,1961
5		L	2:0(1)	
6		L	3:0(2)	DDDD IN R3
10		MVI	DAYS+9,X'3A'	
11		LTR	3:3	
12		BNH	INVALID	IF =0
13		C	3:=F'36160'	
14		BNL	INVALID	TOO HIGH
15		SLA	3:2	
16		SR	2:2	
17		D	2:=F'1461'	
18		SRA	2:2	
19		LR	5:3	
20		N	5:=XL4'3'	LEAP+11
21		BNZ	OK	
22		BCTR	2:0	
23	OK	AM	3:=H'1'	
24		LR	5:3	
25		MH	5:HUNDRED+2	
26		N	3:=XL4'3'	LEAP?
27		BNZ	NOTLEAP	
28		MVI	DAYS+9,X'3B'	
29		CH	2:=H'9B'	
30		BNH	NOTLEAP	
31		BCTR	2:0	
32	NOTLEAP	LH	4:=H'26'	
33	CHECK	CM	2:DAYS(4)	
34		BCTR	4:0	
35		BCTR	4:0	
36		BNH	CHECK	POUND!
37		SH	2:DAYS+2(4)	DAY
38		SRA	4:1	MONTH
39		AR	5:4	YEAR & MONTH
40		SR	4:4	
41		M	4:HUNDRED	
42		AR	5:2	YYMMDD IN BINARY
43	CONV	CVD	5:DECIMAL	
44		UNPK	YYMMDD:DECIMAL	
45		MVZ	YYMMDD=5(11),=X'F0'	
46		L	2:4(1)	
47		MVC	0(6,2):YYMMDD	
48		RETURN	(12,12)	
53	INVALID	SR	5:3	
54		B	CONV	
55		DS	00	
56	DECIMAL	DS	D	
57	HUNDRED	DC	F'100'	
58	DAYS	DC	H'0(6):1,3B,0,60,119,150,180,211,242,272,303,333	
59	YYMMDD	DS	CL6	
60		END		
61			=F'36160'	
62			=F'1461'	
63			=XL4'3'	
64			=H'1'	
65			=H'9B'	
66			=H'26'	
67			=X'F0'	

АРХИТЕКТУРА СИСТЕМЫ БАНКА ДАННЫХ В Д

Г.В.Лучкин

В работе рассмотрена общая структура универсальной автоматизированной системы банка данных (В Д), которая разрабатывается Вычислительным центром Латвийского государственного университета им.П.Стучки.

1. Введение

Банк данных - это автоматизированная информационная система, осуществляющая накопление, интегрированное хранение и изменение больших совокупностей взаимосвязанных данных многофункционального назначения, а также подготовку и выдачу их по запросам различных пользователей (абонентов).

Составными частями банка данных являются:

- база данных (информационная база),
- система управления банком данных (система банка данных),
- техническая база (ЭВМ с периферийным оборудованием).

База данных - это совокупность данных в виде одного или нескольких файлов, состоящая из проблемных данных, поставляемых пользователем, и системных, создаваемых системой банка данных автоматически, используемых для организации логических связей между проблемными данными и обеспечивающих эффективный доступ к ним.

Система банка данных представляет собой комплекс алгоритмов, программ, языковых средств и методических ма-

териалов, обеспечивающих выполнение основных функций банка данных:

- описание базы данных,
- создание банка данных,
- изменение содержания и структуры данных,
- выдача данных по запросу,
- защита банка данных,
- оптимизация использования ресурсов и др.

Языковые средства банка данных представляют собой совокупность средств описания структуры данных (язык описания данных - *DDL*), управления данными (*DML*) и обмена абонента с банком данных (язык запросов - *DBL*). Различные категории пользователей применяют различные языковые средства: администратор банка данных - *DDL*, системный программист (в некоторых случаях - проблемный) - *DML* и абонент банка данных - *DBL*.

По характеру используемых языковых средств можно выделить два класса систем банков данных:

- системы с базовым языком,
- системы с автономным языком.

Системы с базовым языком в качестве основного описательного и процедурного средства используют универсальный или проблемноориентированный язык программирования, расширенный введением новых операторов, библиотечных процедур и др. В качестве базового языка обычно используются: *COBOL*, *PL/I*, *FORTRAN*. В некоторых системах возможно использование нескольких базовых языков. При использовании систем с базовым языком может быть достигнута высокая эффективность обработки и большая гибкость, но время получения ответа на нерегламентированный запрос к банку данных удлиняется, т.к. необходимо составить, отладить и запустить новую программу.

Системы с автономным языком используют специализированный язык для обмена с абонентами. Язык обмена позволяет

даже неквалифицированному пользователю в простом и наглядном виде формулировать запросы на поиск нужной информации в банке данных, изменение содержания банка данных, несложную обработку информации при выдаче. Такие системы обычно рассчитаны на работу в режиме коллективного доступа.

По типу структуры базы данных системы банков данных делятся на два класса:

- системы с форматизованными данными,
- системы с неформатизованными данными.

В системах с форматизованными данными единицы хранения и обработки (записи, сегменты, реквизиты) имеют жестко установленный формат, так что смысл каждого отдельного элемента определяется его расположением и значение имеет постоянную структуру. Описание проблемных данных в таких системах обычно включается в состав базы данных в виде отдельных системных файлов (таблиц описания - *DDI*). Возможно также и встроенное описание, когда вместе со значением каждого данного хранится и его описание, указывающее на формат и смысл. Системы с форматизованными данными применяются в автоматизированных системах управления, системах обработки данных экономического характера, административных системах.

В неформатизованных системах банков данных информационные единицы не имеют жесткой структуры, т.е. записи могут быть составлены из различного числа полей, располагающихся в любом порядке. Основное применение таких систем - информационно-поисковые системы различного назначения.

По нашему мнению, особый интерес представляют системы, относящиеся к промежуточному классу - полужатизованные. У таких систем структура данных должна быть стандартизована, но формат жестко не фиксирован и может быть изменен пользователем для каждого конкретного данного. Системы такого класса позволят покрыть значительную область приложений - от АСУ до ИПС.

Кроме указанных свойств, системы банков данных отличаются также:

- техникой логических связей (адресные цепи или индексные таблицы);
- методами доступа к проблемным данным (последовательный, индексно-последовательный, прямой);
- представлением записи как единицы хранения (физически связанный набор полей, логическая совокупность);
- используемое вычислительное оборудование и операционные системы и др.

Нашей задачей является разработка основных принципов построения систем банка данных универсального назначения с подформатизованными данными, доступ к которым возможен как на базовом языке, так и на языке запросов. Для хранения базы данных в системе должны использоваться запоминающие устройства прямого доступа ЕС ЭВМ, система должна работать под управлением операционной системы ДЭС/ЕС. При этом должны быть приняты во внимание следующие основные требования, предъявляемые к системам банков данных:

- универсальность, позволяющая удовлетворить потребности широкого круга пользователей,
- гибкость структуры базы данных, обеспечивающая ее легкое изменение и наращивание,
- минимальная избыточность представления данных,
- независимость данных от их использования, что позволяет независимо менять программы, структуру и физическое представление данных,
- дифференцированность пользователей и средств работы с банком данных,
- защита данных от их разрушения,
- секретность данных, связанная с установлением компетенции пользователей в отношении вида доступа к определенным категориям данных,
- удобство пользования и эффективность обработки.

2. Структура базы данных

Предлагаемая методика построения базы данных системы **8D** основывается на следующих принципиальных положениях:

- В качестве основной единицы хранения и обработки принята запись данных, которая при хранении может быть разделена на сегменты.
 - Структура записи стандартизована, но формат жестко не установлен и в системе не описан.
 - Описание состава записи является встроенным в структуру записи.
 - Описания отдельных полей (ключей, дескрипторов, адресных ссылок, информационных полей) вынесены из основных данных и хранятся отдельно.
 - Логические связи между записями определяются только отношением включения, число иерархических уровней не ограничивается.
 - Логические связи определяются наличием у записей дескрипторов с определенным именем и значением и реализуются созданием индексных таблиц; в качестве вспомогательного средства используются адресные ссылки, поддерживаемые самим пользователем.
 - Для управления как проблемными, так и системными данными создается система рандомизированной организации данных (**ROD**), предусматривающая использование запоминающих устройств с произвольным доступом.
 - База данных представляется в виде одного файла, состоящего из трех субфайлов: основного, организующего и рабочего.
- Стандарт оформления записи информационной базы системы **8D** заключается в следующем:
- Любая запись представляется в форме записи переменной длины (V) аналогично СУБВ операционных систем

(ДЭС/ЕС, ОС/ЕС и др.). При этом к основной информации записи добавляется поле длиной в четыре байта, в первых двух из которых в виде двоичного целого числа указывается общая длина записи (< 65536).

- Поле основной информации разделяется на произвольное число элементов, каждый из которых также имеет переменную длину. Длина элемента указывается в его левом байте в виде двоичного целого числа (< 256).

- Первым элементом записи является ключевой элемент (ключ), который определяет тип записи. Остальные элементы могут располагаться в произвольном порядке.

- Записи одного типа могут состоять из разного набора элементов. Элементы с неопределенным (не присвоенным) значением могут в записи не храниться.

- Каждый элемент записи определяется двумя категориями: именем элемента и значением элемента. Имя элемента определяет смысл его значения и является встроенным описателем.

- В базовом варианте системы В₁ длина имени элемента равна 3 символам (буквы, цифры, специальные знаки). Если ограничиться только 84 знаками, имеющимися на устройствах подготовки данных и печатающих устройствах, то максимальное количество различных имен элементов равно 592704.

Описанный стандарт структуры записи допускает простое расширение. Если не фиксировать длину имени элемента, то можно рассматривать два крайних случая его структуры:

а) элемент-значение, т.е. элемент с нулевой длиной имени. В этом случае структура записи не имеет встроенного описания и должна иметь жестко установленный формат (форматизованные данные);

б) элемент-имя с нулевой длиной значения. Такие элементы могут быть ключевыми словами и составлять неформатизованные структуры.

По назначению и использованию в системе все элементы

записей данных можно разделить на четыре класса:

- а) ключи,
- б) дескрипторы,
- в) адресные ссылки,
- г) информационные поля.

Ключ однозначно идентифицирует тип записи (имя) и конкретную запись в типе (значение). Дескриптор (аспект) используется для образования логических связей между записями. При этом определяются два уровня логических связей: связь записей с одинаковыми именами дескрипторов и связь записей с одинаковыми именами и значениями дескрипторов. Адресная ссылка прямо указывает адрес записи, которая входит в ту же логическую совокупность, что и данная. Адресные ссылки в проблемных данных формируются самим пользователем и являются вспомогательным средством логических связей. Адресные связи системных данных поддерживает сама система. Информационные поля — это элементы данных, формируемые и используемые в проблемных вычислениях.

Каждый из элементов (кроме ключа) может быть не только скалярном, но и массивом (таблицей). Формат раздела таблицы является фиксированным, но число разделов может меняться.

В информационной базе системы *BD* используются единые принципы организации размещения данных в запоминающих устройствах и методов доступа к ним. Эти принципы реализованы в виде системы рандомизированной организации данных (*ROD*), которая является развитием системы управления данными операционной системы. В настоящее время осуществлена первоначальная реализация системы *ROD* под названием "страничная организация памяти на магнитных дисках (*SOP*)" (см. статью в настоящем сборнике). Система *ROD* предназначена для управления размещением записей банка данных в запоминающем устройстве с произвольным доступом и обеспечения прямого доступа к любой записи после указания ее ключа.

В системе *RDD* база данных рассматривается как файл с методом доступа *REGIONAL (1)*. Областью файла является одна дорожка дискового запоминающего устройства ЕС-5056, оформленная в виде физической страницы. Каждая запись, помещенная в физическую страницу, интерпретируется как строка страницы и снабжается уникальным номером (< 256). Номер страницы и номер строки образуют физический адрес, который используется для ссылок на запись и ее непосредственного нахождения. Управление размещением записей в системе *RDD* осуществляется с помощью механизма рандомизации и адресных цепей. Рандомизация заключается в определении местоположения (номера страницы) записи по ее ключу. При этом проблема равномерности заполнения страниц базы данных может быть решена введением управляемой рандомизации.

Рандом-генератор определяет место записи с точностью до страницы. Несколько записей, которым назначена одна страница, называются синонимами. Если на странице отсутствует место для очередного синонима, то он помещается в ближайшее свободное место в пределах цилиндра. Все синонимы одной страницы образуют логическую страницу, реализованную в виде замкнутой адресной цепи. Начало логической страницы (анкер цепи) — это заголовок той физической страницы (анкерной), которая определена рандом-генератором для размещения синонимов — звеньев логической цепи. Записи в этой цепи расположены в последовательности возрастания значений ключей.

Описанная стандартная структура записи рассмотрена с точки зрения проблемного программиста и будет называться внешней структурой. При хранении записи в системе *RDD* она имеет внутреннюю структуру. Запись внутренней структуры содержит еще дополнительные поля: собственный физический адрес, поле адресной ссылки на следующую запись и поле ссылки на свой сегмент. Каждая адресная ссылка — это физический адрес соответствующей единицы хранения.

В системе *ROD* осуществляется сегментация записей, используемая для обеспечения более плотного заполнения страниц, размещение записи возможно близко от анкерной страницы и снятия ограничения на длину записи, накладываемую длиной страницы. Все сегменты одной записи представляются в виде замкнутой логической цепи, анкером которой служит нулевой сегмент. Сегмент-анкер записи должен иметь три адресных поля: собственный адрес, адрес первого сегмента-звена и адрес следующего синонима логической страницы. Сегмент-звено имеет только одно адресное поле. Для несегментируемых записей резервируются только два адресных поля: собственный физический адрес и ссылка на синоним.

При выборе алгоритма сегментирования следует иметь в виду следующие правила:

- расположение ключа записи целиком в анкерном сегменте,
- максимальное увеличение средней длины сегмента,
- установление минимальной длины сегмента,
- размещение всех сегментов записи на одной физической странице или по крайней мере в пределах одного цилиндра и др.

Ввиду различного назначения, режима использования, уровня защиты, секретности и других параметров все данные в системе *BD* группируются в трех субфайлах *ROD* :

- а) субфайл *INF* , в котором размещаются все проблемные данные (основная информация);
- б) субфайл *ORG* , содержащий индексные таблицы логических связей основной информации, данные описательного характера, вспомогательные данные, обеспечивающие ввод и вывод информации и др.;
- в) субфайл *RAW* , являющийся рабочим, где при поиске данных формируются промежуточные и выходные индексные таблицы. В файле *RAW* на основе механизмов системы *ROP* формируются стековые структуры.

Кроме единого файла *ROD* базы данных системы *BD*, в процессе ее функционирования формируется протокольный файл

(журнал) на магнитной ленте *JRN*. Этот файл предназначен для обеспечения перепрогона сеанса работы с банком данных с целью восстановления состояния базы данных. Записями файла *JRN* являются физические страницы *RCD*, которые были изменены при работе с банком данных.

3. Программное обеспечение системы

В системе *BD* программное обеспечение представляет собой совокупность библиотечных программ, к которым с помощью средств операционной системы могут иметь доступ различные группы пользователей. Программы системы, обслуживающие потребности проблемных программистов, работающих на базовом языке программирования, вызываются прикладной программой с помощью оператора *CALL*. Этот оператор используется в языках *FORTRAN*, *COBOL*, *PL/I*. Форма представления параметров позволяет использовать эти языки как базовые. В качестве основного выбран язык *PL/I*. На этом же языке программируются и сами процедуры программного обеспечения банка данных. При программировании некоторых системных процедур (например, рандом-генератора) используется язык *ASSEMBLER*.

Программное обеспечение системы *BD* обслуживает следующие функции:

- инициализация и параметрическое описание банка данных;
- описание и начальное создание системного субфайла (*ORF*);
- накопление и корректировка проблемных данных;
- формирование и обслуживание логических связей между данными;
- поиск и выдача данных по запросу пользователя;
- защита данных от некомпетентного доступа;
- обеспечение безопасности базы данных с возможностью восстановления информации, разрушенной при обработке или хранении.

Состав процедур программного обеспечения базовой версии системы *BD* следующий:

1. *BDCU* - разметка областей базы данных;
2. *BDPI* - подготовка страниц базы данных;
3. *BDCI* - ведение системного субфайла базы данных;
4. *BDCP* - справки о составе системного субфайла;
5. *BDQ* - обработка запроса пользователя;
6. *BDINF* - обмен с субфайлом *INF* ;
7. *BDORG* - обмен с субфайлом *ORG* ;
8. *BDRAV* - обмен с субфайлом *RAV* ;
9. *BDRP* - обработка записи банка данных;
10. *BDJRN* - ведение файла *JRN* ;
11. *BDJR* - восстановление базы данных с *JRN* ;
12. *BDIO* - загрузка/выгрузка базы данных.

Процедуры 1-4 обеспечивают режим инициализации банка данных, изменения и контроля состава системных данных. Эти процедуры отделены от пользователя и входят в компетенцию системного программиста (администратора банка данных).

Интерфейс проблемного программиста (пользователя) с банком данных обеспечивается средствами операционной системы и универсальной процедурой *BDQ*, воспринимающей и обрабатывающей запрос и с помощью других процедур (*BDINF*, *BDORG*, *BDRAV*) выполняющей информационный обмен с базой данных.

В базовой версии системы *BD* не предусматривается работа одновременно с несколькими пользователями. Закрепление устройств ЭВМ за определенным заданием (или разделом) препятствует возможности одновременной работы нескольких проблемных программ со своими экземплярами процедуры *BDQ*. Тем самым проблема синхронизации действий пользователей решается за счет возможности работы со многими абонентами одновременно. Возможным решением было бы выделение системы *BD* в качестве отдельного задания, работающего в условиях ДОС/ЕС в отдельном разделе. При этом для много-

программной работы с *БД* необходимо наличие командно-информационной связи между разделами (заданиями), которая в ДС/ЕС отсутствует.

Так как структура записи, предоставляемой пользователю, достаточно сложна (например, сканирование элементов записи средствами *FORTRAN* или *COBOL* затруднительно), в системе *БД* предусмотрена специальная процедура обработки записи, доступная для проблемных программ.

Процедура *ADDRP* выполняет следующие функции:

- A* - инициализирование записи,
- B* - упорядочение элементов (лексикографическая сортировка),
- C* - включение нового элемента,
- D* - корректировка (замена) элемента,
- E* - аннулирование (устранение) элемента,
- F* - выборка элемента.

При выполнении функций над элементом (*C-F*) записи он может быть идентифицирован именем, порядковым номером, либо двумя параметрами сразу. Эти функции заключаются в информационном обмене между записью и полями рабочей памяти, имеющими вид, удобный для обработки на основных языках программирования. Каждый элемент представляется четырьмя рабочими полями (развернутый вид): длина, имя, значение, порядковый номер в записи.

4. Языковые средства системы *БД*

Языковые средства системы *БД* включают:

- а) язык описания базы данных,
- б) язык управления данными (внутренний язык),
- в) язык запросов (внешний язык).

Язык описания используется системным программистом при начальном создании базы данных и при последующих изменениях

ее структуры и системы обслуживания абонентов. На этом языке описываются общие параметры системы (размеры основного файла и субфайлов, возможность сегментации записей и др.), элементы записей, пароли и приоритеты пользователей. Язык представляет собой набор небольшого числа операторов, имеющих фиксированную синтаксическую структуру и действующих независимо. Каждый оператор помещается на одной перфокарте и синтаксис его полностью определяется макетом перфорации. Оператор описания элемента записи должен иметь следующие параметры (режимы):

- имя элемента,
- класс элемента (ключ, дескриптор, адресная ссылка, поле),
- атрибуты данного (основание системы счисления, упаковка, способ представления, разрядность),
- длина элемента таблицы, если элемент является таблицей (массивом),
- допустимость нулевого и отрицательного значения,
- обозначение пароля, ограничивающего доступ к записи (для ключей),
- полное имя элемента.

Язык управления данными предназначен для описания и реализации следующих функций:

- открытие файла базы данных, т.е. начало работы с системой *BD*,
- закрытие (окончание обработки),
- чтение отдельной записи или логической совокупности записей данных,
- запись в банк данных отдельных записей.

Общие вид оператора внутреннего языка:

CALL DD (< список параметров >);

Полный список параметров включает следующие:

- вид обработки (функция и режим),
- условие поиска,

- зона ответа,
- ключ записи,
- физический адрес,
- зона записи.

Вход в процедуру *ADQ* возможен и через дополнительные точки входа. Например, начало работы с банком данных реализуется оператором:

CALL BDQOPN (<обозначение абонента>, <пароль>);,
а завершение обработки:

CALL BDQCLS

Для задания условия поиска в языке используется логическое выражение, составленное из программных условий, объединенных с помощью символов логических операций "И" (*&*) и "ИЛИ" (*|*). Перед простым условием может стоять символ операции "НЕ" (*!*). Простое условие может иметь одну из трех форм:

- а) <имя элемента> <знак операции сравнения> <константа>,
- б) <имя элемента> <знак операции сравнения> <имя данного>,
- в) <имя элемента> = ?.

Имя элемента - это трехсимвольное имя анализируемого при поиске элемента записи данных. Некоторые из символов имени могут быть заменены служебным знаком '?', который запрещено употреблять при назначении имен элементов. Этот знак обозначает любой символ. Например, обозначение

A??

соответствует любому имени, начинающемуся с буквы А. Если элемент является таблицей, то имя может индексироваться.

В качестве знаков операции сравнения допустимо использовать все такие знаки, определенные в языке *PL/I*

Константа, указанная в форме а) условия поиска, представляет собой любой набор символов, который может быть преобразован в соответствии с описанием имени, состоящего в левой части.

Имя данного, указанное в форме б), - это имя рабочей области, сформированной пользователем в форме элемента записи, имя которой является параметром оператора внутреннего языка.

Условие поиска в форме в) указывает необходимость проверки наличия у записи хотя бы одного элемента с определенным именем и произвольным значением.

Язык запросов состоит из операторов, реализующих функции, подобные описанным для языка управления данными. Основные операторы языка запросов следующие:

- OPEN** (ОТКРЫТЬ) - начало работы с банком данных с проверкой пароля абонента,
CLOSE (ЗАКРЫТЬ) - окончание сеанса связи,
READ (ЧИТАТЬ) - чтение и выдача на терминал абонента отдельной записи,
WRITE (ПИСАТЬ) - записывание данных, напечатанных абонентом на терминале,
DELETE (УБРАТЬ) - стирание (зачеркивание) записи с указанным ключом,
RETRIEVE (НАЙТИ) - поиск совокупности записей в соответствии с условием поиска и сообщением о количестве найденных записей.

Оператор **READ** может работать в дополнительном режиме **NEXT (READ NEXT - ЧИТАТЬ СЛЕДУЮЩУЮ)**, что соответствует выборке и печати на терминале абонента очередной записи из найденной (оператор **RETRIEVE**) логической совокупности.

Кроме указанных основных операторов используются еще вспомогательные операторы, управляющие формой выдачи, организующие обработку записи (аналогично процедуре **BDPP**) и др.

Совокупность описанных языковых средств обеспечивает централизованное обслуживание банка данных, а также гибкую

и удобную работу пользователя как в режиме пакетной обработки программ на базовом языке, так и в диалоговом режиме через абонентский терминал.

5. Расширения системы

Система банка данных *БД* является универсальной и может быть использована в качестве ядра интегральных систем обработки данных.

Основными элементами, расширяющими возможности *БД* могут быть:

- а) блок ввода и предварительной обработки данных в пакетном режиме. Информационно-справочной базой этого блока является файл, описывающий структуру и объемные характеристики вводимых, промежуточных и выходных данных, параметры обработки, средства контроля и др. Программное обеспечение блока ввода должно обеспечить выполнение операций ввода исходных данных с различных носителей (ПК, ПЛ), контроль данных и получение технологических документов, накопление, редактирование, сортировку данных перед их вводом в банк данных;
- б) блок вывода массовых данных, который инициируется регламентированными запросами. Этот блок осуществляет опрос банка данных, формирование рабочих файлов, обработку данных (в основном, редактирование, подсчет итогов, вычисление статистических показателей и т.п.) и вывод результатов в виде документов, удобных для пользователя;
- в) блок диалога с абонентами, осуществляющий информационную связь пользователей с банком данных в режиме реального времени. Основной функцией блока является трансляция высказываний на языке запросов с преобразованием их в операторы внутреннего языка. Эти операторы в системе *БД* интерпретируются процедурой *БД*.

Указанные расширения являются стандартизованными и предусматривают некоторые ограничения на форматы входа и выхода, а также унифицированные схемы обработки. Пользователи, которых не удовлетворяет состав предлагаемых блоков, могут вместо них встроить специализированные в соответствии с характером решаемых проблем. Однако общая структура любой автоматизированной системы обработки информации, построенной на базе банка данных будет следующей:

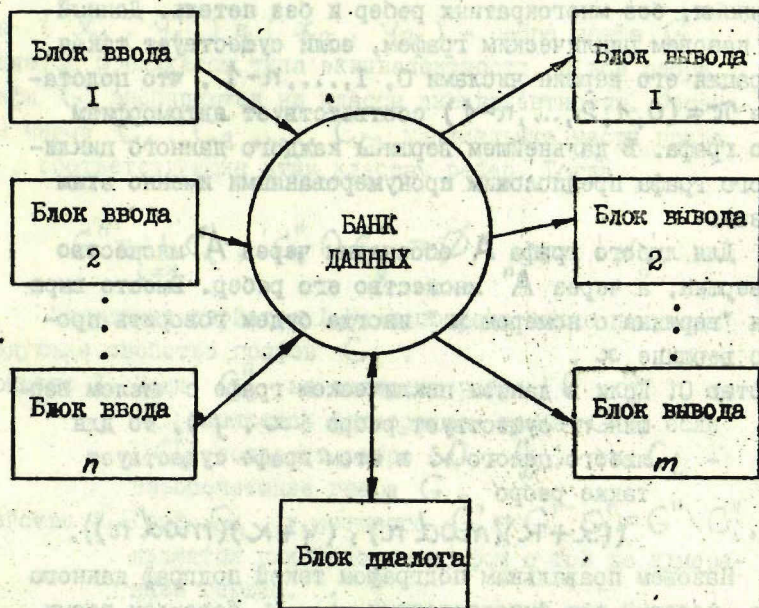


Схема интегрированной автоматизированной информационной системы.

О СТРОЕНИИ ЦИКЛИЧЕСКИХ ГРАФОВ

П. Б. Кикуст.

В настоящей работе изучаются вершинно-симметрические графы, обладающие циклическим автоморфизмом. В литературе, в частности в работах [1,2], такие графы называются "звездными полигонами". Мы предпочитаем термин "циклический граф", а основные понятия теории графов используем согласно [5,6,7].

Исследуемые графы предположим конечными, неориентированными, без многократных ребер и без петель. Данный граф назовем циклическим графом, если существует такая нумерация его вершин числами $0, 1, \dots, n-1$, что подстановке $\pi = (0, 1, 2, \dots, n-1)$ соответствует автоморфизм этого графа. В дальнейшем вершины каждого данного циклического графа предположим пронумерованными именно этим образом.

Для любого графа A обозначим через A' множество его вершин, а через A'' множество его ребер. Вместо выражения "вершина с номером x " иногда будем говорить просто о вершине x .

Свойство 0. Если в данном циклическом графе с числом вершин n существует ребро (x, y) , то для любого целого k в этом графе существует также ребро

$$((x+k)(\text{mod } n), (y+k)(\text{mod } n)).$$

Назовем правильным подграфом такой подграф данного графа, который при фиксированных j, k порожден всеми вершинами с номерами вида $(j+ik)(\text{mod } n)$, $i = 0, 1, 2, \dots$, где n - число вершин данного графа.

Свойство I. Правильный подграф данного циклического графа также является циклическим графом.

Свойство 2. Каждая связная компонента данного циклического графа является его правильным подграфом, все эти компоненты попарно изоморфны.

Свойства 0, 1, 2, непосредственно следуют из определения циклического графа.

Пусть дан циклический граф G с числом вершин n , и подстановке π соответствует его автоморфизм. Очевидно, двухместный предикат на множестве G'' , определяемый формулой

$$\exists \kappa ((x_2, y_2) = (\pi^\kappa(x_1), \pi^\kappa(y_1))),$$

где (x_1, y_1) и (x_2, y_2) - ребра графа G , является предикатом типа эквивалентности, и все ребра графа G распадаются на классы эквивалентности. Обозначим через $C_1, C_2, \dots, C_m$ минимальные части графа G , соответствующие этим классам ребер. Тогда

$$G'' = \bigcup_{i=1}^m C_i'', \quad C_i'' \cap C_j'' = \emptyset \quad \text{при } i \neq j.$$

Используя свойства 1, 2, непосредственно получаем следующие свойства графов C_i .

Свойство 3. Если C_i'' не содержит ребро $(0, \frac{n}{2})$, то C_i является фактором графа G , если C_i'' содержит ребро $(0, \frac{n}{2})$, то C_i - паросочетание графа G .

Свойство 4. Граф G_1 , у которого $G'_1 = G'$, $G''_1 = G'' \setminus C_i''$, является циклическим графом с той же нумерацией вершин.

ТЕОРЕМА I. В связном циклическом графе существует гамильтонов цикл.

ДОКАЗАТЕЛЬСТВО проведем индукцией по числу вершин, полагая для удобства рассуждений, что га-

мильтонов цикл существует в полном графе K_2 .

Пусть во всех связанных циклических графах с числом вершин не больше n существует гамильтонов цикл, а G - такой связный циклический граф, что $|G'| = n+1$. Необходимо рассмотреть случай, когда ни одна из частей $C_1, C_2, \dots, C_m$ не является гамильтоновым циклом графа G .

Обозначим через G_k такой граф, у которого $G'_k = G'$, $G''_k = G''_{k-1} \setminus C_k$, а $G_0 = G$. Очевидно, существует такое k , что граф G_k связный, а граф G_{k+1} несвязный. Согласно свойствам 2, 4, все связные компоненты графа G_{k+1} изоморфны некоторому связному циклическому графу A , такому, что $2 \leq |A'| \leq n$, и в котором по индуктивному предположению существует гамильтонов цикл.

Пусть $C''_{k+1} \supset (0, l)$, A_0 - одна из связных компонент графа G_{k+1} , $i_1, i_2, \dots, i_t$ - номера вершин графа A_0 в графе G , и $(i_1, i_2, \dots, i_t, i_1)$ - гамильтонов цикл графа A_0 .

Вершинам с номерами $i_\alpha + \beta l$, где β - фиксировано, $\alpha = 1, \dots, t$, соответствует некоторая связная компонента графа $G_{k+1} - A_\beta$ и $(i_1 + \beta l, i_2 + \beta l, \dots, i_t + \beta l, i_1 + \beta l)$ - ее гамильтонов цикл. Из-за связности графа G_k существует такое наименьшее число s , что $\bigcup_{\beta=0}^s A'_\beta = G'$. Так как по предположению часть C_{k+1} не гамильтонов цикл, то $t \geq 2$, и искомый гамильтонов цикл графа G задается в следующем виде:
для нечетного s -

$$(i_1, i_2, \dots, i_t, i_t + l, i_{t-1} + l, \dots, i_2 + l, i_2 + 2l, \dots, i_t + 2l, i_t + 3l, \dots, i_t + sl, i_{t-1} + sl, \dots, i_2 + sl, i_1 + sl, i_1 + (s-1)l, \dots, i_1),$$

для четного s - аналогично, добавляя лишь между вершинами $i_2 + sl$ и $i_1 + sl$ простую цепь $(i_3 + sl, i_4 + sl, \dots, i_t + sl)$.

Этим теорема I доказана.

ТЕОРЕМА 2. Пусть p - степень вершин данного циклического графа G , тогда любые две смежные вершины этого графа могут быть соединены p -элементарными внутренне непересекающимися цепями.

ДОКАЗАТЕЛЬСТВО. Можем предположить,

что одна из двух смежных вершин имеет номер 0, вторая - номер ℓ . Ребро $(0, \ell)$ принадлежит некоторой части

C_i графа G . Так как $C_i = G'$, то каждая вершина, смежная вершине 0, принадлежит некоторой связной компоненте графа C_i . Пусть C - одна из таких компонент, а $t_1, t_2, \dots, t_k$ - номера всех вершин смежных вершине 0 и принадлежащих C' . Тогда $(t_j + \ell) \pmod{n}$, $j = 1, 2, \dots, k$ - номера всех вершин, смежных вершине ℓ и также принадлежащих C' . Номера t_j выразим в виде

$t_j = t_1 + \alpha_j \ell$ ($j = 1, 2, \dots, k$). Нумерацию можем выбрать так, чтобы $\alpha_1 < \alpha_2 < \dots < \alpha_k$. Граф

C фактически является либо элементарным циклом, либо ребром, поэтому любые две вершины из C' естественным образом определяют не более двух простых цепей. Так как

$t_j + \ell = t_1 + (\alpha_j + 1)\ell$, и $\alpha_j + 1 \leq \alpha_{j+1}$ при $j = 1, \dots, k-1$, то для вершин с номерами $t_{(j+1) \pmod{k}}$

и $t_j + \ell$, $j = 1, 2, \dots, k$ всегда существует одна такая цепь, которая не содержит других вершин смежных вершине 0 или вершине ℓ .

Таким образом, цепи

$(0, t_{(j+1) \pmod{k}}, K, t_j + \ell, \ell)$, $j = 1, 2, \dots, k$,

где K означает вышеупомянутую цепь графа C , являются элементарными, соединяют вершины 0 и ℓ и попарно не имеют общих внутренних вершин. Случай $t_{(j+1) \pmod{k}} = t_j + \ell$ требует очевидного сокращения записи, а если C' содержит вершину 0, то необходимо отдельно рассмотреть случай

$j=k$, когда приведенное выражение превращается в $(0, t_1, 0, \ell)$. Но этот недостаток также устраняется тривиальным образом.

Рассматривая отдельно все связные компоненты графа G_i , заключаем, что число найденных цепей совпадает с числом вершин, смежных вершине 0, т.е. с p .

Теорема 2 доказана.

Пусть $H_1, H_2, \dots$ - семейство таких подграфов с наименьшим числом вершин неполного связного графа G , что каждый его подграф H_i с множеством вершин $G \setminus H_i$ ($i = 1, 2, \dots$) не связан. Если A_i - наименьшая связная компонента подграфа H_i , то существует такое i_0 , что $|A_{i_0}| = \min |A_i|$. Положим $H_{i_0} = H$, $A_{i_0} = A$, и через B' обозначим подграф с множеством вершин $G \setminus (H' \cup A')$.

В литературе [3,4] подробно изучены подграфы H и A произвольных вершинно-симметрических графов. Здесь имеет место еще следующая теорема, в доказательстве которой, устанавливая необходимые свойства подграфов H и A , используем отличную от [3,4] версию рассуждений.

ТЕОРЕМА 3. Подграф A данного связного циклического графа G является его правильным подграфом.

ДОКАЗАТЕЛЬСТВО. Скажем, что подмножества множества G' несмежны, если ни одна вершина одного подмножества не смежна в графе G ни одной вершине другого подмножества. С очевидно, A' и B' несмежны.

В графе G имеется такое ребро (x, y) , что $x \in H'$ и $y \in A'$. Существует степень t подстановки π , что $y = \pi^t(x)$, и тогда при подстановке π^t каждый образ x и его оригинал смежны.

Рассмотрим множества $\pi^t(H')$, $\pi^t(A')$, $\pi^t(B')$. Имеем

$$A' = (A' \cap \pi^t(H')) \cup (A' \cap \pi^t(A')) \cup (A' \cap \pi^t(B'))$$

$$\pi^t(A') = (H' \cap \pi^t(A')) \cup (A' \cap \pi^t(A')) \cup (B' \cap \pi^t(A'))$$

$$|A'| = |\pi^t(A')|$$

Так как все вершины из $\pi^t(A')$ смежны вершинам из A' , и A' и B' несмежные подмножества, то $\pi^t(A') \cap B' = \emptyset$, и аналогично $\pi^t(B') \cap A' = \emptyset$. Поэтому

$$A' = (A' \cap \pi^t(H')) \cup (A' \cap \pi^t(A'))$$

$$\pi^t(A') = (H' \cap \pi^t(A')) \cup (A' \cap \pi^t(A'))$$

$$|A' \cap \pi^t(H')| = |H' \cap \pi^t(A')|$$

Подобным образом

$$B' = (B' \cap \pi^t(H')) \cup (B' \cap \pi^t(B'))$$

$$\pi^t(B') = (H' \cap \pi^t(B')) \cup (B' \cap \pi^t(B'))$$

$$|B' \cap \pi^t(H')| = |H' \cap \pi^t(B')|$$

Положим

$$|\pi^t(H') \cap H'| = u$$

$$|\pi^t(H') \cap A'| = a$$

$$|\pi^t(H') \cap B'| = b$$

$$|\pi^t(A') \cap A'| = \kappa_1$$

$$|\pi^t(B') \cap B'| = \kappa_2$$

$$|H'| = k$$

Тогда k является числом связности данного графа и $k = u + a + b$.

Из несмежности A' и B' следует несмежность $\pi^t(A')$ и $\pi^t(B')$. Поэтому несмежны подмножества $A' \cap \pi^t(A')$

и $H' \cap \pi^t(B')$, и также подмножества $H' \cap \pi^t(A')$ и $B' \cap \pi^t(B')$.

Докажем, что $\kappa_1 = 0$.

Если это не так, то $\kappa_2 = 0$, ибо в противном случае из-за свойства минимальности множеств H' и A' согласно несмежности указанных подмножеств следует

$$|A' \cap \pi^t(H') \cup (H' \cap \pi^t(A')) \cup (H' \cap \pi^t(H'))| > k$$

$$|B' \cap \pi^t(H') \cup (H' \cap \pi^t(B')) \cup (H' \cap \pi^t(H'))| \geq k,$$

т.е. $2a + u > k$ и $2b + u \geq k$, что противоречит равенству $k = u + a + b$.

Так как $|A'| \leq |B'|$, $|A'| = a + \kappa_1$, $|B'| = b + \kappa_2 = b$, то $a + \kappa_1 \leq b$. Пусть p - степень вершин графа G . Для вершин из $A' \cap \pi^t(A')$ $p \leq a + (\kappa_1 - 1) + (k - b) \leq k - 1$, что противоречит основному свойству числа связности $p \geq k$. Поэтому не может быть $\kappa_1 \neq 0$.

Свойство $\kappa_1 = 0$ означает, что подграф A не содержит ни одного ребра из класса C_i^t , содержащего ребро (x, y) , а так как проведенное рассуждение не зависит от выбора ребра, соединяющего вершину множества A' с вершиной множества H' , то подграф A не содержит ни одного ребра всех соответствующих классов. Из этого следует, что после удаления из G'' всех ребер этих классов, подграф A останется связным. Так как согласно свойству 4 удаление целого класса ребер не влияет на "циклическость", то подграф A является связной компонентой некоторого циклического графа с той же нумерацией вершин, что и у графа G . Из этого и свойства 2 заключаем, что A - правильный подграф данного графа.

Теорема 3 доказана.

Следствие I. Если p - степень вершин связного циклического графа, p' - наибольшая степень вершин во всех его правильных подграфах с числом вершин меньше p ,

то число связности $k \geq p - p'$.

ДОКАЗАТЕЛЬСТВО Из доказательства теоремы 3 непосредственно следует, что $\pi^t(A) = H' \cap \pi^t(A)$, т.е. $\pi^t(A) \subset H'$. Поэтому $|A| = |\pi^t(A)| \leq |H'| = k \leq p$. Если p_1 - степень вершин в подграфе A , то, очевидно, $p - p_1 \leq k$. Из последних двух неравенств следует требуемое неравенство.

СЛЕДСТВИЕ 2. Если все части C_i циклического графа являются его гамильтоновыми циклами, то $k = p$.

ДОКАЗАТЕЛЬСТВО В данном случае все правильные подграфы с числом вершин меньше p являются нуль-графами, т.е. $p' = 0$. Из второго неравенства следует $k \geq p$, и так как $k \leq p$, то $k = p$.

СЛЕДСТВИЕ 3. Если число вершин циклического графа простое, то $k = p$.

ДОКАЗАТЕЛЬСТВО В этом случае все части C_i данного циклического графа являются его гамильтоновыми циклами, и требуемое равенство следует из следствия 2.

ЛИТЕРАТУРА

1. James Turner, Point-Symmetric Graphs with a Prime Number of Points, Journal of Combinatorial Theory 3, p. 136-145, 1967.
2. Bernard Elspas, James Turner, Graphs with Circulant Adjacency Matrices, Journal of Combinatorial Theory vol 9, 297-307, 1970.
3. Mader W. Über den Zusammenhang symmetrischer Graphen. - "Arch. Math.", 1970, vol 21, No. 3, 331-336.
4. Визинг Г. О минимальной связности графов с транзитивной группой автоморфизмов. - В сб.: "Управляемые системы", вып. 7, Новосибирск, 1970, 46-50.
5. Берг К. Теория графов и ее применение. М., ИЛ., 1962.
6. Оре О. Теория графов. М., 1968.
7. Харари Ф. Теория графов. М., 1973.

НЕКОТОРЫЕ СВОЙСТВА ВПОЛНЕ НЕОДНОРОДНЫХ ГРАФОВ

А.О.Кац

Легко показать, что любой граф имеет по крайней мере две вершины с одинаковыми степенями, т.е. ни у какого графа не могут быть все степени вершин попарно различными.

Обыкновенный граф, имеющий более одной вершины называется вполне неоднородным, если все его вершины, за исключением двух, имеют попарно различные степени.

В работе [1] Бехзадом и Чартрэндом доказано, что существует точно два неизоморфных вполне неоднородных (в их терминологии почти совершенных) графа с n вершинами. Эти графы взаимно дополнительные. Небеским в работе [2] исследованы свойства вполне неоднородных графов. Данная заметка также посвящена исследованию некоторых свойств указанных графов.

Связный вполне неоднородный граф с n вершинами ($n > 1$) обозначим через G_n , дополнительный к нему граф через $\bar{G}_n$.

Если из $\bar{G}_n$ удалить изолированную вершину, то в результате получится граф G_{n-1} . Ограничимся здесь изучением свойств графов G_n , т.к. из изложенного выше следует, что эти свойства легко переносятся на свойства графов $\bar{G}_n$.

Вершины графа G_n , имеющие одинаковые степени, назовем однородными вершинами.

Пусть $d(x)$ обозначает степень вершины x . Пронумеруем вершины графа G_n от 1 до n так, чтобы

$N(x) > N(y)$, если $d(x) \geq d(y)$, где x, y - вершины G_n , а $N(x), N(y)$ присвоенные им номера.

Из определения и из результата Бехзада и Чартрэнда легко выводится, что для однородной вершины x или $N(x) = \lfloor \frac{n}{2} \rfloor$ или $N(x) = \lfloor \frac{n}{2} \rfloor + 1$ и если $d(x) < \lfloor \frac{n}{2} \rfloor$, то $N(x) = d(x)$, а если $d(x) > \lfloor \frac{n}{2} \rfloor$, то $N(x) = d(x) + 1$.

(здесь и везде, дальше $[a]$ обозначает целую часть a , т.е. максимальное целое, не превосходящее a).

ТЕОРЕМА 1. Пусть x — вершина графа G_n . Если $d(x) \leq [\frac{n}{2}]$, то x смежна таким и только таким вершинам y графа G_n , для которых $N(y) > n - d(x)$, а если $d(x) > [\frac{n}{2}]$, то x смежна таким и только таким вершинам y , для которых $N(y) \geq n - d(x)$.

ДОКАЗАТЕЛЬСТВО. Легко проверяется правильность теоремы для G_2 и G_3 . Предположим, что теорема верна для всех G_k , для которых $k < n$. Удалим из G_n вершины с номерами 1 и n (здесь и везде дальше словами "удалим вершину" понимается удаление вершины вместе с инцидентными ей ребрами) и полученный граф обозначим G' . Степень каждой вершины G' на 1 меньше степени соответствующей вершины G_n ($d'(x) = d(x) - 1$). Присвоим каждой вершине x графа G' новый номер $N'(x) = N(x) - 1$. А т.к. G' имеет $n-2$ вершины, то по предположению индукции x смежна с y тогда и только тогда, когда $N'(y) > n-2-d'(x)$, при $d'(x) \leq [\frac{n-2}{2}]$ и $N'(y) \geq n-2-d'(x)$, при $d'(x) > [\frac{n-2}{2}]$. Подставляя в эти неравенства $N'(y) = N(y) - 1$ и $d'(x) = d(x) - 1$ получаем, что теорема верна для вершин G_n , отличных от вершин 1 и n . Для вершин с номерами 1 и n проверка условий теоремы тривиальна.

СЛЕДСТВИЕ 1. Вершины графа G_n , у которых $d(x) > [\frac{n}{2}]$ попарно смежны, а вершины, у которых $d(x) < [\frac{n}{2}]$ попарно не смежны.

СЛЕДСТВИЕ 2. Однородные вершины в графе G_n смежны, если n — четное число, и не смежны, если n — нечетное.

СЛЕДСТВИЕ 3. Существует единственный неидентичный автоморфизм графа G_n , переводящий однородные вершины друг в друга, а остальные вершины в себя.

СЛЕДСТВИЕ 4. Число ребер в графе G_n равно $\left[\frac{n^2}{4} \right]$.

ОПРЕДЕЛЕНИЕ. Несамопересекающаяся цепь называется гамильтоновой, если она содержит все вершины графа.

ТЕОРЕМА 2. В графе G_n существует гамильтонова цепь.

ДОКАЗАТЕЛЬСТВО. Согласно теореме I в [2] в графе G_n любые две вершины x и y смежны, если $N(x) + N(y) > n$. Рассмотрим подграф G'_n графа G_n такой, что ребро (x, y) принадлежит G'_n тогда и только тогда, когда $N(x) + N(y) = n+1$ или $N(x) + N(y) = n+2$. Из этого видно, что каждая вершина x графа G'_n смежна не более, чем с двумя вершинами G'_n , имеющими номера $n+1-N(x)$ и $n+2-N(x)$. Если $N(x) = 1$, то x смежна только одной вершине с номером n . Если $N(x) = n+1-N(x)$ или $N(x) = n+2-N(x)$, т.е. если $N(x) = \left[\frac{n+2}{2} \right]$, то x также смежна только с одной вершиной. Любая вершина x , для которой $N(x) \neq 1$ и $N(x) \neq \left[\frac{n+2}{2} \right]$ имеет степень равную 2. Следовательно, G'_n является цепью, соединяющей вершины с номерами 1 и $\left[\frac{n+2}{2} \right]$ и содержащей все вершины графа G_n .

ТЕОРЕМА 3. Если из графа G_n ($n > 3$) удалить все ребра, принадлежащие гамильтоновой цепи и возникающие при этом изолированные вершины, то полученный граф будет изоморфен графу G_{n-2} .

ДОКАЗАТЕЛЬСТВО. Пусть G' обозначает граф, полученный из G_n удалением ребер гамильтоновой цепи G'_n и изолированных вершин. Так как из G_n выбрасываются ребра (x, y) соответствующие равенствам $N(x) + N(y) = n+1$ и $N(x) + N(y) = n+2$, то для ребер (x, y) , принадлежащих G_n , но не принадлежащих G'_n , будет иметь место неравенство $N(x) + N(y) > n+2$. После выбрасывания из G_n ребер G'_n две вершины с номерами 1 и 2 становятся изолированными. Удаляя изолированные вершины, уменьшим нумерацию остальных вершин на 2, т.е. номер вершины x в

G' будет $N'(x) = N(x) - 2$. Тогда неравенство $N(x) + N(y) > n + 2$ в новой нумерации переходит в неравенство $N'(x) + N'(y) > n - 2$. Именно это неравенство определяет ребра графа G' , а так как G' имеет $n - 2$ вершин, то это согласно теореме I в [2] доказывает изоморфность графа G' графу G_{n-2} .

СЛЕДСТВИЕ. Граф G_n можно разложить на последовательность из $\lfloor \frac{n}{2} \rfloor$ цепочек четной (при n нечетном) или нечетной (при n четном) длины.

ТЕОРЕМА 4. Хроматический класс графа G_n равен $n - 1$.

ДОКАЗАТЕЛЬСТВО. Утверждение теоремы является следствием из определения G_n и теоремы Визинга [3] для любого четного n .

Пусть n - нечетное, т.е. $n = 2k + 1$. Раскрасим гамильтонову цепь графа G_n правильно двумя цветами a, b . По теореме 3 нераскрашенные ребра принадлежат графу G_{n-2} . Его гамильтонову цепь раскрасим правильно двумя другими цветами c, d . И так далее, согласно следствиям теоремы 3, для раскраски графа G_n , т.е. $\lfloor \frac{n}{2} \rfloor$ цепочек понадобится $2 \lfloor \frac{n}{2} \rfloor = 2 \lfloor \frac{2k+1}{2} \rfloor = 2k = n - 1$ цвета, т.е. хроматический класс G_n равен $n - 1$.

Выражаю благодарность Э.Я.Гринбергу за постоянное внимание к работе и В.К.Титову за полезные дискуссии и ценные советы.

ЛИТЕРАТУРА

1. Behzad M, Chartrand G. No graphs is perfect. The American Mathematical Monthly, vol 47, №8, 1967, p.962-963.
2. Nebecky L. On connected graphs containing exactly two points of the same degree. Casopis pro pestovani matematiky, roc, 98, 1973, p. 305-306.
3. Зыков А.А. Теория конечных графов. I. Новосибирск, 1969.

СОДЕРЖАНИЕ

1. А.Я.Абеле, М.В.Витиньш. Система отладки программ АСОМ-ПЕНС	3
2. Э.А.Богомолс, Г.В.Лучкин. Страничная организация памяти на магнитных дисках	27
3. Х.К.Буш. Синтез и применение алгоритмов, генерирующих задания в ДОС/ЕС	43
4. Л.М.Васерман. Организация ввода и накопления данных в АСОМ	58
5. С.А.Иванова. Особенности работы с индексно-последовательными файлами в ДОС/ЕС	67
6. Х.И.Калов. Расширение возможностей Дисковой операционной системы при программировании на PL/I и Ассемблере	89
7. Г.В.Лучкин. Архитектура системы банка данных	103
8. П.Б.Кикуст. О построении циклических графов	120
9. А.О.Кац. Некоторые свойства вполне неоднородных графов	128

Ученые записки, том 242

РАЗРАБОТКА И ВНЕДРЕНИЕ АВТОМАТИЗИРОВАННЫХ СИСТЕМ

Выпуск I

Редактор Л.Поспелов
Технический редактор Р.Лиешниене
Корректор Р.Лиешниене

Латвийский государственный университет
Рига 1975

Подписано к печати 16.10.1975 ЯТ 12360 Зак. № 1207.
Ф/б 60x84/16. Бумага М1. Физ.п.л. 8,5. Уч.-изд.л. 6,3
Тираж 500 экз. Цена 63 к.

Отпечатано на ротационте, Рига-50, ул.Вейдегбаума, 5
Латвийский государственный университет им. П.Стучки