Министерство высшего и среднего специального образования Р С Ф С Р

Куйбышевский ордена Трудового Красного Знамени авиационный институт имени С.П.Королева

М.А. К о р а б л и н

П Р О Г Р А М М И Р О В А Н И Е М О Д Е Л Е Й Д И С К Р Е Т Н Ы Х С И С Т Е М Н А Я З Ы К Е *GPSS*

Учебное пособие

Куйбышев 1981

УДК 519:95

К о р а б л и н М.А. Программирование моделей дискретных систем на языке *GPSS* .Куйбышев:КуАИ,1981,72 с.

Настоящее учебное пособие посвящено методам разработки программ моделирования на языке *GPSS (General Purpose System Simulator)*, который активно используется у нас в стране и за рубежом для цифровой имитации сложных автоматизированных и вычислительных систем, систем обработки данных и массового обслуживания на ЭВМ. Изложение материала сопровождается большим количеством иллюстративных примеров.

Пособие предназначено для студентов специальностей 0646, 0647, изучающих курс "Моделирование систем на ЭВМ", а также выполняющих курсовые и дипломные работы, связанные с цифровой имитацией сложных систем.

Темплан 1981 г. поз. 691.

Рецензенты: Б.К.Б р ю х а н о в, Д.Д.К л о в с к и й

Утверждено на редакционно-издательском совете института 28.II.79.

I. В В Е Д Е Н И Е

Язык *GPSS (General Purpose System Simulator)* разработан в США и с 1962 года входит в стандартное математическое обеспечение машин серии IBM 360/370 [1]. В СССР имеются варианты *GPSS* в ОС и ДОС ЕС, нередко *GPSS* называют ПИДС – пакет для моделирования дискретных систем [2], ОЦСМ – общецелевая система моделирования [3], *SIMDIS* – дискретное моделирование и т.д. В этом пособии мы будем придерживаться первоначального названия – *GPSS*. Следует отметить, что настоящее пособие содержит лишь самые основные сведения о языке, при необходимости более детального знакомства с *GPSS* следует обратиться к полному описанию языка [2,7].

Изложение материала построено таким образом, что по мере получения сведений о тех или иных формальных конструкциях языка их использование иллюстрируется примерами. Вначале эти примеры просты, постепенно по мере знакомства с языком они усложняются. Такой стиль изложения вносит свою специфику: приводимые примеры во многом определяют последовательность описания тех языковых конструкций, которые в них используются. С точки зрения формальной логики и классификации элементов языка, это не традиционный подход, однако, на наш взгляд, такое изложение материала способствует лучшему усвоению формализмов *GPSS* студенческой аудиторией.

Язык *GPSS* предназначен для разработки и построения моделей систем, которые относятся к классу дискретных. К таким системам можно отнести многочисленные системы массового обслуживания (СМО), вычислительные системы, разного рода автоматизированные системы. Язык основан на схеме транзактов (сообщений). Под транзактом понимается формальный объект, который "путешествует" по системе, встречая на своем пути всевозможные задержки, вызванные занятостью тех или иных единиц оборудования. В качестве транзакта может высту-

пать, например, программа обработки информации (при моделировании вычислительных систем (ВС), корабль, приведший в порт на разгрузку, отказ системы при исследовании надежности и т.д., т.е. в понятие транзакта можно вложить самое различное смысловое содержание. Каждый транзакт обладает совокупностью параметров (до 100), которые называются атрибутами транзакта. В процессе имитации атрибуты могут меняться в соответствии с логикой работы исследуемой системы.

Язык *GPSS* – это язык интерпретируемого типа, он связан с пошаговым выполнением инструкций каждого отдельно взятого оператора. Имеет блочную организацию, т.е. управление осуществляется с помощью инструкций, называемых блоками. Языку свойственна двойная кодировка программы, а именно: в виде инструкций, кодируемых в последствии на перфокартах и в виде блок-схем. (В этом пособии изображение програми в виде блок-схем не рассматривается).

Правильное течение времени в модели поддерживается автоматически с помощью комплекта управляющих программ, называемого симулятором. Время, как и все параметры и атрибуты объектов языка, – величина целочисленная, что требует от пользователя правильного и рационального определения ее масштаба.

Управляющие операторы или блоки языка выполняют следующие основные функции:

1. Создание и уничтожение транзактов.
2. Изменение их атрибутов.
3. Задержка транзактов.
4. Изменение маршрутов транзакта в системе.

Различают следующие основные группы элементов языка:

1. Элементы, имитирующие единицы оборудования системы (устройства, памяти и логические переключатели).
2. Статистические элементы: очереди, таблицы.
3. Вычислительные элементы.
4. Списки.
5. Прочие элементы.

Рассмотрим некоторые из элементов языка.

У с т р о й с т в о имитирует единицу оборудования, которое может одновременно обрабатывать только одно сообщение, один транзакт.

П а м я т ь имитирует единицу оборудования, в которой могут обрабатываться (храниться) несколько транзактов одновременно.

О ч е р е д ь - статистический элемент, связанный со сбором статистики о задержках, возникающих на пути прохождения транзакта.

Т а б л и ц ы определяют форму вывода статистической информации.

Каждый из этих элементов управляется своей совокупностью блоков.

При моделировании вычислительных систем устройство может имитировать, например, центральный процессор ЭВМ, канал ввода-вывода и т.п. Память имитирует ту или иную память вычислительной системы.

Кроме отмеченных основных групп, в языке имеется еще целый ряд элементов: блоки, транзакты, матрицы ячеек, группы и т.д. С некоторыми из них мы познакомимся в процессе изложения материала. Любая программа на *GPSS* связана с созданием транзактов, проведением их через последовательность блоков и уничтожением транзактов. При этом создание или генерация транзактов основывается на знании закономерностей информационных потоков, циркулирующих в моделируемой системе, а путь прохождения транзакта через блоки определяется спецификой работы оборудования исследуемой системы.

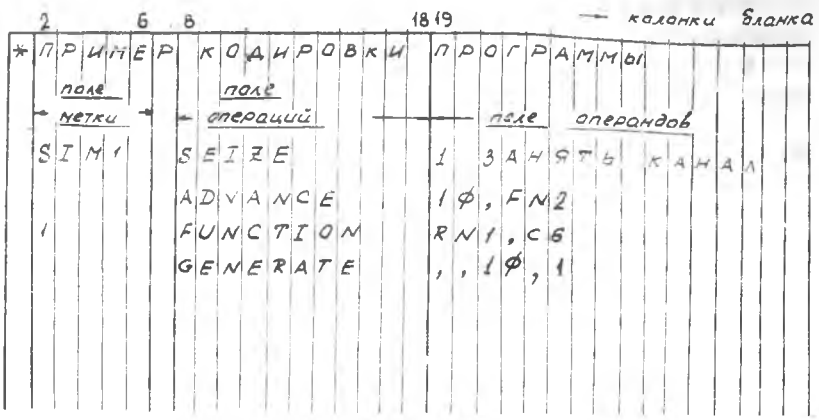
Иногда транзакт называют сообщением, чаще всего эта терминология используется применительно к моделям систем обработки данных. В дальнейшем мы будем использовать и тот и другой термин. Вложить в рамки формальной схемы *GPSS* конкретное смысловое содержание, определяемое исследуемой системой, - задача всегда непростая: для этого необходимо знать как формализмы языка, так и логику работы моделируемой системы. Но тем не менее программирование на *GPSS* существенно облегчает пользователю процесс моделирования, сокращая и время чистого программирования (по сравнению, например, с универсальными языками типа ПЛ/I или *FORTRAN*), и время отладки программы.

2. ВХОДНОЙ ФОРМАТ ПРОГРАММЫ

Программа на *GPSS* имеет зонную организацию, т.е. инструкции языка пишутся в определенных полях бланка кодировки программы (рис I).

В поле метки пишется либо номер, либо символическое имя блока или другого элемента языка.

В поле операций пишется имя блока, начиная с 8-й колонки.



Р и с. I. Бланк кодировки программы

В поле операндов (с I9-ой колонки по 72-ую) пишутся операнды. Каждая перфокарта содержит одну инструкцию языка.

Поле операндов состоит из 7 подполей: A, B, C, *D* E, *F* , *G* - содержимое которых отделяется друг от друга запятой. Поле операндов заполняется без пробелов, текст после первого, встреченного транслятором пробела, воспринимается системой как комментарий. Кроме того, любой текст при наличии в I-ой колонке символа " * " также будет восприниматься как комментарий. Если для какого-либо блока одно из подполей поля операндов необходимо опустить, пробел все равно не ставится, вместо него ставится запятая: ",". Например, *GENERATE„ 1ф,1* . В этой конструкции используются поля *A, B, C, D* , причем поля *A* и *B* опущены.

3. СОЗДАНИЕ И УНИЧТОЖЕНИЕ ТРАНЗАКТОВ

С созданием и уничтожением транзактов в языке связано 2 типа блоков:

Генерирование - *GENERATE* . Этот блок генерирует поток сообщений - транзактов, поступающих в систему. Программа составляется с учетом того обстоятельства, что в блок *GENERATE* не могут

входить какие-либо транзакты. В простых программах *GENERATE* - обычно первый блок программы.

Временные интервалы между поступающими в систему транзактами определяются содержимым поля операндов. При этом подполя имеют следующий смысл:

A - среднее время между поступлениями транзакта в систему. Если поле A пусто, среднее время равно I.

B - модификатор времени.

C - начальная задержка (в этом поле указывается время появления I-го транзакта).

D - общее число транзактов, которое должно быть сгенерировано этим блоком. Если поле *D* пусто, блок выдает неограниченное число транзактов.

E - приоритет транзакта, может быть 0-127. Приоритет возрастает в соответствии с номером. Если поле E пусто, то сообщению присваивается "нулевой" приоритет.

F - число параметров (атрибутов) транзакта (0-100). Если поле пустое, то каждому транзакту присваивается 12 параметров по умолчанию.

G - пишется код либо *F* , либо H. Если *F* , транслятор каждому параметру резервирует в памяти слово, если H - полуслово. Если ничего не написано - резервируется полуслово.

В поле B может быть модификатор двух типов: модификатор-интервал и модификатор-функция.

Модификатор-интервал определяет целочисленную величину, принимающую равновероятные значения в определенном диапазоне. Например, в конструкции *GENERATE* 10,5 интервал времени между транзактами - целочисленная случайная величина, равномерно распределенная в диапазоне [10-5, 10+5]. При использовании модификатора-интервала необходимо следить, чтобы содержимое поля B было меньше содержимого поля A (B < A) так, чтобы время не могло принимать отрицательные значения. В противном случае транслятор выдает сообщение об ошибке.

При использовании модификатора-функции интервал времени между транзактами, поступающими в систему, определяется произведением содержимого полей A и B, например, *GENERATE 10,FN2* . Интервал времени T=10·*FN2* . Функция определяется специальными картами языка. В этом примере *FN2* - обозначение (стандартный числовой атрибут) функции с номером 2.

Блок уничтожения транзактов - *TERMINATE* . Обычно для простых программ - это последний блок программы. Транзакты, попадающие в этот блок, уничтожаются и больше не участвуют в процессе моделирования. При этом при поступлении каждого транзакта в блок *TERMINATE* из специального счетчика транзактов вычитается величина, указанная в поле А блока *TERMINATE* .

Первоначальная величина счетчика устанавливается специальной управляющей картой *START* и пишется в поле А этой карты.

Если в поле А блока *TERMINATE* ничего не написано, счетчик не уменьшается. При этом моделирование продолжается до тех пор, пока программа не будет снята оператором.

Например, конструкция

```
TERMINATE  1
START      1ФФ
```

обеспечивает такую длительность моделирования, при которой через программу модели пропускается 100 транзактов.

4. РАБОТА С УСТРОЙСТВАМИ, ЗАДЕРЖКА СООБЩЕНИЙ, ОЧЕРЕДИ

Устройства - один из типов элементов языка, которые имитируют работу оборудования системы (каналов, устройств ввода-вывода, процессоров и т.п.). В любой момент времени устройство может обслуживать только один транзакт. Работа с устройствами в *GPSS* осуществляется несколькими блоками, мы рассмотрим только 2 основных.

Блок *SEIZE* - занять устройство. В поле А этого блока пишется номер устройства, которое транзакт (сообщение) должен занять. При этом устройство, занятое сообщением, не освободится до тех пор, пока соответствующее сообщение не попадает в блок *RELEASE* .

Блок *RELEASE* - освободить устройство. В поле А этого блока указывается также номер устройства.

Задержка во времени, связанная с занятием устройства имитируется в программе блоком *ADVANCE* , для которого в полях А и В указываются соответственно среднее время занятия устройства и модификатор времени. Использование модификатора аналогично блоку *GENERATE*.

Например,

```
SEIZE    1
ADVANCE 10.5
RELEASE 1
```

В этом фрагменте симулятор не разрешит сообщению, поступившему в блок *SEIZE* и занявшему таким образом устройство с номером I, попасть в блок *RELEASE* и освободить это устройство до тех пор, пока не истечет необходимый отрезок системного времени, устанавливаемый блоком *ADVANCE* . Поскольку конструкция *ADVANCE* 10, 5 обеспечивает задержку транзакта на случайное время, принадлежащее диапазону [10-5, 10+5], устройство с номером I окажется занятым на это время. При поступлении транзакта в блок *SEIZE* устройство I занимается (если оно было к этому моменту времени свободным), транзакт продвигается в блок *ADVANCE* , в котором он задерживается на определенное время. По истечении этого времени он поступает в блок *RELEASE* , освобождает устройство I и продолжает свое дальнейшее продвижение по программе модели.

Если же транзакт поступает в блок *SEIZE* в момент занятости устройства I (обслуживание другого транзакта), он не войдет в этот блок до тех пор, пока устройство I не освободится. Таким образом перед блоком *SEIZE* может образовываться как бы "пробка"-очередь, препятствующая прохождению других транзактов - сообщений через модель систем. При необходимости можно собрать статистику об этой очереди.

Для сбора такой статистики следует использовать 2 блока работы с очередями:

поставить в очередь - *QUEUE* ,

вывести из очереди - *DEPART* .

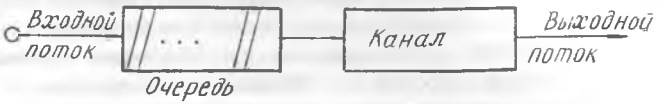
В полях A этих блоков пишется номер очереди. Если поле B этих блоков пусто, при попадании транзакта соответственно в *QUEUE* и *DEPART* счетчик очереди увеличится или уменьшится на единицу. Если поле B не пусто, то счетчик увеличится или уменьшится на величину содержимого поля B.

В программе на *GPSS* возможно использование символической адресации, что позволяет, например, заменить номера устройств символическими именами:

```
SEIZE    CHAN
RELEASE  CHAN
```

5. МОДЕЛЬ ОДНОКАНАЛЬНОЙ СМО С ЦЕЛОЧИСЛЕННЫМ ВХОДНЫМ ПОТОКОМ

Структура рассматриваемой СМО изображена на рис. 2. Входной



Р и с. 2. Одноканальная СМО

поток сообщений-заявок поступает на канал обслуживания, перед которым может образовываться очередь заявок. Обслуженные в канале заявки покидают систему.

Ниже приведен фрагмент программы на *GPSS* , описывающий простейшую модель данной системы массового обслуживания. В этом фрагменте использованы все введенные выше конструкции языка, связанные с генерацией сообщений - транзактов, занятием и освобождением канала и уничтожением транзактов. Блоком *GENERATE* в этом примере генерируется целочисленный входной поток заявок в СМО, причем, время поступления первой заявки в систему равно 2, а интервалы времени между последующими заявками суть равновероятные целочисленные значения, принадлежащие диапазону [5,15]. Блоки *QUEUE* и *DEPART* позволяют собрать статистику об очереди заявок, которая может возникнуть перед каналом. Время обслуживания заявки в канале равно 4 единицам. Оно устанавливается в блоке *ADVANCE* .Это один из блоков, в котором заявка СМО-транзакт задерживается на определенное время. Кроме того транзакт в этом фрагменте может задержаться перед блоком *SEIZE* по причине занятости канала обслуживанием предыдущей заявки.

```
SIMULATE
GENERATE 10,5,2
QUEUE    1
SEIZE    CHAN
DEPART   1
ADVANCE  4
RELEASE  CHAN
TERMINATE 1
START    100
```

Карта *START* инициирует начало моделирования. В поле А этой карты, как уже отмечалось, указывается количество прогонов программы или количество транзактов, проходящих через систему. В этом примере через модель СМО проводится 100 заявок.

Карта *SIMULATE* относится к управляющим картам. Без этой карты осуществляется только трансляция программы, но не происходит ее выполнение.

6. ВЫЧИСЛИТЕЛЬНЫЕ ЭЛЕМЕНТЫ ЯЗЫКА: ФУНКЦИИ

Одним из вычислительных элементов языка является функция. Функция описывается картой описания функции *FUNCTION* , у которой в поле метки стоит номер функции или ее имя.

В поле А операндов указывается аргумент. В поле В указывается тип функции и количество пар аргументов и значений.

Будем рассматривать функции 2-х типов С и *D* .

Функции типа С - непрерывны, *D* - дискретны. Например, если в поле В стоит С12, это означает, что функция непрерывна и у нее будет использоваться 12 пар аргумент-функция:

1 FUNCTION RN1, C12.

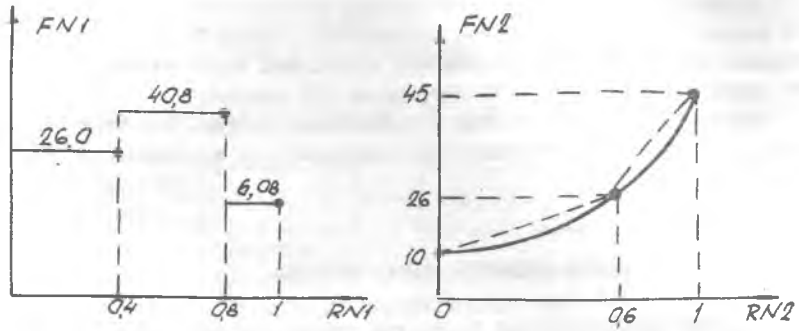
При описании любой из функций с помощью вычислительных объектов языка *GPSS* происходит интерполяция. При указании типа функции *D* - это кусочно-постоянная интерполяция. При указании С -линейная интерполяция.

Координаты функции, задаваемые парами, являются узлами интерполяции.

Ниже, на рис. 3 приведены графики двух функций *FN1* и *FN2*, *FN1* - дискретна, она должна быть описана типом *D* , *FN2* - непрерывна, ее следует описать типом С. Однако следует помнить, что при описании *FN2* типом С произойдет ее линейная интерполяция в соответствии с заданными узлами интерполяции.

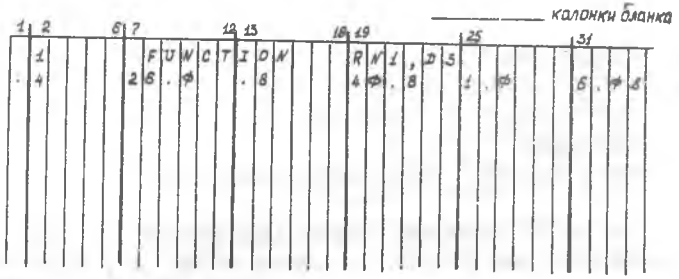
За картой описания функции всегда следует карта задания функции. Карты задания могут быть оформлены в двух форматах: фиксированном (зонном) и свободном.

При использовании фиксированного формата координаты и значения функции располагаются в определенных полях перфокарты.



Р и с. 3. Типы функций

При использовании свободного формата - в любых полях, начиная с I-го, каждая пара отделяется друг от друга наклонной чертой, пробелы недопустимы. В паре аргумент-функция аргумент отделяется от значения функции запятой. Ниже описана в фиксированном формате функция *FN1* (см.рис. 3):



В свободном формате ее описание будет выглядеть таким образом:

I *FUNCTION RN1, D3.*
.4,26.Ø/.8,4Ø.8/I.Ø,6.Ø8 (без пробелов, начиная с I-й колонки)

Карты описания и задания функции располагаются в начале колоды до блока *GENERATE* . Координаты точек функции записываются как чис-

ла с плавающей запятой. Поскольку язык оперирует с целочисленными величинами, любое значение функции округляется до целого числа. (Исключением из этого правила являются блоки *GENERATE* и *ADVANCE*, использующие в поле B модификатор-функцию. В этом случае округление до целого происходит после умножения значения соответствующей функции на содержимое поля A этих блоков).

Функция *FN2* (см.рис.3) в свободном формате будет описываться следующим образом:

```
2 FUNCTION RN2,C3
Ø, IØ/.6, 26/I, 45
```

7. ГЕНЕРАТОР СЛУЧАЙНЫХ ЧИСЕЛ

В языке существуют 8 датчиков равномерно распределенных случайных величин, мнемоническое обозначение которых в программе *RN1 – RN8* . Эти датчики выдают равновероятные целочисленные значения из диапазона 0 – 999. Если какой-либо из датчиков используется в качестве аргумента функции, то он выдает числа от 0 до I, как, например, в приведенных выше функциях *FN1* , *FN2* .

8. ИМИТАЦИЯ ПРОСТЕЙШЕГО ИНФОРМАЦИОННОГО ПОТОКА

В простейшем потоке интервалы времени между поступлениями сообщений в систему распределены экспоненциально с плотностью вероятности [4] $f(x)=\lambda e^{-\lambda x}$, где λ – интенсивность потока.

При использовании метода обратной функции [4] алгоритм получения реализаций величины x выглядит следующим образом:

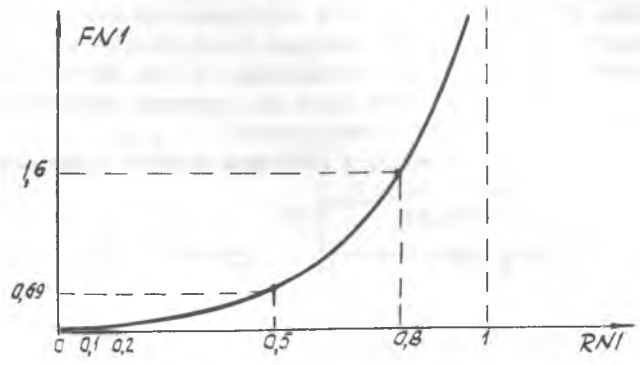
$$x=-\frac{1}{\lambda}\ln(1-R)=-\bar{x}\ln(1-R).$$

Здесь R – равномерно распределенная на [0,I] случайная величина, λ – интенсивность потока, $\bar{x}$ – среднее значение случайной величины x .

Если положить $\bar{x}$ = I, то нетрудно определить функцию, необходимую для имитации простейшего потока с единичной интенсивностью:

$FN1=-\ln(1-RN1)$.

График этой функции приведен на рис. 4.



Р и с. 4. Модификатор-функция в имитаторе простейшего потока

Карты описания и задания функции (при использовании узлов интерполяции) будут выглядеть следующим образом:

```
I  FUNCTION   RN1, C6
0, 0/.1,. 1/.2,.2/.5,.69/
.8,1.6/.999,8.0
```

Для того, чтобы смоделировать простейший поток, нужно к этим картам добавить блок, который будет генерировать транзакты в соответствии с заданной функцией:

```
GENERATE  10, FN1.
```

В этой конструкции среднее время между транзактами равно 10, а интервалы времени между поступлениями их в систему распределены по экспоненциальному закону.

9. ИСПОЛЬЗОВАНИЕ ИДЕНТИФИКАТОРОВ

При составлении сложных программ те или иные объекты языка удобно идентифицировать не с помощью номера, а с помощью некоторого имени или названия. С этой целью в языке разрешается вводить

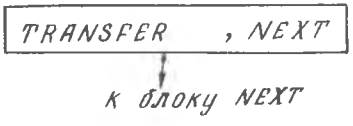
идентификаторы. Идентификатор может содержать от 3 до 5 алфавитно-цифровых символов, первые 3 из которых буквы. При этом идентификатор должен быть хотя бы один раз употреблен в поле метки. При использовании идентификатора в поле операндов он отделяется от текста символом ☆

```
EXPON  FUNCTION  RN1, C6
.............  карты описания
       GENERATE  1Ф, FN☆EXPON
```

IO. ИЗМЕНЕНИЕ МАРШРУТОВ СООБЩЕНИЙ

Основным блоком, который позволяет изменить направление пути сообщения в системе является блок *TRANSFER* , который дает возможность осуществить безусловные, условные и статистические переходы. Тип перехода определяется мнемокодом, записанным в поле А. Направление перехода определяется содержимым полей В и С.

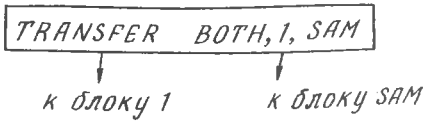
Безусловный переход. При безусловном переходе поле А пусто. В поле *B* указывается имя или номер блока к которому должен направится транзакт.



Если блок, к которому направляется транзакт, в текущий момент системного времени не может его принять (например, *SEIZE*),транзакт остается в блоке *TRANSFER* и повторяет попытку перехода при каждом пересчете системного времени симулятором. Такая организация увеличивает длительность процедуры моделирования.

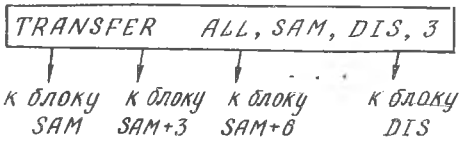
Условный переход. При условном переходе в поле А блока *TRANSFER* могут быть записаны различные мнемокоды, самый простой режим определяется мнемокодом BOTH, в поле В пишется номер или идентификатор блока, к которому направляется транзакт в I-ю очередь, в поле С также стоит номер или идентификатор блока, к кото-

рому транзакт направляется во 2-ю очередь:



Если блок I занят, то транзакт направляется к *SAM* . Если оба заняты, то транзакт остается в блоке *TRANSFER* и повторяет попытки перехода при каждом пересчете системного времени.

Обобщением этого блока является *TRANSFER ALL*



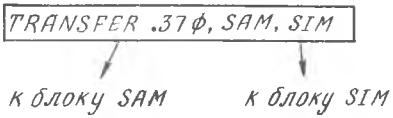
В поле B пишется идентификатор I-го блока, в который пытается войти транзакт, в поле C - идентификатор последнего блока, в поле *D* пишется интервал приращения номеров блоков.При этом должно выполняться условие:

$№ DIS = № SAM + M \cdot 3 \; ; \quad M > 0 - ц$

(№ - номер блока)

При использовании этой конструкции транзакт пытается войти по очереди в *SAM, SAM+3, ..., DIS* и занимает I-ый свободный блок. Если ни один из этих блоков не может принять транзакт, то он остается в *TRANSFER* и повторяет попытку перехода при каждом изменении системного времени.

Статистический переход. В этом режиме направление пути транзакта определяется не условиями занятости оборудования, а некоторым вероятностным механизмом, основанным на случайном выборе пути дальнейшего продвижения транзакта. В режиме статистического перехода в поле А пишется десятичная дробь - вероятность перехода транзакта к блоку, указанному в поле С. При этом переход транзакта к блоку, указанному в поле В будет происходить с вероятностью (I - <A>), где <A> - содержимое поля А.



Вероятность перехода к блоку *SIM* будет указываться содержимым в поле А, а к блоку *SAM* - величиной, дополненной до I.

$$P\{\rightarrow SIM\} = 0{,}370$$

$$P\{\rightarrow SAM\} = 0{,}630$$

Если оба блока заняты, то транзакт остается в блоке *TRANSFER* и повторяет попытку перехода к выбранному ранее блоку при каждом изменении системного времени.

Существует и ряд других режимов работы блока *TRANSFER* , два из них рассматриваются в п. 29.

II. МОДЕЛЬ МНОГОКАНАЛЬНОЙ СИСТЕМЫ МАССОВОГО ОБСЛУЖИВАНИЯ С ОБЩЕЙ ОЧЕРЕДЬЮ ВХОДНЫХ СООБЩЕНИЙ И ЗАНЯТИЕМ I-го СВОБОДНОГО КАНАЛА

Ниже на рис. 5 приведена структурная схема рассматриваемой СМО.

Р и с. 5. Многоканальная СМО

Входной поток заявок - сообщений поступает на вход системы, при этом каждая заявка претендует на обслуживание в любом из каналов. Если все каналы уже заняты обслуживанием, заявка ставится в общую очередь (с дисциплиной *FIFO* [4]) и ожидает, пока система сможет заняться ее обслуживанием. Обслуженные заявки образуют выходной поток информации.

Подобная структура СМО распространена при решении целого ряда прикладных задач. В отдельных случаях многоканальные СМО могут быть исследованы и аналитически [4], однако общей методологией исследования подобных систем является имитация.

Описание на *GPSS* многоканальной СМО приводится ниже. В этом фрагменте не заполнены поля операндов блоков *GENERATE* и *ADVANCE*, поскольку мы не конкретизируем здесь ни дисциплину входного потока, ни дисциплину обслуживания заявки в каналах - они могут быть самыми различными. Причем время обслуживания заявки в канале может быть одним и тем же для всех каналов, а может быть в каждом канале своим собственным.

```
          SIMULATE
          GENERATE   ...
          QUEUE      1
          TRANSFER   ALL, CHAN1, CHAN5,5
CHAN1     SEIZE      1
          DEPART     1
          ADVANCE    ...
          RELEASE    1
          TRANSFER   ,ENDM
CHAN2     SEIZE      2
          DEPART     1
          ADVANCE    ...
          RELEASE    2
          TRANSFER   ,ENDM
. . . . . . . . . . . . . . . . . .
CHAN5     SEIZE      5
          DEPART     1
          ADVANCE    ...
          RELEASE    5
ENDM      TERMINATE  1
          START      100
```

Как следует из этого простого фрагмента, каналы СМО отождествляются при имитации системы с устройствами *GPSS* , номера которых I-5. Кроме того, в этом фрагменте используется один статистический элемент языка - очередь с номером I. Этот пример иллюстрирует одно из возможных применений конструкции *TRANSFER ALL*.

I2. РАБОТА С ПАМЯТЬЮ

Памяти - особый тип элементов языка *GPSS* , который призван имитировать разного рода накопители, используемые в исследуемых системах: запоминающие устройства (ЗУ), концентраторы и т.д., в которых могут одновременно находиться несколько сообщений-транзактов. Любая память описывается в программе картой описания памяти *STORAGE* . В поле метки этой карты пишется номер или имя памяти, в поле А указывается объем памяти в условных единицах:

MEMOR STORAGE 100

Существует 2 основных блока управления памятью:

блок - ввести сообщение в память *ENTER*

и блок - вывести сообщение из памяти *LEAVE*.

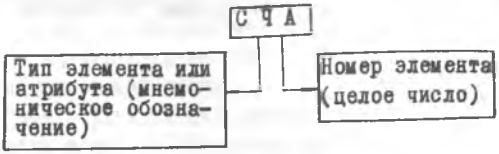
В полях А этих блоков пишутся либо номер, либо идентификатор соответствующей памяти, описанной картой *STORAGE* .

В поле В указывается число единиц памяти, занимаемых сообщениями при входе в память, или освобождаемых при выводе сообщения из памяти.

I3. СТАНДАРТНЫЕ ЧИСЛОВЫЕ АТРИБУТЫ ЯЗЫКА (СЧА)

Существует несколько разновидностей элементов языка, таких, как устройства, память, статистические элементы (очереди), вычислительные элементы (функции) и т.д. Каждый из этих элементов описывается своими стандартными числовыми атрибутами (СЧА), которые однозначно определяют статус этого элемента в модели системы. СЧА меняются в процессе имитации, изменить их может как симулятор, так и пользователь.

СЧА могут быть использованы в качестве аргументов поля операндов практически любых блоков языка. СЧА состоит из мнемонического обозначения типа элемента или его атрибутов и целого числа - номера элемента:



Ниже приводится краткая таблица (табл. I), в которой перечислены некоторые СЧА основных элементов языка.

Таблица I

Тип элемента	Основные СЧА
Сообщение (транзакт)	P_i - значение i -го параметра транзакта PR - приоритет транзакта $M1$ - время пребывания сообщения в модели
Устройство	Fi - состояние i -го устройства (F_i = 0, - i -ое устройство свободно, I - занято)
Очередь	Q_i - текущая длина очереди с номером i QMi - max длина очереди с номером i
Память	Si - заполненный объем памяти с номером i Ri - свободный объем памяти с номером i
Блок	Ni - число сообщений, вошедших в блок i с начала моделирования
Функция	FN_i - значение функции с номером i
Системные атрибуты	RN_i - значение случайного числа, получаемого от датчика с номером i K_i - константа, значение которой равно i $C1$ - системное время

Системные СЧА, такие, как С1 или RNi , не могут быть изменены пользователем, они меняются только симулятором.

Системный атрибут - константа может иметь два обозначения: *Ki* или просто *i* . Например блоки

ADVANCE K1ФФ
и
ADVANCE 1ФФ.

совершенно идентичны. Каждый из них обеспечивает задержку поступающего транзакта на 100 единиц системного времени.

СЧА активно используются в полях операндов различных блоков языка. Например, *SEIZE FN1* - занять устройство, номер которого определяется значением функции *FN1* .

Или: *ADVANCE P3* - задержать транзакт на время, определяемое значением третьего параметра этого транзакта. *LEAVE 1,S1* - освободить память с номером I полностью.

Приведенная выше таблица содержит далеко не полный перечень элементов языка и их СЧА, с некоторыми дополнительными типами элементов мы встретимся в рассматриваемых далее примерах.

14. ИЗМЕНЕНИЕ МАРШРУТОВ СООБЩЕНИЯ: БЛОКИ *GATE* И *TEST*

Эти 2 типа блоков связаны с изменением маршрутов сообщения в программе в зависимости от выполнения тех или иных логических условий.

Блоки *GATE* и *TEST* отличаются друг от друга тем, что первый из них изменяет путь сообщения только в зависимости от состояния моделируемого оборудования, а блок *TEST* изменяет маршрут сообщения в зависимости от выполнения самых разнообразных логических условий, определенных на множестве СЧА.

Код этих блоков записывается в программе следующим образом:

3	12	13 14 15		19
GATE		Мнемокод состояния объекта		
TEST		Мнемоника отношения		

Мнемокод состояния объекта определяется в соответствии со следующей таблицей:

Состояние объекта	Мнемокод блока *GATE*
Память:	
пуста	*SE*
не пуста	*SNE*
заполнена	*SF*
не заполнена	*SNF*
Устройство:	
свободно	*NU*
занято	*U*

(Здесь приведены не все возможные мнемокоды).

Блок *GATE* может работать в 2-х режимах: режиме отказа и в режиме условного перехода.

В поле А этого блока пишется номер проверяемой единицы оборудования (проверяемого объекта). В поле В может быть написан номер блока, к которому должен быть направлен транзакт. Если поле В пусто, то блок работает в режиме отказа. Если не пусто, то в режиме условного перехода. В I-ом случае сообщение не может войти в блок *GATE* до тех пор, пока не выполнится условие, записанное в виде мнемокода состояния объекта, т.е. до тех пор, пока данный объект не перейдет в нужное состояние. Как только это произойдет, транзакт направляется к следующему за *GATE* блоку. Если поле В не пусто, *GATE* работает в режиме условного перехода, при этом в случае, если проверяемый объект не находится в требуемом состоянии, транзакты, поступающие на блок *GATE* , направляются к блоку,указанному в поле В, в противном случае, если указанный объект находится в требуемом состоянии, транзакт направляется к следующему за *GATE* блоку.

Например, *GATE SE 1, SIM.*

Эта конструкция работает следующим образом. Если память с номером 1 пуста, транзакт, поступивший в блок *GATE* , проходит дальше по программе модели, если не пуста, он направляется к блоку с именем *SIM* . Здесь используется режим условного перехода.

Блок *TEST* работает во многом аналогично, но в колонках с I3 по I5 перфокарты указывается мнемоника отношения: *L* – < ; *LE* – ≤ ; *E* – = ; *NE* – ≠ ; *G* – > ; *GE* – ≥ .

В полях A и B блока *TEST* пишутся СЧА языка, используемые в проверяемом отношении, а в поле C пишется номер блока к которому пытается перейти транзакт.

Если C – пусто, то блок работает в режиме отказа, если C – не пусто, – в режиме условного перехода.

П р и м е р

TEST E S1, Kф, 4

В этой конструкции используются 2 СЧА: *S1* – текущее содержимое памяти с номером I и *Kф* – константа, определяющая *0* . В том случае, если *S1=0* , т.е. память с номером I пуста, *TEST* передает транзакт следующему блоку по программе. Если же *S1≠0* (память не пуста), транзакт направится к блоку с номером 4.

Блок *TEST* обладает более широкими возможностями, чем *GATE*: в его полях A и B можно писать любые СЧА. Режимы отказа и условного перехода у блоков *TEST* и *GATE* очень схожи.

I5. ВЫЧИСЛИТЕЛЬНЫЕ ОБЪЕКТЫ ЯЗЫКА: ПЕРЕМЕННЫЕ, БЛОК *SAVEVALUE*

Кроме функций, которые были рассмотрены выше, к вычислительным элементам языка относятся еще переменные и ячейки *SAVEVALUE*.

В *GPSS* используются булевские и арифметические переменные. Арифметические переменные описываются картой описания переменной *VARIABLE* или *FVARIABLE* .

В поле операндов этих карт указывается арифметическое сочетание СЧА языка, в котором могут использоваться арифметические операции +,–, * , / и деление по модулю *@* . Значения переменных вычисляются слева направо, умножение и деление выполняется раньше, чем сложение и вычитание. Деление на 0 дает *0* . Все числа без знака считаются положительными. При выполнении операции *@* частное от деления отбрасывается, результатом операции является остаток, который считается положительным).

При использовании карты *VARIABLE* от каждого СЧА берется целая часть, и арифметические операции выполняются над целыми числами. Результат округляется до целого числа.

При использовании *FVARIABLE* целые части от операндов не берутся, а лишь результат всей операции округляется до целого числа.

П р и м е р :

```
1 VARIABLE    1ф(11/3)      результат 30
1 FVARIABLE   1ф(11/3)      результат 36
```

Знак умножения может быть опущен. Допускается использование скобок. СЧА переменной обозначается символом V_i - как для *VARIABLE*, так и для *FVARIABLE*. i - номер переменной, стоящий в поле метки карты описания переменной.

Булевские переменные описываются картой *BVARIABLE*. Для булевских переменных в поле операндов могут быть указаны логические, условные и булевские операции.

Логические операции: мнемокоды логических операций похожи на мнемокоды блока *GATE* (мнемокоды памяти такие же, а к мнемокодам устройств добавляется буква *F* :

```
1 BVARIABLE  FNU1
```

Булевские переменные принимают значение I (истина), если логическое условие, записанное в поле операндов карты описания этой переменной, выполнено.

Так, в этом примере переменная *BV1* принимает значение I, если устройство с номером I свободно (мнемокод *FNU*) (См. описание блока *GATE*). В противном случае *BV1* принимает значение *0* .

Условные операции: мнемоника аналогична мнемонике блока *TEST*, а переменными являются СЧА языка:

```
2 BVARIABLE   V1'G'V2.
```

Условные операторы берутся в кавычки.

В этом примере переменная *BV2* принимает значение I, если переменная *V1>V2* (оператор *G*), и значение 0 в противном случае.

Булевские операции используют знаки "+" и "*", которые обозначают логические "или" и логическое "и".

Возможны в поле операндов комбинации логических, условных и булевских операторов.

П р и м е р :

```
3 BVARIABLE (V2'G'K5)*(FNU2+SF3)
```

BV3 принимает значение "Истина" , когда *V2* > 5 и (устройство с номером 2 не занято или память с номером 3 полна). СЧА булевской переменной: BV_i , где i - номер переменной, указанный в поле метки соответствующей карты описания.

Слежение за выполнением условий, указанных в поле операндов, осуществляется симулятором автоматически.

Ячейки *SAVEVALUE* предназначены для хранения некоторых постоянных И/ИЛИ изменяющихся значений переменных программы. В поле А этой карты указывается номер ячейки, сохраняющей значение и вид изменения этого значения ("+" - накопление, "-" - уменьшение).

В поле В указывается либо СЧА, либо целое число, которое либо добавляется, либо вычитается, либо заменяет содержимое ячейки.

П р и м е р:

SAVEVALUE 1ϕ+, V1

Каждый раз при поступлении транзакта в блок *SAVEVALUE* к содержимому IO-й ячейки прибавится значение переменной *V1*

SAVEVALUE 1ϕ,V1

В этой конструкции при поступлении транзакта в блок *SAVEVALUE* в ячейку IO запишется значение переменной *V1* . *SAVEVALUE* не ставит никаких преград на пути движения транзакта. Чаще всего на базе ячеек организуются разного вида счетчики. Если в поле С стоит код Н, то ячейка занимает полуслово, если С пусто, то - целое слово. СЧА ячеек *SAVEVALUE* обозначается X_i , где i - номер ячейки. Перед началом имитации содержимое всех используемых в программе ячеек *SAVEVALUE* устанавливается равным нулю.

I6. МОДЕЛЬ РАБОТЫ ДВУХТАКТНОГО БЗУ

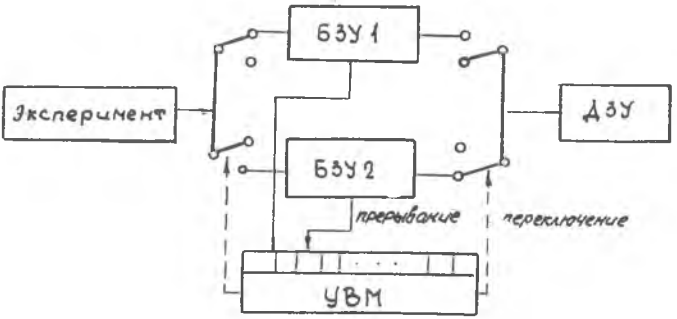
Схема двухтактного буферирования получила широкое распространение на практике при построении самых различных автоматизированных систем обработки данных. Идея двухтактного буфера очень проста: совместить во времени процессы сбора и записи информации в буфер (БЗУ) ограниченного объема и перезаписи информации в долговременную память (ДЗУ), объем которой неограничен.

Технически осуществить двухтактное буферирование можно, например, с использованием канала прямого доступа и программного канала.

Рассмотрим упрощенную модель системы двухтактного буферирования.

Экспериментальная информация в процессе сбора записывается словами в буферную область памяти.

При заполнении приемного БЗУ до определенного уровня генерируется запрос к УВМ на передачу информации в ДЗУ (включается система прерываний). Между моментом возникновения запроса и переключением ключей (рис. 6) проходит определенное время, по истечении которого приемное БЗУ становится передающим (например, подключается канал прямого доступа и информация пишется на диск), а для сбора информации выделяется в ОЗУ новая область, которая становится приемным БЗУ.



Р и с. 6. Структура двухтактного БЗУ

При исследовании такой системы на модели могут ставиться различные вопросы, например: каков должен быть уровень заполнения БЗУ, при котором в системе генерируется запрос на переключение ключей. Как связан этот уровень с интенсивностью потока экспериментальных данных и вероятностью "переполнения" приемного БЗУ за время между запросом и переключением ключей и т.п.

Описание модели этой системы на *GPSS* , приведенное ниже, существенно упрощено: в нем не учитывается целый ряд практически важных деталей. Это объясняется иллюстративным характером обсуждаемого примера.

Допустим, что объем БЗУ - 1024 слова. Разрядность одного слова - 8 единиц. При этом память БЗУ1 или БЗУ2 может быть описана такой конструкцией:

```
2 VARIABLE    1Ø24*8
1 STORAGE     V2
```

Допустим, что запрос на прерывание возникает при заполнении БЗУ до 1020 слов, тогда уровень заполнения может быть описан картой

```
3 VARIABLE    1Ø2Ø*8
```

Введем еще карту описания среднего времени задержки между запросом и переключением БЗУ:

```
4 VARIABLE    1Ø/4
```

Предположим, что поток слов, записываемых в БЗУ, во времени описывается моделью простейшего потока. Тогда описание функционирования этой системы на *GPSS* будет выглядеть следующим образом:

```
          GENERATE      1Ø,FN1
          ENTER         1,8
          GATE  SNF     1,OTKAZ
          TEST  E       S1,V3,ENDM
          ADVANCE       V4,FN1
          LEAVE         1 , S1
          TERMINATE     1
OTKAZ     SAVEVALUE     1Ø+,1
ENDM      TERMINATE     1
```

Комментарии:

Бл.1 – генерация простейшего потока транзактов.

Бл.2 – занятие памяти: каждое слово занимает 8 единиц в памяти с номером 1.

Бл.3 – проверка условия потери информации: в соответствии с мнемоникой *SNF* блок *GATE* , работающий в режиме условного перехода, посылает транзакт на блок OTKAZ в случае, если память *S1* заполнена, и передает его в блок *TEST* в противоположном случае.

Бл.4 – проверка условия генерации запроса: запрос на переключение будет симитирован в блоке *TEST* при условии *S1=V3*, здесь *TEST* работает в режиме условного перехода.

Бл.5,6,7 – перезапись информации в ДЗУ: *ADVANCE* задержит транзакт на некоторое случайное время, имитирующее интервал от возникновения запроса до переключения. *LEAVE* – освободит БЗУ полностью. После этого *S1* транзактов уничтожится блоком *TERMINATE*.

Бл. 8,9 – счетчик слов, потерянных в системе из-за "переполнения" приемного БЗУ.

Для того, чтобы это описание превратилось в программу машинного счета, перед блоком *GENERATE* необходимо поставить все карты описания переменных $V2 \div V4$, памяти *S1* и функции *FN1* (ее мы обсуждали раньше) и снабдить колоду управляющими картами: *SIMULATE* перед началом программы, *START* и *END* в конце программы. (Подробнее см.п. 28 и прил. I.).

I7. ИЗМЕНЕНИЕ ПАРАМЕТРОВ СООБЩЕНИЯ

Каждый транзакт в системе *GPSS* может иметь до I00 параметров. Каждому из параметров транзакта может быть присвоено определенное значение с помощью блока

ASSIGN.

В поле A этого блока указывается номер параметра и вид его изменения $i, i+, i-$.

В поле B указывается в виде целого числа или СЧА значение, присваиваемое этому параметру.

В поле C указывается модификатор в виде целого числа. Это число определяет номер функции, на значение которой умножается содержимое поля B при использовании блока *ASSIGN* с модификацией.

П р и м е р ы: *ASSIGN* I, I0

При вхождении транзакта в блок *ASSIGN* первому параметру присваивается значение, равное I0.

ASSIGN *I, V1, 3*

При вхождении транзакта в блок *ASSIGN* первому параметру присваивается значение *V1* FN3*: *P1:= V1* FN3*. *ASSIGN 1+,V1* В этом случае происходит присвоение

$P1 := P1 + V1$

Транзакт никогда не задерживается в блоке *ASSIGN* .

I8. КОСВЕННАЯ АДРЕСАЦИЯ

Косвенная адресация в *GPSS* используется в сочетании с любым СЧА, кроме текущего времени CI, случайного числа RN_i и времени пребывания сообщения в системе MI.

Эта адресация может использоваться практически в любом типе блоков. Структура косвенной адресации имеет такой вид: СЧА* <целое число (0÷I00)>. Под целым числом подразумевается номер параметра, в котором указан номер соответствующего СЧА: *FN1* – прямая адресация – функция с номером I. *FN*1* – косвенная адресация – функция, номер которой определен содержимым I-го параметра транзакта, поступающего в блок в поле операндов которого используется *FN*1* .

П р и м е р :

*SEIZE FN*1.*

При поступлении транзакта в этот блок необходимо решить вопрос: какое занять устройство. Симулятор дает ответ на этот вопрос, обратившись к I-му параметру транзакта, определив по нему номер функции и получив значение этой функции. Это значение и будет определять номер занимаемого устройства.

Если в тексте программы встречается запись " $*i$ ", то она эквивалентна " P_i " – параметр с номером i .

Косвенная адресация не допускается ни в одном из полей блока *GENERATE*, кроме поля A.

I9. ВРЕМЕННАЯ МОДЕЛЬ СИСТЕМЫ ИНФОРМАЦИОННОГО ОБМЕНА С ДИСКОВОЙ ПАМЯТЬЮ

Рассмотрим систему, в которую поступают запросы на информационный обмен с памятью на диске. Каждый запрос определяет цилиндр, с которого надо считывать или записывать информацию и объем передаваемых данных. Составим описание этой системы на *GPSS* , приняв следующие исходные данные:

среднее число запросов на информационный обмен, поступающих в систему в единицу времени - I/20;

число цилиндров - 203;

средняя длина запроса (объем, записываемой или считываемой информации) - 2000 байтов;

число дорожек в цилиндре - 2;

число байтов в дорожке - 6I44;

период оборота диска - 40 мс;

скорость передачи информации - I дорожка за 40 мс;

время поиска информации на диске при перемещении головки через " n " цилиндров:

$6 + 2\,n$ мс $\quad 0 \leqslant n \leqslant 8$

$16 + 3\,n/4$ мс $\quad 9 \leqslant n \leqslant 24$

$26 + n/3$ мс $\quad n \geqslant 25$

поток запросов - простейший;

объем информации, участвующей в обмене по запросу, находится в пределах I000-3000 байт. (Эти данные взяты для диска ЭВМ *PDP*-II. См. Э.Таненбаум. Многоуровневая организация ЭВМ.-М:Мир,I979,с.7I).

Опишем вычислительные объекты, необходимые для анализа этой системы.

Функция, описывающая время поиска информации на диске:

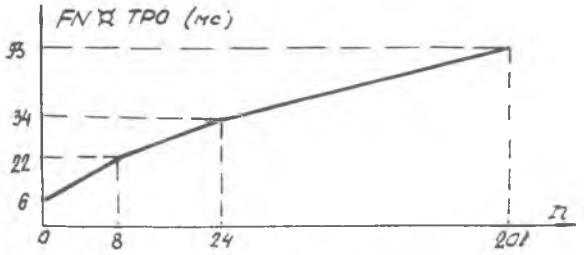
```
TPO FUNCTION V1,C4

Ø,6/8,22/24, 34/2Ø2,93
```

В качестве аргумента этой функции используется переменная

```
1 VARIABLE   (P1-X1)@2Ø3
```

В I-м параметре транзакта PI мы будем запоминать номер цилиндра, участвующего в информационном обмене по запросу, а в ячейке *SAVEVALUE* с номером I (ее СЧА *X1*) мы будем запоминать положение головки на диске.

Таким образом модуль разности $|P1-X1|$ имеет смысл количества цилиндров, через которые должна переместиться головка при поиске нужного цилиндра, т.е. $|P1-X1|$ имеет смысл n (рис. 7)



Р и с. 7. Время поиска информации на диске

График рис. 7 построен по приведенным выше соотношениям. (Операция @ в выражении для V1 используется для определения модуля разности).

Время передачи информации. Исходя из принятых условий, время передачи информации может быть описано переменной V2

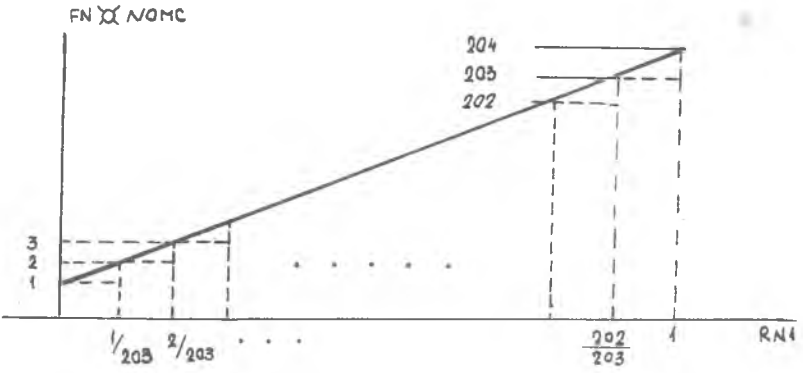
```
2 VARIABLE    P2/6144*40
```

при условии, что во 2-м параметре транзакта мы запишем объем информации, участвующей в обмене по данному запросу.

Описание потока запросов. Поток простейший, поэтому описывается функцией EXPON FUNCTION..., которая нам уже неоднократно встречалась (см.п. 8).

Функция, осуществляющая случайный выбор номера цилиндра: на диске 203 цилиндра. Каждый запрос адресуется определенному цилиндру. Будем считать, что запрос с одинаковой вероятностью может потребовать ресурсы любого из этих цилиндров. При этом при описании функции случайного выбора номера цилиндра следовало бы определить ее тип, как тип D . В этом случае функция имела бы вид, изображенный на рисунке пунктиром (рис. 8).

Для задания такой функции потребовалось бы использовать 203 пары аргумент-функция, что, конечно, неудобно. Однако, учитывая, что GPSS оперирует с целочисленными величинами и производит округление значения FN¤NOMC до целого числа, есть возможность задать эту функцию типом С, аппроксимируя ступенчатую ломаную прямой (сплошная линия на рисунке). При этом для задания функции оказы-



Р и с. 8. Функция случайного выбора

вается достаточно двух пар:

NOMC FUNCTION RN1, C2

ф, 1/1,2ф4

Эта функция будет осуществлять равновероятный выбор любого из 203 цилиндров. Ее аргументом является системный числовой атрибут *RN1*. Подобное описание функции случайного выбора используется очень широко в силу своей компактности. Однако не следует забывать, что процессы случайного выбора могут быть описаны функцией типа С2 лишь при условии равновероятности выбираемых целочисленных значений.

Функция, осуществляющая случайный выбор объема обмениваемой информации определяется аналогично функции *NOMC*. По нашим условиям, средняя длина запроса в байтах 2000. Мы условились считать, что объем данных, участвующих в обмене, может находиться в пределах от 1000 до 3000, причем все значения этого диапазона будем считать равновероятными. По аналогии с функцией *FN☒NOMC* определим функцию *FN☒VINF*

VINF FUNCTION RN1, C2

ф, 1ффф/1, 3ф01

Теперь мы имеем возможность составить программу, описывающую сам механизм информационного обмена:

```
GENERATE      2ф, FN¤EXPON
ASSIGN        1 , FN¤NOMC
ASSIGN        2,  FN¤VINF
SEIZE         DISK
ADVANCE       FN¤TPO
ADVANCE       V2
RELEASE       DISK
SAVEVALUE     1, P1
TERMINATE     1
```

Как видно, она довольно проста и не требует дополнительных пояснений. В этом примере большую трудность представляет определение вычислительных объектов программы.

20. ПРИОРИТЕТЫ

Каждый транзакт может иметь в системе свой приоритет. номер приоритета меняется от 0 до 127. Чем больше номер, тем больше приоритет. Во всех ситуациях, когда конкурируют транзакты с разными приоритетами, например, при занятии устройств, памятей, каналов ввода-вывода и т.д., предпочтение отдается транзакту с более высоким приоритетом. Если в этой конкуренции участвуют транзакты с одинаковыми приоритетами, то предпочтение отдается ранее поступившему. Для назначения приоритета используется блок

PRIORITY,

в поле A которого указывается значение присваиваемого приоритета 0-127.

2I. СПИСКИ

Информационные структуры типа списка определяют главным образом внутреннюю организацию *GPSS* .

Всего в системе 5 типов списков. Некоторые из этих списков ведет симулятор (пользователь доступа к ним не имеет): это списки текущих событий, список будущих событий, списки прерываний, списки синхронизируемых транзактов. В списках текущих событий в любой момент времени находятся транзакты, у которых время очередной передвижки в программе модели меньше системного времени. Это либо активные транзакты, которые надо передвинуть в настоящий момент системного времени, либо транзакты, находящиеся в состоянии задержки по причине, например, занятости устройства, конкуренции и т.д.

Все транзакты, находящиеся в состоянии задержки по той или иной причине, образуют свои списки задержанных транзактов. В список будущих событий вносятся транзакты, время передвижки которых больше текущего момента системного времени (список аналогичен календарю системы *REATE* [5]). В этом списке, например, всегда находится транзакт, поступивший в блок *ADVANCE*. Списки прерываний содержат транзакты, обслуживание которых на определенных устройствах было прервано другими транзактами.

Наконец, особый тип списков – списки пользователя. Списки пользователя управляются пользователем и симулятором. Списков пользователя может быть в системе до 20 или 100 в зависимости от версии языка.

Со списками пользователя связаны следующие основные СЧА:

CA_i – среднее число транзактов в i -ом списке;

CH_i – текущее число транзактов в i – ом списке;

CM_i – максимальное число транзактов в i – ом списке:

Пользователь управляет списками с помощью двух блоков: *LINK* и *UNLINK*.

LINK – связан с постановкой – включением транзакта в список пользователя. В поле A этого блока указан номер списка. В поле B указывается дисциплина внесения транзакта в список.

В качестве дисциплины внесения могут использоваться 3 дисциплины:

1. Первый пришел, первый обслуживается – *FIFO* .
2. Последний пришел, первый обслуживается – *LIFO* .
3. Упорядочение списка по параметру P_i .

По дисциплине *FIFO* транзакт, поступающий в *LINK* , ставится в конец списка, по дисциплине *LIFO* - в начало.

При использовании дисциплины P_i - транзакт ставится в список в соответствии со значением его i -го параметра (список упорядочен по значению P_i).

Использование блока *LINK* позволяет пользователю на некоторое время отдельные транзакты переводить в пассивное состояние,т.е. выводить из системы.

Блок *UNLINK* - извлечение транзакта из списка. В поле А указывается номер списка. В поле В - номер или имя того блока, к которому направляется извлеченный из списка транзакт. В поле С - количество транзактов, извлекаемых из списка:

UNLINK 1, NEXT, 2

При попадании некоторого транзакта в *UNLINK* пассивный транзакт, находящийся в списке пользователя, извлекается оттуда и направляется к блоку, указанному в поле В, основной же транзакт, вызвавший это извлечение, продолжает свое обычное движение в программе модели.

Использование этих двух типов блоков позволяет программировать разного рода асинхронные процессы, происходящие в системе.

22. СТАТИСТИЧЕСКИЕ ТАБЛИЦЫ

Для сбора статистики в *GPSS* существует много возможностей. В частности, любой параметр, интересующий пользователя, может быть подвержен статистическому анализу с построением гистограммы.Для проведения такого анализа используется особый тип элементов языка - таблицы.

Карта описания таблицы - *TABLE* .

В поле метки этой карты указывается имя или номер таблицы.

В поле А - СЧА, который подвергается табулированию.

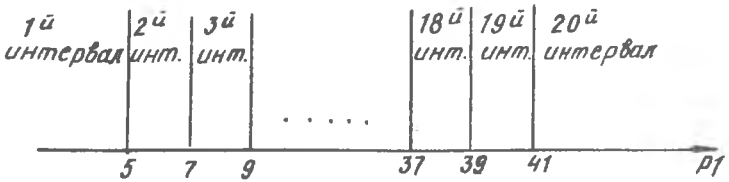
В поле В - верхняя граница I-го классового интервала гистограммы.

В поле С - ширина классового интервала.

В поле D - количество классовых интервалов.

Сам процесс табулирования происходит, когда транзакт поступает в блок табулирования - *TABULATE*.

1 TABLE P1, 5,2, 2ф



В поле А этого блока пишется ссылка на соответствующую таблицу (либо номер таблицы, либо имя). Каждый раз, когда транзакт поступает в блок табулирования, к соответствующему классовому интервалу таблицы будет приплюсована I. Выбор классового интервала осуществляется в зависимости от значения табулируемой величины на момент поступления транзакта в блок TABULATE .

Факт поступления транзакта в блок TABULATE служит для симулятора сигналом об очередном сборе статистики. Кроме построения гистограммы для каждой карты описания таблицы автоматически осуществляется оценка среднего и среднеквадратического значения табулируемого СЧА.

23. МОДЕЛЬ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ С НЕСКОЛЬКИМИ АБОНЕНТСКИМИ ПУНКТАМИ

Рассмотрим систему, в которой ЭВМ обслуживает 6 абонентских пунктов (АП). Программа управления каналом связи опрашивает АП в соответствии со списком опроса. Если у опрашиваемого АП имеется сообщение для передачи, оно посылается в ЭВМ. После завершения передачи данных канал освобождается, и как только выходное сообщение от ЭВМ готово, оно сразу занимает канал связи для передачи, т.е. выходное сообщение (ответ ЭВМ) имеет приоритет перед входным (запрос пользователя). (С незначительными упрощениями этот пример заимствован из монографии Дж.Мартина Системный анализ передачи данных . Кн.2, М.:Мир.).

Исходные данные:

I. Опрос циклический.

2. Время, затрачиваемое на опрос одного АП-100 мс.

3. Интервалы времени между опросами - 10 мс.

4. Передача информации ведется со скоростью - 300 симв/с.

5. Время обработки сообщения в ЭВМ - 500 мс.

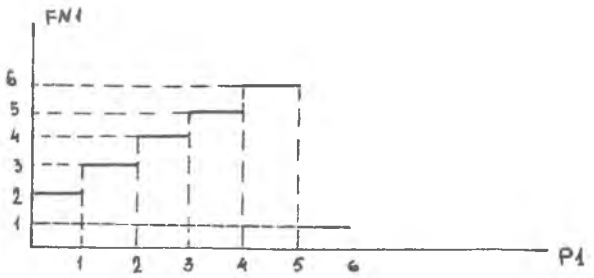
П р и м е ч а н и е. Механизм розыгрыша момента возникновения сообщения на АП и длины сообщения поясняются в программе.

Прежде всего определим вычислительные объекты языка: функции и переменные.

FN1 - функция, которая будет имитировать список опроса. Для описания этого опроса будет использоваться функция типа *D* , ее аргументом будет являться параметр P1 при условии, что в него будет записываться номер опрашиваемого АП (рис. 9).

```
1 FUNCTION P1,D6
1,2/2,3/3,4/4,5/5,6/6,1
```

FN2 - функция, определяющая интервалы времени между возникновениями сообщения на АП.



Р и с. 9. Список опроса

Поскольку в задании конкретно не указаны закономерности возникновения сообщений на АП, предположим, что интервалы времени между возникающими сообщениями распределены экспоненциально (поток сообщений простейший [4]).

В этом случае можно использовать функцию, описанную в п. 8 -

```
2  FUNCTION RN1, C6
0,0/ ..................
```

с которой мы уже неоднократно встречались.

FN3 - функция, по которой разыгрывается номер АП, имеющего сообщение для передачи в ЭВМ.

Поскольку закономерности возникновения сообщений на АП для их последующей передачи в ЭВМ также не оговорены в исходных данных, предположим, что сообщение для передачи может с одинаковой вероятностью возникнуть на любом из 6 терминалов (АП).

При этом функция *FN3* должна осуществить равновероятный выбор одного из 6 АП. Значит ее аргументом будет являться случайное число, а тип такой функции уже обсуждался выше при имитации системы информационного обмена с дисковой памятью (см.п.I9).

```
3 FUNCTION RN2, C2
 Ø,1/1,7
```

FN4 - функция, определяющая число символов во входном сообщении, возникающем на АП.

Допустим, что сообщение может содержать от 6 до 60 сотен символов. Функция, осуществляющая случайный выбор длины сообщения, определяется аналогично *FN3* (при условии, что этот выбор равновероятный):

```
4 FUNCTION RN3, C2
 0,6/1,6I
```

V1 - переменная, определяющая время занятия канала связи для передачи сообщения в ЭВМ.

Эта переменная полностью определяется условиями задачи как частное от деления длины сообщения на скорость передачи информации по каналу:

```
1 VARIABLE FN4 * 1ØØØ/3.
```

Появление коэффициента I000 в этой карте обусловлено тем,что в качестве единицы времени в модели целесообразно выбрать I мс.

Переменную, определяющую время занятия канала передачей ответного сообщения от ЭВМ к АП, и функцию, определяющую длину ответного сообщения (*V2* и *FN5*), определим аналогично *V1* и *FN4* .

Описание таблицы, определяющей гистограмму времени ответа ЭВМ на сообщение - запрос пользователя, поступающий с АП:

```
I TABLE M1,Ø, 2ØØ, 4Ø
```

M1 - СЧА транзакта, определяющий время его жизни (см.п.I3).

Программа имитации рассматриваемой системы состоит из 2-х взаимосвязанных фрагментов.

А. Модель передачи информации от АП в ЭВМ и обратно

```
GENERATE  1,FN2      Генерируется транзакт, имитирующий возникновение сообщения на АП
ASSIGN    1, FN3     В I-й параметр транзакта записывается номер АП, на котором появилось сообщение
LINK      P1, FIFO   Транзакт помещается в список пользователя, номер которого соответствует номеру АП, на котором появилось сообщение
```

Этот фрагмент программы связан с имитацией сообщений, возникающих на АП, и постановкой их в списки пользователя. Система заводит по одному списку на каждый АП.

Ниже следует фрагмент программы, описывающий процесс передачи входного сообщения, расчеты ЭВМ и передачу выходного сообщения.

```
XMIT  SEIZE     1     Занять канал связи для передачи входного сообщения
      ADVANCE   V1    Задержка на время передачи входного сообщения
      RELEASE   1     Освобождение канала
      ADVANCE   5ФФ   Задержка на время обработки сообщения в ЭВМ
      PRIORITY  1     Выходное сообщение имеет более высокий приоритет, чем входное
      SEIZE     1     Занять канал связи для передачи выходного сообщения
      ADVANCE   V2    Имитируем время передачи выходного сообщения
      RELEASE   1     Освобождаем канал
      TABULATE  1     Табулируем время ответа ЭВМ
      TERMINATE 1     Уничтожаем транзакт
```

В. Моделирование процесса опроса АП

```
        GENERATE    ,,,1          Генерируется один транзакт, имитирующий сигнал опроса
        ASSIGN      1,K1          Параметру транзакта PI присваивается номер I-го опрашиваемого АП
POLL    ASSIGN      1,FN1         Определяется номер опрашиваемого АП в соответствии со списком опроса. Он записывается в PI
        SEIZE       1             Занять канал связи на время опроса
        ADVANCE     1ФФ           Комментарии к этим блокам даны ниже
        TEST NE     CH*1,KФ,REL
        UNLINK      P1,XMIT
REL     RELEASE     1             Освобождение канала связи, опрос окончен
        ADVANCE     1Ф            Задержка на время между опросами
        TRANSFER    ,POLL         Повторение цикла опроса
```

Блок *TEST* проверяет, пуст ли список, соответствующий опрашиваемому АП. Если список пуст, управление передается блоку с меткой *REL* и продолжается процесс опроса. Если список не пуст, блок *UNLINK* выводит из этого списка сообщение для передачи в ЭВМ и передает управление блоку *XMIT* . Транзакт, вызвавший извлечение из списка сообщения, по-прежнему направляется к *REL* и продолжает цикл опроса. Таким образом, в модели опроса один единственный транзакт, порожденный блоком *GENERATE* и имитирующий сигнал опроса, циркулирует по "замкнутому кругу", определяемому списком опроса, занимая на своем пути канал связи (он имитируется устройством I) и задерживаясь на определенные условиями опроса интервалы времени. *TRANSFER* всегда посылает этот транзакт к *POLL* , опрашивается новый АП в соответствии со списком опроса (*FN1*) и т.д. Конструкция блока *TEST* , встречающаяся на пути транзакта, позволяет ему либо обойти блок *UNLINK* , либо пройти через него. Первый случай имеет место, когда список пользователя, соответствующий опрашиваемому АП, пуст (*CH*1* = 0); второй, - когда он не пуст (*CH*1* ≠ 0), т.е. когда у опрашиваемого АП есть сообщение для передачи в ЭВМ. В этом случае транзакт, имитирующий сигнал опроса, поступает в блок *UNLINK* и вызывает извлечение из списка пользователя транзакта, имитирующего сообщение, возникшее на АП. Последний направляется в *XMIT* (где имитируется передача сообщения по

каналу АП-ЭВМ и обратно), а первый, т.е. сигнал опроса направляется к *REL* , продолжая процесс опроса.

Налицо два процесса - "опрос" и "передача", причем каждый из них воплощается в самостоятельный фрагмент программы и имитируется движением "своего собственного" транзакта. Взаимодействие же этих процессов осуществляется через списки пользователя (процесс "передача" заполняет списки, а "опрос" - освобождает). Подобные соображения позволяют иногда утверждать, что в основе *GPSS* лежит схема процессов [5], а схема транзактов - лишь удобная для пользователя формализация понятия процесса.

24. ЛОГИЧЕСКИЕ ПЕРЕКЛЮЧАТЕЛИ

Одним из основных элементов языка, имитирующим состояние оборудования, являются логические переключатели. Логический переключатель может находиться в двух положениях "включен" или "выключен". Все переключатели перед началом выполнения программы устанавливаются в положение "выключен". Для управления логическим переключателем используется блок *LOGIC* . При поступлении транзакта в этот блок состояние логического переключателя, номер которого указан в поле А поля операндов, меняется в соответствии с мнемоникой, записанной в поле операции карты *LOGIC* (в I4-й колонке).

LOGIC	*R*	*1*	- логический переключатель с номером I устанавливается в состояние "выключен"
LOGIC	*S*	*10*	- логический переключатель с номером IO устанавливается в состояние "включен"
LOGIC	*I*	*4*	- логический переключатель с номером 4 меняет свое состояние (инвертируется)

Состояние логического переключателя может быть проверено транзактом в любой части модели. В зависимости от этого состояния транзакт может изменить путь своего дальнейшего продвижения в программе модели, изменить содержание своего параметра и т.п. Логические переключатели могут использоваться в модели для самых различных целей, этот тип элементов языка не имеет СЧА. Далее описывается фрагмент модели, использующий логические переключатели.

25. МОДЕЛЬ ИНФОРМАЦИОННОГО ОБМЕНА В ДВУХУРОВНЕВОЙ ПАМЯТИ

Рассмотрим систему, в которой используется набор из IO программ пользователя. В определенные моменты времени система может потребовать выполнения любой из этих IO программ. Однако в области ОЗУ, отведенной под программы пользователя, все IO программ не могут быть размещены одновременно. Первоначально все программы записаны на магнитной ленте (МЛ), и при генерации системы часть их загружена в ОЗУ. Если в процессе работы системы потребуется программа, отсутствующая в ОЗУ, то она будет загружена с МЛ в область, занимаемую программой, количество обращений к которой в период всего функционирования системы оказывается минимальным. Условимся, что единицей информационного обмена МЛ ⟷ ОЗУ является программа, при этом загружаемая программа не может занять в ОЗУ бо́льшего объема памяти, чем затираемая. Допустим, что в начальный момент времени программы пользователя размещены следующим образом:

PR1	PR2	PR3	PR4	PR5	PR6 ÷ PR1ϕ
ОЗУ	ОЗУ	ОЗУ	ОЗУ	ОЗУ	МЛ

Поставим в соответствие каждой программе логически переключатель, который будет находиться в состоянии "выключен", если соответствующая программа отсутствует в ОЗУ и "включен" – в противоположном случае.

Начальная загрузка программ в ОЗУ может быть описана, например, следующим образом:

```
GENERATE   ,,, 1
LOGIC  S     1
LOGIC  S     2
LOGIC  S     3
LOGIC  S     4
LOGIC  S     5
TERMINATE 1
```

(Эти же действия выполняет одна из карт инициализации *INITIAL*:

INITIAL LS1-LS5 см. п. 30)

В процессе имитации в какой-то части модели должен быть сгенерирован транзакт, имитирующий обращение к той или иной программе пользователя. Поскольку механизм возникновения такого обращения нами никак не описан, не будем приводить описания программы, связанной с генерацией такого транзакта-обращения. Но предположим, что в первом параметре этого транзакта записан номер программы, к которой происходит обращение в исследуемой системе. Подобный транзакт должен быть направлен в программе модели к фрагменту, описывающему загрузку МЛ ⟶ ОЗУ. Этот фрагмент может быть описан на *GPSS*, например, следующим образом:

Транзакт-обращение (в PI-номер требуемой программы)	: *SAVEVALUE*	**1+,1*	Счетчик обращений к программе: *X*1:=X*1+1*
	GATE LS	**1,BGN1*	Проверка состояния ключа: если ВКЛ, перейти к след. блоку, если ВЫКЛ, перейти к *BGN1*
	ADVANCE	*FN1*	Задержка на время отработки программы
	TRANSFER , *OUT*		Выход из алгоритма
BGN1	Найти среди ячеек *X1÷X10* , для которых ключи *L1÷L10* включены, ячейку с минимальным содержимым и записать ее номер в параметр P10		
BGN2	*LOGIC R*	**10*	Перевести ключ, соответствующий затираемой программе, в состояние ВЫКЛ
	LOGIC S	**1*	Перевести ключ, соответствующий загружаемой программе, в состояние ВКЛ
	ADVANCE	*FN2*	Задержка на время загрузки программы в ОЗУ
	ADVANCE	*FN1*	Задержка на время отработки программы

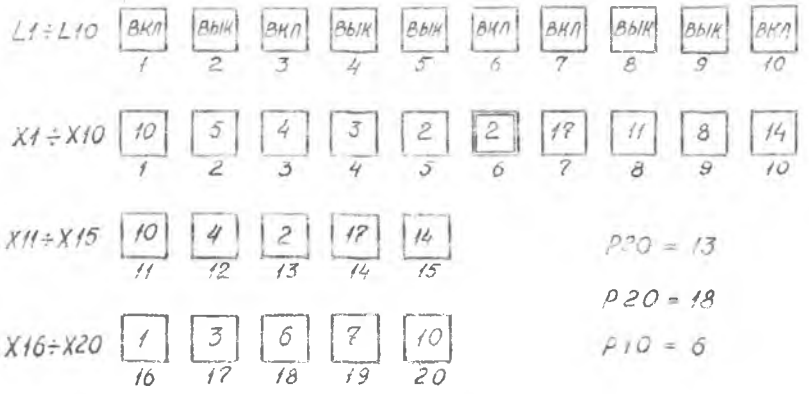
GATE разветвляет дальнейший путь транзакта в зависимости от состояния переключателя, номер которого соответствует номеру программы, к которой происходит обращение, и записан в параметре I

транзакта, имитирующего обращение. Если переключатель выключен (*LR*), транзакт поступает на фрагмент *BGN1*, если включен – на блок *ADVANCE FN1*. Функция *FN1* описывает время отработки программы, а *FN2* – время загрузки программы с МЛ в ОЗУ (подробно *FN2*, как и *FN1*, мы здесь не описываем). Далее блок *LOGIC S* включает соответствующий переключатель (свидетельствуя, что требуемая программа загружена в ОЗУ), а *LOGIC R* (с меткой *BGN2*) имитирует затирание соответствующей программы в ОЗУ. Ниже подробно расписывается фрагмент *BGN1*, определяющий номер затираемой программы в ОЗУ.

BGN1	*ASSIGN*	*2,ϕ*	*P2:=ϕ*
	ASSIGN	*3,11*	*P3:=11*
	ASSIGN	*4,16*	*P4:=16*
	ASSIGN	*5,1ϕ*	*P5:=1ϕ*
BGNϕ1	*ASSIGN*	*2+,1*	*P2:=P2+1*
	GATE LS	**2,BGNϕ2*	
	SAVEVALUE	**3,X*2*	В ячейку с номером P3 переписывается содержимое ячейки с номером P2
	ASSIGN	*3+,1*	*P3:=P3+1*
	SAVEVALUE	**4,*2*	В ячейку с номером P4 записывается P2
	ASSIGN	*4+,1*	*P4:=P4+1*
BGNϕ2	*LOOP*	*5,BGNϕ1*	Повторить фрагмент *BGNϕ1÷BGNϕ2* (P5-1) раз (т.е. 9 раз)
	SELECTMIN	*2ϕ,11,15,,X*	Среди ячеек *X*11 ÷ *X*15 находится ячейка с минимальным содержимым, и ее номер записывается в P2*ϕ*
	ASSIGN	*2ϕ+,5*	*P2ϕ:=P2ϕ+5*
	ASSIGN	*1ϕ,X*2ϕ*	*P1ϕ:=X*2ϕ.*

Конструкция *SELECTMIN* идентифицирует среди ячеек *X11÷X15* ячейку с минимальным содержимым, блок *LOOP*, организующий цикл, подробнее описывается в п. 28.

На рис. 10 для иллюстрации работы фрагмента *BGN1* приведено одно из возможных состояний поля атрибутов, имитирующих обмен *МЛ←→ ОЗУ*. Приведенные на рисунке значения параметров P20, P10 соответствуют отработке трех последних блоков фрагмента *BGNϕ2*.



Р и с. 10. Массивы *L1÷L10* , *X1÷X20* , имитирующие состояние информационного обмена МЛ ——ОЗУ

Выше при описании блока *GATE* (см. п.14) мы оперировали только с мнемокодами состояний устройств и памятей. Этот пример иллюстрирует возможности использования в блоке *GATE* мнемокодов состояний ключей

LS – переключатель включен;

LR – переключатель выключен.

Механизм работы блока *GATE* при этом остается прежним.

26. ПРЕРЫВАНИЯ

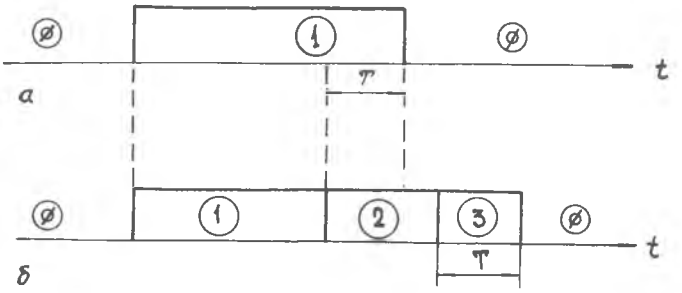
При имитации вычислительных процессов реального времени, систем с разделением времени, мультипрограммных систем и т.д. нередко приходится воспроизводить в модели механизм работы системы прерываний. В *GPSS* существуют специальные конструкции для реализации механизмов прерываний. Процесс прерываний обслуживается двумя блоками: *PREEMPT* и *RETURN* , в поле А которых указывается номер (имя) устройства, на котором прерывается обслуживание. Например, рассмотрим конструкцию

```
PREEMPT   CPU
ADVANCE   1Ø
RETURN    CPU
```

В этом фрагменте транзакт, поступающий на блок *PREEMPT* , занимает устройство *CPU* (если оно было свободно в момент поступления транзакта), удерживает *CPU* в течение 10 единиц времени и освобождает устройство, т.е. в случае, если устройство было свободно, приведенный фрагмент аналогичен по своим действиям следующему:

```
SEIZE  CPU
ADVANCE 10
RELEASE CPU
```

Если в момент поступления транзакта на блок *PREEMPT* устройство *CPU* оказалось занятым (обслуживанием другого транзакта), происходит прерывание обслуживания. По истечении 10 единиц системного времени это прерывание снимается конструкцией *RETURN* , а на устройстве *CPU* возобновляется прерванное обслуживание. Ситуация иллюстрируется диаграммой рис. 11.



Р и с. 11. Прерывание обслуживания:
а) обслуживание без прерывания:
ø - CPU свободно,
1 - CPU занято;
б) обслуживание с прерыванием:
2 - обслуживание прервано,
3 - прерывание снято, обслуживание продолжено

Таким образом, использование блоков *PREEMPT* и *RETURN* позволяет прерывать обслуживание транзакта любым другим транзактом (сам свое собственное обслуживание транзакт прервать, естественно,

не может). Работа механизма прерываний иллюстрируется примером п. 29).

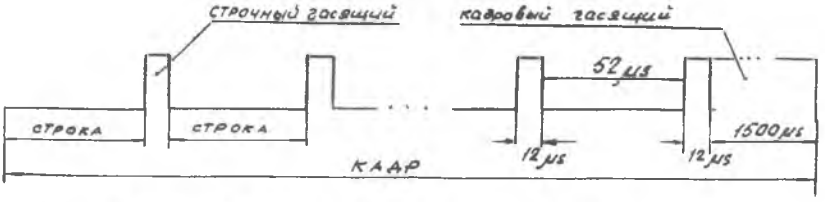
27. КОПИИ ТРАНЗАКТОВ

Каждый транзакт при прохождении через модель системы может породить своего двойника – копию (или несколько копий). Осуществляется это при попадании транзакта в блок *SPLIT* . В поле A этого блока указывается количество копий, которые необходимо создать, а в поле B – метка (номер) блока, к которому они направляются. Основной транзакт при этом направляется к следующему за *SPLIT* блоку. Копии в системе обладают "всеми правами" транзактов: они могут занимать устройства, памяти, задерживаться, уничтожаться и т.д. Копирование позволяет разветвлять процесс "путешествия транзакта" на два и более подобных процесса, что весьма удобно при имитации параллельных механизмов работы сложных систем.

28. ЦИКЛЫ

Для организации циклов в программе на *GPSS* служит блок *LOOP*. В поле A этого блока указывается номер параметра, который выполняет функцию счетчика циклов: каждый раз при поступлении транзакта в блок *LOOP* из этого параметра вычитается I и разность вновь записывается в данный параметр. Как только эта разность станет равна 0, транзакт направится в блок, следующий за *LOOP* . При разности большей нуля транзакт направляется к блоку, имя (или номер) которого указаны в поле B блока *LOOP* .

Например, в следующем фрагменте с использованием блока *LOOP* описывается временной алгоритм **синхрогенератора** телевизионной развертки на интервале I0 кадров. Временная диаграмма развертки изображена на рис. I2 (в кадре 287 строк).



Р и с. I2. Временная диаграмма развертки

```
        GENERATE    ,,,1
        ASSIGN      1,1ф
KDR     ASSIGN      2,287
STR     ADVANCE     52
        ADVANCE     12
        LOOP        2,STR
        ADVANCE     15фф
        LOOP        1,KDR
        TERMINATE   1
```

29. СИСТЕМНОЕ ВРЕМЯ

Выше уже отмечалось, что системный числовой атрибут C_1, определяющий текущее значение системного времени, доступен пользователю в любой точке его программы. Кроме того, каждый транзакт при генерации всегда снабжается отметкой времени ("датой рождения"):

< дата рождения > := $C1$.

Время пребывания транзакта в модели (атрибут $M1$) отсчитывается от момента его рождения:

$M1 := C1 -$ < дата рождения >

"Дату рождения" транзакта, зафиксированную блоком *GENERATE*, можно изменить в любом месте программы, используя для этого блок *MARK* (с пустым полем *A*). Транзакт, попадающий в блок *MARK*, меняет "дату рождения" ("молодеет") на величину $M1$. В поле *A* блока *MARK* можно указать номер параметра транзакта, при этом транзакт сохранит свою "дату рождения", но в указанном параметре запишется текущее значение системного времени - $C1$:

MARK 1ф эквивалентно $P10 := C1$.

Создав таким образом отметку времени, в дальнейшем можно ее использовать для измерения времени прохождения транзактом отдельных участков программы, обращаясь к его СЧА MP_n (в этом примере $MP10$).

Например

```
        . . . . . .
        MARK
        . . . . . .
        TESTE  M1,K1ф,BGN
```

В этом фрагменте *M1* - время, отсчитываемое от момента попадания транзакта в блок *MARK* .
Еще пример:

```
          . . . . . . . . . .
          MARK . . . . . .
          MARK      1ф.
          . . . . . . . . .
          TEST E    C1 , K5ф , BG N1
SR1       TEST GE   M1 , MP1ф , BG N2
```

Первый блок *TEST* здесь проверяет условие, связанное с текущим значением системного времени (*C1* =50?). Второй блок *TEST* сравнивает *M1* (время от попадания транзакта в блок *MARK* до попадания его в блок *SR1*) с *MP1ф* (временем от попадания транзакта в блок *MARK* *1ф* до попадания его в блок *SR1*).

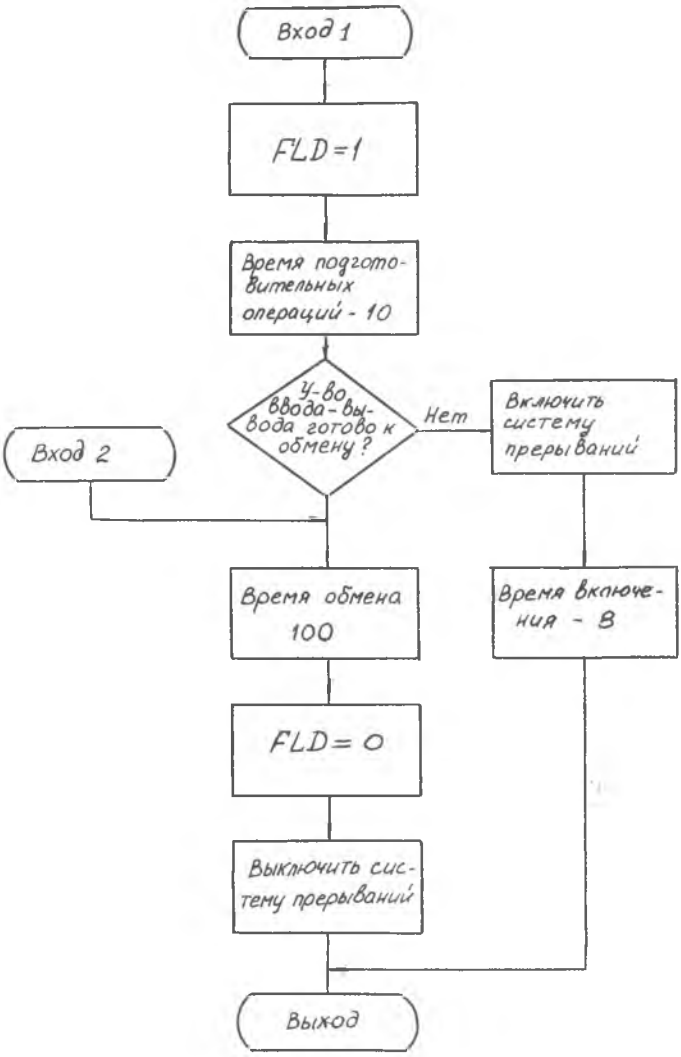
Использование СЧА *M1, MPh(MP*h), C1* позволяет определять сложные логические условия, основанные на временных соотношениях в моделируемой системе.

30. МОДЕЛЬ ИНФОРМАЦИОННОГО ОБМЕНА, ИСПОЛЬЗУЮЩЕГО ДРАЙВЕР УСТРОЙСТВА ВВОДА-ВЫВОДА

Программный обмен информацией с внешними устройствами в управляющих системах происходит с использованием специальных программ-драйверов ввода-вывода. Ниже рассматривается упрощенная модель такого обмена. На рис. I3 приведена структурная схема программы драйвера. Здесь *FLD* - флаг занятости драйвера (I- занят, 0 - свободен). Вход I - программный, вход 2 - вход по прерыванию. Процессор управляющей машины, выполняя расчеты по программам, инициирует запросы к драйверу на ввод-вывод информации, причем "войти" в программу драйвера можно только в случае, если он свободен (*FLD=0*). В противном случае, инициировав запрос, процессор циклически повторяет его, дожидаясь освобождения драйвера.

Программа драйвера проверяет состояние устройства ввода-вывода. Если оно оказывается не готовым к обмену, включается система прерываний, и управление из драйвера передается в вызывающую программу для продолжения расчетов. При включенной системе прерываний устройство ввода-вывода информации в момент готовности прервет выполнение расчетов процессором, передав управление на вход 2 драйвера.

Ниже приведено описание на *GPSS* структуры драйвера, изображенной на рис. I3.



Р и с. I3. Упрощенная структура программы драйвера

```
        LOGIC       S     1
        ADVANCE           1Ф
        GATE        LS    2,SPR
ENT     ADVANCE           1ФФ
        LOGIC       R     1
        LOGIC       R     3
        TRANSFER          ,END1
SPR     LOGIC       S     3
        ADVANCE           8
END1    TRANSFER          ,BGN
```

Ключ I имитирует флаг занятости драйвера.

Ключ 2 - готовность устройства ввода-вывода.

Ключ 3 - включение (выключение) системы прерываний.

BGN - метка фрагмента программы, имитирующего обращения к драйверу. Если условиться, что время между моментом выхода из драйвера и следующим обращением к нему распределено равномерно в диапазоне 50-80 единиц, имитация потока запросов к драйверу может быть осуществлена следующим описанием:

```
        GENERATE    ,,,1
        SEIZE       CPU
BGN     ADVANCE     65,15
        GATE LR     1
```

В программе этот фрагмент должен размещаться перед предыдущим.

Готовность устройства ввода-вывода к обмену может определяться целым рядом дополнительных факторов (в первую очередь все зависит от того, какое это устройство). Учитывая иллюстративный характер примера, мы не будем конкретизировать тип подобного устройства, описав его функционирование вероятностной моделью. Условимся, что от момента завершения обмена (т.е. момента освобождения драйвера) до момента готовности устройства ввода-вывода к следующему обмену проходит некоторое случайное время, равномерно распределенное в диапазоне [60, 90].

При этом условии управлять процессом готовности устройства ввода-вывода можно, вводя в описание драйвера дополнительную конструкцию, использующую блок *SPLIT*:

```
*       Имитация процесса программного счета
        GENERATE    ,,,1
        ASSIGN      1,K1
        SEIZE       CPU                 Имитация запросов к драй-
BGN     ADVANCE     65,15               веру
        GATE  LR    1
*-------------------------------------------------------------------
        LOGIC  S    1                   Модель драйвера:
        ADVANCE     1Ф
        GATE  LS    2,SPR
        LOGIC  R    2                   Перевод устройства ввода-
ENT     ADVANCE     1ФФ                 вывода в состояние "не
        LOGIC  R    1                   готово"
        LOGIC  R    3
        TEST  NE    P1,KФ,BGN1
        SPLIT       1,UBB               Создание копии для инициа-
        TRANSFER    ,END1               лизации готовности устрой-
SPR     LOGIC  S    3                   ства ввода-вывода
        ADVANCE     8
END1    TRANSFER    ,BGN
*-------------------------------------------------------------------
UBB     ADVANCE     75,15               Инициализация готовности
        LOGIC  S    2                   устройства ввода-вывода

        TERMINATE   1                   Уничтожение копии
```

Приведенное описание не учитывает пока возможности прерывания программы. Описать механизм прерываний можно следующим фрагментом:

```
*       Имитация системы прерываний
        GENERATE    ,,,1
        ASSIGN      1,KФ
BGN2    GATE LS     3
        GATE LS     2
        PREEMPT     CPU
        TRANSFER    ,ENT
BGN1    RETURN      CPU
        TRANSFER    ,BGN2
```

В целом описание модели состоит из двух рассмотренных секций: в каждой из них генерируется по одному транзакту и связь между секциями устанавливается только через элементы, имитирующие оборудование: логические ключи № 1, 2, 3 и устройство *CPU*. Первая секция связана с занятием устройства *CPU* обычным счетом по программе,при этом не делается различий,по какой именно программе(драйвер или программа пользователя, инициирующая запросы к нему, - процессор (CPU) всегда занят), поэтому в первой секции нет блока *RELEASE*.

Вторая секция связана с прерыванием программного счета, имитируемого в первой секции (прерыванием обслуживания на устройстве CPU). При формировании задания на моделирование безразлично, в какой последовательности эти секции будут размещены в общей программе. Транзакт "путешествующий" по программе первой секции, содержит в *P1* единицу,по второй - ноль, блок *TEST* синхронизирует их взаимодействие. Фактически в основном вся логика работы такой модели определяется управлением тремя ключами 1, 2, 3, их включением и выключением, поскольку именно их состояние определяет временные задержки на пути прохождения транзакта. При необходимости легко собрать статистику об этих задержках, размещая в соответствующих местах программы модели карты *QUEUE* и *DEPART* или используя блок *MARK*. (Перед началом имитации устройство ввода-вывода должно находиться в состоянии"готово", т.е. ключ № 2 должен быть включен).

При имитации на *GPSS* систем программного обеспечения широко используется режим "подпрограмма" *(SBR)* блока *TRANSFER*. Для этого режима в поле *A* блока *TRANSFER* пишется мнемокод *SBR*, в поле *B* указывается блок, к которому пытается перейти транзакт, значение поля *C* интерпретируется как номер параметра сообщения,- в этом параметре системой *GPSS* автоматически записывается номер блока *TRANSFER*. Например, для конструкции *TRANSFER SBR, NEXT,10* транзакт, попавший в *TRANSFER* будет направлен к блоку с меткой *NEXT*, а в *P10* запишется номер блока *TRANSFER*.

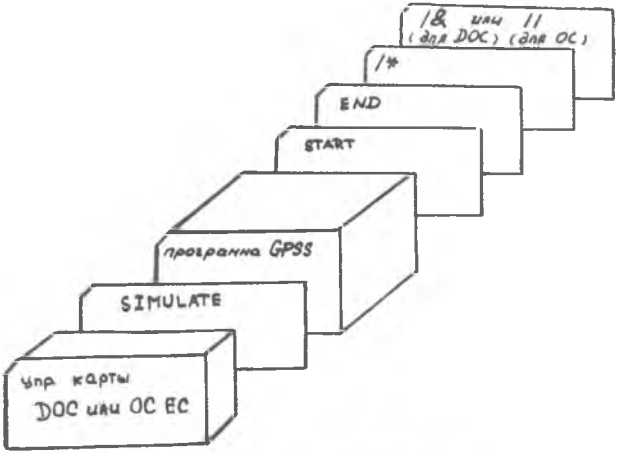
Для организации возврата из подпрограммы используется режим "параметр" *(P)* блока *TRANSFER*. При этом значение аргумента поля *B* интерпретируется *GPSS* как номер параметра вошедшего в *TRANSFER* транзакта. Для определения следующего для этого транзакта блока значение этого параметра складывается с содержимым поля *C*, получаемая сумма определит номер блока, к которому будет пытаться перейти транзакт, "возвращающийся из подпрограммы":

```
*                *    основная программа
                      . . . . . .
                      TRANSFER SBR, NEXT, 1Ф
                      ADVANCE
                      . . . . . .
*                *    подпрограмма
NEXT                  . . . . . .
                      . . . . . .
                      TRANSFER Ф,1Ф,1 конец подпрограммы
```

В этом примере транзакт, направленный *TRANSFER SBR* в подпрограмму *NEXT* , после отработки модели подпрограммы будет возвращен конструкцией *TRANSFER Ф* в блок *ADVANCE* . Рассмотренный пример модели драйвера легко модифицируется с использованием этих конструкций *GPSS* .

3I. УПРАВЛЯЮЩИЕ КАРТЫ И ИНИЦИАЛИЗАЦИЯ МОДЕЛИ

На рис. I4 приведена структура набора перфокарт, содержащего задание на моделирование системы с использованием *GPSS* .



Р и с. I4. Задание на моделирование

Наборы управляющих карт ДОС и ОС ЕС приведены в прил.. Без карты *SIMULATE* моделирование не производится, осуществляется лишь трансляция программы *GPSS* . Карта *START* инициирует начало моделирования, в поле А этой карты обязательно должно быть указано целое число (счетчик прогонов), определяющее длительность имитации. Карта *START* взаимодействует в модели с блоками *TERMINATE* у которых не пусты поля А. При каждом поступлении транзакта (или копии) в такой блок из счетчика прогонов карты *START* вычитается содержимое поля А карты *TERMINATE* . В программе может быть несколько блоков *TERMINATE* (как с пустыми, так и с непустыми полями А). Если во всех этих блоках поле А пусто, процесс моделирования будет продолжаться неограниченно долго, до снятия программы оператором.

Для организации работы модели на определенном интервале системного времени (например, от 0 до 1000 единиц времени) можно оформить отдельный фрагмент программы, завершающий моделирование по истечении заданного времени:

```
GENERATE    1000,1   }
TERMINATE   1        }   имитатор интервала времени моделирования
. . . . . .
START       1
```

Подобный имитатор интервала времени взаимодействует только с картой *START* и может быть никак не связан с содержательной стороной остальных фрагментов модели. Организовать завершение моделирования с использованием блоков *GENERATE*, *TERMINATE* и карты *START* можно самыми разнообразными способами, хотя формально окончание моделирования всегда определяется только одним условием - обнулением счетчика прогонов.

END - последняя карта задания;

*/** и */&* - обычные управляющие карты операционной системы (*/** -„конец задания" для ДОС ЕС заменяется в ОС ЕС картой *//*), *SIMULATE*, *START*. *END* пишутся в поле операций бланка, начиная с 8 колонки, без пробелов.

В начале процесса имитации пользователю *GPSS* может потребоваться присвоить некоторые начальные значения ячейкам *SAVEVALUE* и включить определенные ключи. Делается это путем введения в задание карты *INITIAL* . Например, *INITIAL X10 , K11* - присвоить десятой ячейке *SAVEVALUE* значение 11.

Или *INITIAL X5-X7, K20/X10,K1* - присвоить ячейкам 5-7 значение 20, ячейке 10 - значение 1.

Включение ключей: *INITIAL LS1-LS7* - включить ключи 1-7.

Напомним, что первоначально перед инициализацией модели все ключи, используемые в программе, считаются выключенными, содержимое используемых ячеек *SAVEVALUE* - нулевое.

Вычислительные элементы языка (карты описания и задания функций, карты описания переменных, карты инициализации ячеек *SAVEVALUE*) обычно размещаются перед описанием алгоритма функционирования системы (перед блоком *GENERATE*). Все остальные элементы языка (устройства, памяти, списки и т.д.) не требуют отдельного описания и инициализации, - такое описание заводится системой *GPSS* при просмотре программы пользователя. Перед началом имитации все устройства не заняты, памяти и списки - пусты.

При необходимости осуществления нескольких циклов моделирования с различными начальными значениями параметров пользователю *GPSS* следует ввести в программу карты *RESET* , разместив их, например, таким образом:

```
INITIAL     X1, K10
. . . . . .
START       100
RESET
INITIAL     X1, K50
START       300
RESET
INITIAL     X1, K100
START       300
END
```

Действие этой карты сводится к обнулению системного времени и обращению всех накопленных статистик в 0, сохраняя при этом текущие состояния и значения устройств, накопителей, очередей, ячеек, датчиков случайных чисел. Таким образом, в приведенном выше фрагменте будет выполнено 3 цикла моделирования для *X1=10* , 50 и 100.

Аналогично используются и карты *CLEAR* , но в отличие от *RESET* она приводит к исходному состоянию всю модель в целом (кроме датчиков *RN1÷RN8*).

В *GPSS* имеется целый ряд других управляющих карт, которые здесь не рассматриваются.

32. СООБЩЕНИЯ СИСТЕМЫ *GPSS*

Все сообщения системы делятся на две основные группы: информация о выявленных ошибках и статистическая информация, полученная в процессе моделирования.

Ошибки, фиксируемые системой *GPSS*, разделяются на следующие группы:

ошибки, выявляемые Ассемблером *GPSS* на фазе просмотра программы пользователя;

ошибки, обнаруживаемые программой ввода;

ошибки выполнения;

ошибки, обнаруживаемые программами вывода;

предупреждающие сообщения при выполнении.

Информация, выводимая системой на фазе просмотра программы, располагается в распечатке сразу после блока, в котором найдена ошибка. Например,

```
LOGIG    R        1
ERROR    NUMBER (S) 2 .
```

Здесь зафиксирована ошибка номер 2 - неправильно заполнено поле операции (*LOGIG* вместо *LOGIC*). При этом система не приступает к выполнению программы, пока все ошибки подобного рода (ошибки в синтаксисе) не будут устранены. Полный перечень ошибок содержится в работах [2],[7]. Аналогично идентифицируются ошибки, обнаруженные программой ввода. В сообщении *ERROR NUMBER (S) W,X,Y,Z* следующим за картой, содержащей ошибку, *W, X, Y, Z* - номера ошибок. По этим номерам, пользуясь полным перечнем ошибок, легко установить их смысловое содержание (в некоторых случаях номер ошибки не указывается, а просто печатается сообщение: *ERROR IN ABOVE CARD* - ошибка в предыдущей карте).

По окончании просмотра программы печатается алфавитный список имен блоков, а также других объектов (функции, переменные), используемых в программе. Каждому символическому имени присваивается номер, после чего повторяется распечатка текста программы, но

уже без символической адресации:

```
BLOCK   NUMBER  SYMBOL    REFERENCES  BY CARD NUMBER
          2     ASG          30
         74     BGN 1       113          116
         83     BGN 11      111
         12     DR W00       28
         49     DR W01       99

FACILITY SYMBOLS  AND  CORRESPONDING  NUMBERS
          2       CPU
FUNCTION SYMBOLS  AND  CORRESPONDING  NUMBERS
          3       OTL 1
          4       OTL 2
          5       OTL 3
          6       OTL 4
```

Ошибки, обнаруженные системой на фазе выполнения в распечатке программы, кроме указания номера ошибки, снабжаются дополнительной информацией:

номер транзакта, обрабатываемого симулятором в данный момент времени;

номер блока, из которого он выходит;

номер блока, в который входит этот транзакт;

текущее системное время;

значение счетчика, определяющего момент окончания моделирования:

```
ERROR    NO     611
TRANS   1 FROM   6 TO  9  CLOCK 1 TERMINATIONS TO GO 1Ø
```

После фиксации такой ошибки процесс моделирования может продолжаться. Предупреждающее сообщение о возможной ошибке никогда не вызывает прекращения машинного моделирования.

После окончания моделирования системой выдается статистическая информация по всем блокам, устройствам, памятям, очередям, таблицам и т.п., используемым в программе. Мы рассмотрим только стандартный (обычный) вывод информации об объектах языка.

Б л о к и. Для каждого блока программы пользователя указываются значения его двух СЧА: количество транзактов, находящихся в этом блоке, и общее количество транзактов, прошедших через блок за время моделирования:

BLOCK	CURRENT	TOTAL
I	0	294
2	0	293
3	I	293

У с т р о й с т в а. Ниже приведен пример распечатки информации об устройстве

FACILITY	AVERAGE UTILIZATION	NUMBER ENTRIES	AVERAGE TIME/TRANS	SEIZING TRANS NO	PREEMPTING TRANS NO
2	143	9	58,11	81	7
10	2Ø4	8	1549,6	5	1

В первом столбце указан номер устройства, используемого в программе пользователя, во втором - коэффициент использования устройства, в третьем - количество занятий устройства, в четвертом - среднее время использования устройства одним сообщением, в пятом - номер транзакта занимающего устройства (после отработки модели) и, наконец, в шестом - номер транзакта, вызвавшего прерывание на этом устройстве (после отработки модели). Коэффициент использования устройства определяется отношением времени его занятости к общему времени моделирования.

О ч е р е д и

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENT	PERCENT ZEROES	AVERAGE T/TRANS	AVERAGE T/TRANS	QTABLE NUM.	CURRENT CON.
10	72	41,5	10	6	60.0	90.3	100.7		

I столбец - номер очереди;
2 - максимальная длина очереди;
3 - средняя длина очереди;
4 - общее число входов в очередь;
5 - число входов с нулевым временем ожидания;
6 - процент входов с нулевым временем ожидания;
7 - среднее время пребывания в очереди;
8 - среднее время пребывания в очереди (без учета входов с нулевым временем ожидания);

9 – в излагаемом варианте не используется;
10 – текущая длина очереди.

П а м я т и

STORAGE	CAPACITY	AVERAGE CONTENTS	AVERAGE UTILIZAT	ENTRIES T/TRANS	AVERAGE	CURRENT CONTENTS	MAXIMUM CONTENTS
1ф	1ффф	555.2	0.5552	3ффф	478.9	634	908

I столбец – номер (имя) памяти;
2 – объем памяти;
3 – среднее содержимое;
4 – коэффициент использования;
5 – число входов в память;
6 – среднее время пребывания сообщения в памяти;
7 – текущее содержимое памяти;
8 – максимальное содержимое памяти.

Т а б л и ц ы. Ниже приведен пример распечатки таблицы, заимствованный нами из работы [2] (табл. 2).

В верхней строке указано:
количество входов в таблицу (блок *TABULATE*);
среднее значение аргумента таблицы (табулируемого параметра);
среднее квадратичное отклонение аргумента;
общая сумма табулируемых значений аргумента.
(Сообщение *NON WEIGHTED* свидетельствует о табулировании в режиме "без взвешивания").

На каждый частотный интервал таблицы в распечатке выделяется одна строка. В первом столбце такой строки указывается верхняя граница данного частотного интервала. И далее:

2 столбец – число попаданий аргумента (табулируемой величины) внутрь данного частотного интервала;
3 столбец – относительная частота (число попаданий, отнесенное к общему числу входов в таблицу);
4 столбец – накопленная относительная частота (суммарное число значений аргумента, меньших верхней границы данного частотного интервала, в процентах от общего числа входов в таблицу). Фактически, накоп-

TABLE 2

ENTRIES IN TABLE	MEAN ARGUMENT	STANDART DEVIATION	SUM OF ARGUMENTS
6	59.166	I3.847	I35.000

UPPER LIMIT	OBSERVER FREQENCY	PER CENT OF TOTAL	CUMULATIVE PERCENTAGE	CUMULATIVE REMAINDER	MULTIPL OF MEAN	NON WEIGHTED DEVIATION FROM MEAN
IØ	Ø	.ØØ	.Ø	IØØ.Ø	.169	-3.550
2Ø	Ø	.ØØ	.Ø	I00.0	.338	-2.828
30	0	.00	.0	I00.0	.507	-2.I06
40	I	I6.66	I6.6	83.3	.678	-I.384
50	0	.00	I6.6	83.3	.845	- .66I
60	I	I6.66	33.3	66.6	I.0I4	.060
70	4	66.66	I00.0	.0	I.I83	.782

REMAINING FREQENCIES ARE ALL ZERO

ленная относительная частота является гистограммой интегральной функции распределения вероятностей, выраженной в процентах;

5 столбец – дополненная накопленная относительная частота (в процентах). Сумма содержимого 4 и 5 столбцов в каждой строке равна 100%;

6 столбец – верхняя граница данного частотного интервала, выраженная в единицах среднего значения аргумента (отношение содержимого I-го столбца данной строки к среднему);

7 столбец – отклонение верхней границы каждого частотного интервала от среднего значения аргумента в единицах среднего квадратичного значения

$$\left(\frac{\text{содержимое I-го столбца – ср.значение}}{\text{среднеквадратичное отклонение}}\right)$$

Содержимое 6 и 7 столбцов позволяет исследователю сопоставить полученное путем моделирования распределение вероятностей аргумента таблицы соответственно с распределениями Эрланга и нормальным [4].

Последняя строка табл. 2 свидетельствует о том, что в последний частотный интервал (в этом примере его нижняя граница равна 70) не попало ни одного значения аргумента. Если такие попадания при табулировании имели место, после распечатки основной информации о частотных интервалах печатается сообщение *AVERAGE VALUE OF OVERFLOW*, после которого указывается среднее значение аргумента таблицы в последнем частотном (или классовом [4]) интервале. Это среднее определяется как частность:

$$\frac{\text{сумма значений аргумента в последнем интервале}}{\text{число попаданий в последний интервал}}$$

Я ч е й к и *SAVEVALUE*

Стандартный вывод результатов моделирования предусматривает распечатку содержимого всех ячеек *SAVEVALUE* , используемых в программе пользователя, содержимое которых отлично от нуля:

CONTENTS OF (NONZERO) SAVE, LOC	*VALUE*	*FULLWORD LOC*	*SAVEVALUES VALUE*	*LOC*	*VALUE*
1	*1*	*26*	*10*	*35*	*60401*
101	*73*	*539*	*6*		

В столбцах *SAVE*, *LOC* и *LOC* указаны номера ячеек, в столбцах *VALUE* - содержимое соответствующих ячеек.

33. ВНУТРЕННЯЯ ОРГАНИЗАЦИЯ *GPSS*

Система *GPSS* в целом как программный продукт состоит из II отдельных модулей: 2 модуля Ассемблера (первый и второй проходы), 3 модуля ввода, 4 модуля симулятора (управление, выполнение, перераспределение памяти, модификация) и 2 модуля вывода. Из перечисленных модулей только модуль управления (собственно симулятор) находится постоянно в ОЗУ и осуществляет процесс имитации.

Динамика функционирования симулятора основана фактически на схеме событий [5], при этом событием считается любое изменение состояния моделируемой системы. Основной функцией симулятора является поддержание правильного хода часов системного времени и выяснение возможностей продвижения транзактов в программе модели. Симулятор оперирует с рядом информационных структур, основными из которых являются списки: список будущих событий (*FEC*), список текущих событий (*CEC*), список прерываний, список задержанных транзактов и другие.

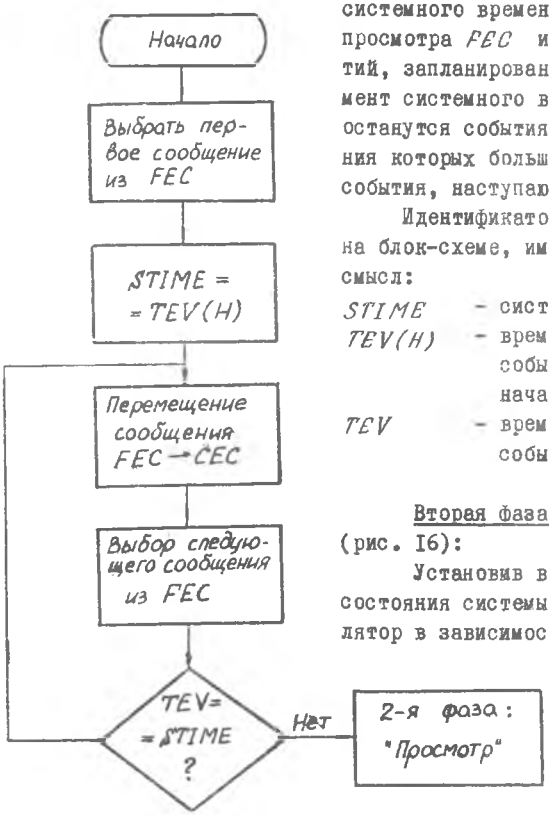
Внутренняя работа симулятора может быть разбита на три основные фазы:

1. Изменение значения системного времени.
2. Просмотр списка текущих событий.
3. Движение сообщений.

Первая фаза управления выполняется симулятором всегда, когда на текущий момент системного времени ни одно из активных сообщений, находящихся в списке текущих событий (*CEC*), не может быть продвинуто в программе модели и, кроме того, состояние системы не может быть изменено.

Алгоритм пересчета системного времени иллюстрируется блок-схемой, изображенной на (рис. I5).

Выбирая первое сообщение из *FEC*, симулятор присваивает системному времени *STIME* время очередной передвижки этого сообщения *TEV(H)* в программе модели и перемещает соответствующее событие в список *CEC*. Подобная процедура осуществляется для всех событий в *FEC*, время наступления которых равно *TEV(H)* (для всех событий, время наступления которых равно текущему значению



Р и с. I5. Пересчет системного времени

системного времени). При этом после просмотра *FEC* и перемещения событий, запланированных на текущий момент системного времени, в *FEC* останутся события, время наступления которых больше *STIME* , т.е. события, наступающие в будущем.

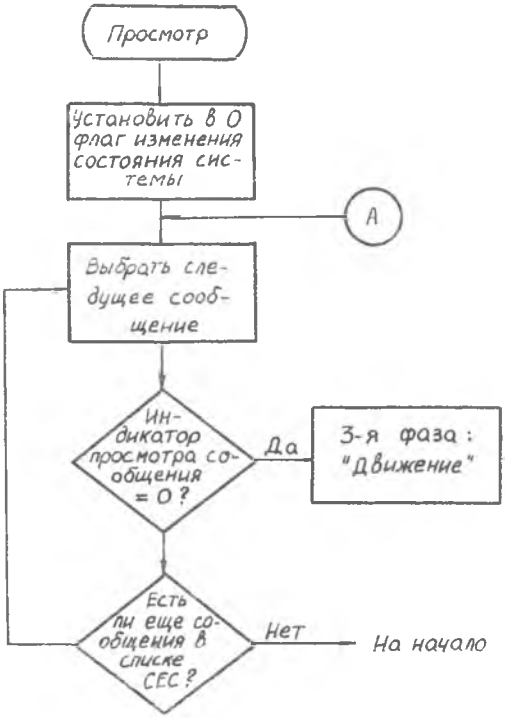
Идентификаторы, используемые на блок-схеме, имеют следующий смысл:

STIME - системное время;
TEV(H) - время возникновения события, стоящего в начале списка;
TEV - время возникновения события.

Вторая фаза "Просмотр" (рис. I6):

Установив в 0 флаг изменения состояния системы на 2 фазе, симулятор в зависимости от значения индикатора просмотра сообщения (0,I) решает вопрос, передать сообщение на 3-ю фазу или нет.

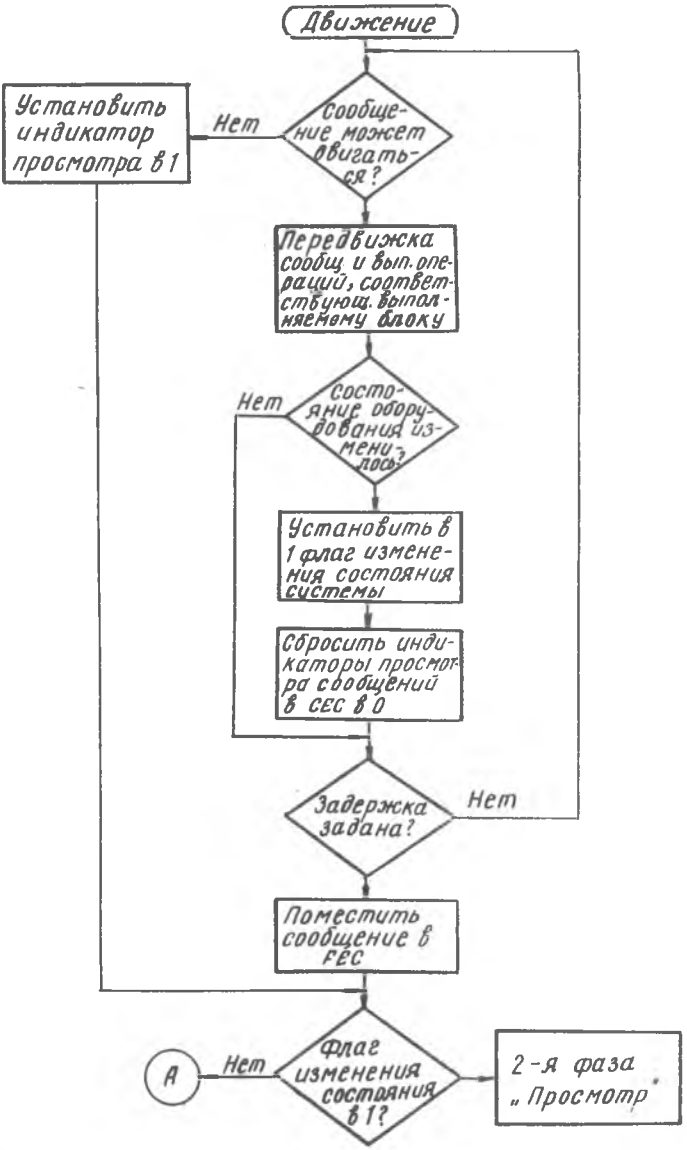
На 3-ю фазу передаются только активные сообщения, индикатор просмотра которых равен 0. Пассивные сообщения находятся в состоянии задержки, например, по причине занятости имитируемого оборудования. Такие сообщения не попадут на 3-ю фазу до тех пор, пока соответствующее оборудование не будет освобождено, т.е. пока не изменится состояние системы. Подобный принцип работы симулятора экономит машинное время, поскольку симулятор не предпринимает попыток продвижения пассивных сообщений в программе модели до изменения состояния системы.



Р и с. I6. Просмотр *СЕС*

Третья фаза "Движение" (рис. I7)

На 3-й фазе активные сообщения, выявленные на фазе просмотра, симулятор пытается продвинуть как можно дальше по программе модели. Если при этой передвижке меняется состояние системы, все пассивные сообщения, находящиеся в *СЕС* и задержанные по той или иной причине, получают статус активных. Их индикаторы просмотра устанавливаются в ноль. Если же при передвижке сообщений оказывается явно заданной их задержка (сообщение поступает в блок *ADVANCE*),



Р и с. I7. Фаза движения транзакта

то сообщение помещается в *FEC* . Таким образом, на 3-й фазе происходит передвижка активных сообщений, изменение состояния системы, пересмотр индикаторов сообщений и планирование будущих событий (помещение сообщения в *FEC*).

Программа моделирования размещается в памяти системы в виде последовательно расположенных слов, причем для каждого вида информации выделяется специальный блок памяти: информация об устройствах, памятях, очередях и др. Симулятор преобразует последовательность блоков программы в числовые элементы.

Взаимодействие между подобными информационными блоками осуществляется с использованием механизма ссылок, например: пусть в программе имеется блок *SEIZE 2ϕ* , и он по порядку является IO-м (отсчет идет от *SIMULATE*). Информация об этом блоке будет записана в IO-е слово памяти, выделенное под блоки, а в 20-е слово поля памяти, выделенное под устройства, будет записываться информация об устройстве, с которым работает этот блок. При этом каждый раз симулятор будет анализировать эти два слова совместно, хотя принадлежат они разным полям памяти. При программировании пользователь должен учитывать внутренние размеры программы и связанные с ними ограничения, накладываемые на программу пользователя. Эти ограничения определяются размером памяти системы. Ниже приведена краткая таблица (табл. 3), которая иллюстрирует, как изменяется максимально допустимое в программе пользователя количество основных элементов языка для вычислительных систем с объемами памятей в I28 К и 256 К.

Таблица 3

Тип объекта	Объем памяти на каждый объект в байтах	Число объектов	
		I28 К	256 К
Сообщение	I6	600	I200
Блоки	I2	500	I000
Устройства	28	I50	300
Таблицы	48	30	I00
Памяти	40	I50	300

Литература

1. Филип Дж. Кивиа. Языки моделирования. – В кн. Т.Нейлор и др. Машинные имитационные эксперименты с моделями экономических систем. – М.: Мир, 1975. 500 с.
2. Пакет прикладных программ для моделирования и исследования на ЭВМ дискретных систем. Описание языка. ПРО.309.007.Д1, 1977, 625 с.
3. Голованов О.Б., Дуванов С.Г., Смирнов Б.Н. моделирование сложных дискретных систем на ЭВМ третьего поколения. – М.: Энергия, 1978. 160 с.
4. Дерябкин В.Н., Кораблин М.А. Моделирование систем на ЭВМ. – Куйбышев: авиационный институт, 1977. 82 с.
5. Кораблин М.А., Смирнов С.Б. Монитор для имитационного моделирования систем с дискретными событиями. – Куйбышев: авиационный институт, 1980.
6. Мартин Дж. Системный анализ передачи данных. –М.: Мир, 1975, кн.2. 430 с.
7. Шрайбер Т. Дж. Моделирование на *GPSS* . – М.: Машиностроение, 1980. 592 с.

О Г Л А В Л Е Н И Е

Михаил Александрович Кораблин

ПРОГРАММИРОВАНИЕ МОДЕЛЕЙ ДИСКРЕТНЫХ СИСТЕМ
НА ЯЗЫКЕ *GPSS*

Учебное пособие

Редактор Н. В. К а с а т к и н а
Техн. редактор Н. М. К а л е н ю к
Корректор С. С. Р у б а н

Бумага писчая белая. Оперативная печать. Усл.п.л. 4.I.
Уч.-изд.л. 4,0. Тираж 500 экз. Заказ № 3498 Цена I5 коп.
Куйбышевский ордена Трудового Красного Знамени авиационный
институт имени С.П. Королева, г. Куйбышев, ул. Молодогвар-
дейская, I5I.

Областная типография имени В.П. Мяги, г. Куйбышев, ул.
Венцека, 60.