рисунке. Другой вариант определения входных и выходных элементов заключается в том, что пользователь присваивает этим элементам определенные номера, например «1» — для входного, «0» — для выходного элементов.

Объем интерпретируемых таблиц может быть сокращен, если для представления линейных частей модульной программы использовать гнездовые списки [11].

В заключение отметим, что для пользования описанным языком модульного программирования достаточно минимального предварительного знакомства с ним. Данный подход предполагается применить для оперативного создания модульных программ при автоматизации сбора и обработки данных в физических экспериментах.

ЛИТЕРАТУРА

1. R. M. Brown, M. A. Fisherkeller, A. E. Gromme, J. V. Levy. The SLAG high-energy spectrometer data acquisition and analysis system.— "Proc. of the IEEE", 1966, vol. 54, № 12.
2. Н. Н. Воробьева, Л. С. Нефедьева. Язык общений в системах приема и обработки физической информации.— Препринт № 10-4595. Дубна, Изд. ОИЯИ, 1969.
3. Ю. Я. Кузьмин. Об одной альтернативе программирования эксперимента.— В кн.: Кибернетизация научного эксперимента. (Учен. зап. Латв. Гос. ун-та им. П. Стучки). Рига, 1973, т. 196, вып. 5.
4. I. Klingels. Some thoughts on the future of modular programming.— "Data Proc.", London, 1971, vol. 13. № 4.
5. Е. А. Жоголев. Система модульного программирования (СИМП).— В кн.: Вычислительные методы и программирование. Вып. XVII. М., Изд. МГУ, 1971.
6. Е. А. Жоголев. Принципы построения многоязычной системы модульного программирования.— «Кибернетика», 1974, № 4.
7. И. Н. Парасюк, И. В. Сергиенко. Некоторые вопросы разработки и исследования одного класса универсально-специализированных автоматизированных систем обработки данных.— «Кибернетика», Киев, 1973, № 6.
8. И. Н. Парасюк, И. В. Сергиенко, Н. И. Тукалевская. Универсально-специализированная автоматизированная система обработки данных на ЦВМ (система УСОД).— «Упр. сист. и маш.», Киев, 1974, № 2.
9. W. G. Repsher. BELLFLOW draws flow diagrams automatically.— "Bell. Laboratories Record", New York, 1971, vol. 49, № 7.
10. Обработка данных и программирование. Схемы алгоритмов и программ. Правила выполнения. ГОСТ 19427-74.
11. А. И. Китов. Программирование информационно-логических задач. М., «Сов. радио», 1967.

Поступила в редакцию 18 февраля 1975 г.

УДК 681.3.06

С. В. БРЕДИХИН, П. М. ПЕСЛЯК

(*Новосибирск*)

ПРОСТАЯ СИСТЕМА ПРОГРАММИРОВАНИЯ ДЛЯ САМАС

ВВЕДЕНИЕ

Система программирования SICS (аббревиатура от simple interactive CAMAC system) предназначена для описания и исполнения простейших действий с аппаратурой САМАС. Основные пользователи

SICS — инженеры, которым по роду своей деятельности приходится писать небольшие программы с целью проверки и наладки отдельных элементов CAMAC оборудования. Разрабатывая эту систему, мы стремились создать простой и надежный «инструмент» для этих целей. Мы надеемся, что SICS станет удобной основой для:

1) обучения приемам программирования CAMAC;
2) тестирования и наладки CAMAC оборудования;
3) написания и исполнения небольших программ для экспериментов, не критичных ко времени.

Систему SICS составляют интерпретатор и операционная система. Обе программы являются резидентными. Статья посвящена описанию двух языков системы: языка операторов — входного языка интерпретатора — и языка директив — языка управления заданиями операционной системы.

Настоящий текст — это предварительное сообщение о системе; в нем не затрагиваются вопросы архитектуры системы и технологии ее реализации. Предполагается, что читатель знаком с основными понятиями и определениями стандарта CAMAC [1, 2].

Основное внимание в статье уделяется входному языку интерпретатора. Цель авторов — не определить, но описать этот язык. Всюду, где это представлялось возможным, мы старались избегать формальных определений. Для наглядности синтаксис языка представлен не в нотации Бекуса — Наура, а графически, в виде диаграмм Конвея. Семантика операторов задается традиционным описанием. С учетом предполагаемого применения языка и возможности реализации его для малой машины он содержит сравнительно небольшое число основных понятий.

Система SICS была реализована весной 1975 г. для машин семейства «Электроника-100». Конечно, реализация наложила свой отпечаток в виде ряда ограничений, подавляющее большинство которых обусловлено желанием реализовать систему на базовом комплекте оборудования вычислительной машины.

КОНФИГУРАЦИЯ ОБОРУДОВАНИЯ

В этом пункте мы укажем необходимый (возможный) состав оборудования вычислительной машины и CAMAC оборудования.

Необходимый состав оборудования вычислительной машины:

* процессор семейства «Электроника-100» с 4*K* памяти. Блок арифметического расширителя не требуется;
* терминал (клавиатура и устройство печати). В качестве терминала можно использовать машинку «Консул-254», алфавитно-цифровой дисплей "Videoton-340" либо телетайп модели «ASR-33»;
* фотосчитыватель;
* перфоратор.

SICS допускает возможность манипулирования с CAMAC оборудованием, организованным одним из следующих способов:

1. CAMAC оборудование состоит из одного крейта.
2. CAMAC оборудование сконфигурировано в ветвь.

В первом случае необходимый состав CAMAC оборудования таков:

* крейт с источником питания;
* контроллер для вычислительной машины семейства «Электроника-100». Во втором случае, в дополнение к перечисленному выше, необходимо иметь:
* драйвер ветви, выполненный в виде модуля CAMAC;

* контроллеры типа *A*. Число таких контроллеров равно числу крейтов в ветви (максимально — 7).

Конфигурация CAMAC оборудования в виде ветви приведена на рис. 1. По соглашению, принятому в SICS, крейт, контроллер которого соединен с вычислительной машиной, имеет номер нуль.

Система практически не зависит от типа контроллера, подключаемого к вычислительной машине. Модификация системы под конкретный контроллер требует незначительных переделок. Для ее осуществления необходимо знать точные ответы (т. е. последовательности команд ввода — вывода) на следующие вопросы:

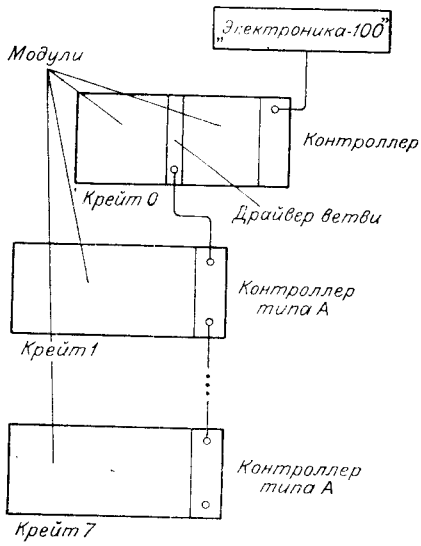

Рис. 1.

1. В чем состоит процесс инициации контроллера?
2. Как исполняется цикл CAMAC?
3. Как осуществляется проверка общего запроса от крейта?
4. Как найти запрос от конкретного модуля?
5. Как осуществить обмен данными между вычислительной машиной и шинами данных CAMAC?
6. Как проверить сигнал по шине *Q*?
7. Как проверить сигнал по шине *X*?

КРАТКИЙ ОБЗОР СИСТЕМЫ

Алгоритм или программа для вычислительной машины состоит из двух основных частей: описания *действий*, которые должны быть осуществлены, и описания *данных*, над которыми совершаются эти действия. Действия описываются так называемыми *операторами*, данные представляются значениями *переменных*. В языке *операторов* — фиксированное число переменных. Значение каждой переменной задается непосредственным определением. Переменная может принимать значения из фиксированного множества значений, определенного для этой переменной. Значениями переменных являются положительные целые числа, которые могут быть представлены в десятичной, восьмеричной или двоичной форме. Любой комбинации значений ***cnaf*** может быть присвоено имя. Эти имена мы будем называть *идентификаторами*.

Программа, написанная на входном языке, состоит из *строк*. В каждой строке может быть записан только один оператор. Каждому оператору приписан *номер оператора*. Операторы могут вводиться в произвольном порядке, однако в программе они будут расставлены в порядке возрастания номеров. Роль *меток* для реализации взаимных ссылок выполняют номера операторов. Поскольку каждый оператор имеет номер, то все операторы являются помеченными. Каждая строка программы должна заканчиваться маркером *конец оператора*.

Перейдем к обзору операторов входного языка.

Наиболее фундаментальным среди операторов является оператор **исполнить** (***exec***). Он предписывает исполнить CAMAC операцию ***f*** в крейте ***c***, в модуле, расположенном на позиции ***n***, по субадресу ***a***. Оператор **присвоить** (***let***) служит для непосредственного определения зна-

чений переменных. В результате исполнения оператора **определить** (***define***) с идентификатором связывается определенный набор значений переменных ***cnaf***. Такие идентификаторы используются в операторе ***use***, который является другой формой оператора ***exec***. Операторы **взять** (***get***), **положить** (***put***), **ввести** (***input***) оперируют с 24-разрядным программным регистром. Далее этот регистр будем называть *виртуальным*. В результате исполнения оператора ***get*** состояние *R*-шин, которое было зафиксировано во время исполнения предыдущего CAMAC цикла, будет перенесено в виртуальный регистр. В результате исполнения оператора ***put*** указанная в операторе переменная получает в качестве значения содержимое этого регистра. В момент исполнения оператора ***input*** содержимое виртуального регистра может быть изменено пользователем с клавиатуры. В результате исполнения оператора **печать** (***print***) содержимое виртуального регистра может быть выведено на терминал в десятичной, восьмеричной или двоичной форме.

Операторы **переход** (***go***) и **если** (***if***) изменяют естественный порядок исполнения операторов (в порядке возрастания номеров). Эти операторы непосредственно указывают своего «преемника». В операторе ***go — to*** это делается непосредственным указанием номера «преемника». Оператор ***if*** изменяет порядок исполнения операторов в зависимости от **условия**. Условия определяются наличием запроса с указанной позиции (***l***), наличием хотя бы одного запроса в системе (***lam***), сигналом по шине ***q*** или значением кода, установленного на ключевом регистре передней панели машины (***s***). Оператор ***go — subroutine*** задает **переход к подпрограмме.** Выход из подпрограммы осуществляется по оператору **возврата** (***return***).

Любая последовательность операторов программы может быть повторена в **цикле**. Для определения параметра цикла (числа повторений) служит оператор ***do***. Оператор ***next*** завершает группу операторов, повторяемых в цикле.

Для **приостановки** исполнения программы или ее **завершения** служат операторы ***break*** и ***end*** соответственно. Для тех пользователей, которым необходим непосредственный доступ к исполнению машинных команд, оставлена «лазейка» в виде оператора **вызова** (***call***), осуществляющего вызов непосредственно исполняемой программы, написанной в машинном коде.

ОСНОВНЫЕ ПОНЯТИЯ

Синтаксис входного языка представлен графически в виде *диаграмм*. Каждая диаграмма описывает способ порождения множества синтаксически верных конструкций языка. Для того чтобы породить некоторую синтаксически верную конструкцию, следует продвигаться по выбранному на диаграмме пути, избегая «острых углов». По мере продвижения следует выписывать встречающиеся на пути терминальные символы. Такими символами в языке являются английские слова (например, ***if***, ***let***), некоторые буквы латинского алфавита (***c***, ***n***, ***a***, ...), арабские цифры и специальные знаки (%, ″, ...). Терминальные символы указаны в разрывах линий диаграммы. Кроме терминальных символов, могут встречаться обращения к поддиаграммам и ссылки на понятия, которые объясняются в тексте. В диаграммах эти понятия и обращения к поддиаграммам обозначены русскими словами и выделены строчными буквами. Правила пользования поддиаграммами очевидны. Символы и слова, которые указаны над линией диаграммы, являются необязательными и служат для удобства чтения. При составлении языковых конструкций их можно опускать, однако при выводе программы на терминал или перфоленту они появятся в тексте про-

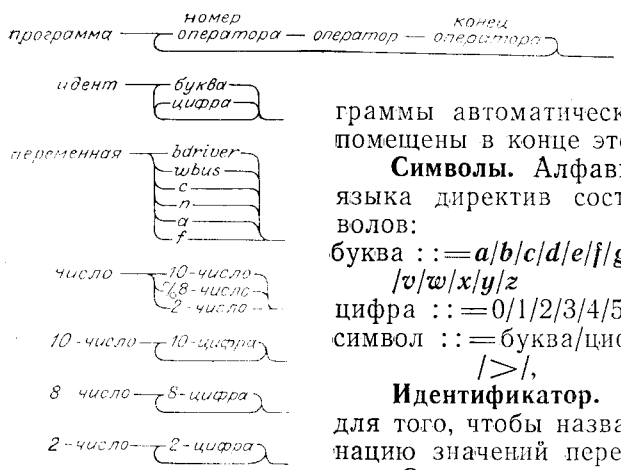

Рис. 2.

граммы автоматически. Диаграммы (рис. 2, 3) помещены в конце этого пункта.

Символы. Алфавит языка операторов и языка директив состоит из следующих символов:

буква ::=***a/b/c/d/e/f/g/h/i/j/k/l/m/n/o/p/q/r/s/t/u/v/w/x/y/z***

цифра ::=0/1/2/3/4/5/6/7/8/9

символ ::=буква/цифра/пробел/"/%/(/)/./;/=/>/,

Идентификатор. Идентификатор служит для того, чтобы назвать определенную комбинацию значений переменных ***cnaf*** своим именем. Заметим, что эта комбинация может состоять из одной переменной. Идентификатором является произвольная последовательность букв и цифр, начинающаяся с буквы. Следует помнить, что два одинаковых идентификатора не могут встречаться в одной программе.

Переменная. В языке операторов зафиксированно восемь переменных, а именно: ***bdriver***, ***wbus***, ***c***, ***n***, ***a***, ***f***, ***l*** и ***s***. Каждая переменная имеет строго определенный смысл и может принимать значения из фиксированного диапазона значений, который определен для каждой переменной. Смысл переменных и соответствующие диапазоны значений таковы:

bdriver — номер позиции модуля «Драйвер ветви», который устанавливается в нулевом крейте. Значением переменной ***bdriver*** может быть десятичное число из диапазона от 1 до 23 включительно;

wbus — код на *W*-шинах, который будет установлен при исполнении следующего цикла САМАС в крейте с номером нуль. Значением переменной ***wbus*** может быть число из диапазона от 0 до 77777777 включительно;

c — номер крейта	$0 \leqslant c \leqslant 7$;
n — позиция в крейте	$1 \leqslant n \leqslant 31$;
a — субадрес	$0 \leqslant a \leqslant 15$;
f — код функции	$0 \leqslant f \leqslant 31$;
l — номер запроса	$1 \leqslant l \leqslant 23$;
s — код на ключевом регистре	$0 \leqslant s \leqslant 7777$.

Двум последним переменным (***l*** и ***s***) значения присваиваются только непосредственным определением в операторе ***if***.

Число. Допускается использование трех типов чисел: десятичные, восьмеричные и двоичные. На диаграммах они обозначены как 10-число, 8-число и 2-число соответственно. Для указания типа числа используются предупреждающие символы: точка (.) для представления двоичных чисел, процент (%) для представления восьмеричных чисел. Число, перед первой цифрой которого не стоит ни один из указанных предупреждающих символов, считается десятичным.

Номер оператора. Каждый оператор программы снабжается номером, который называется номером оператора. Номер оператора выбирается программистом и записывается с первой позиции строки. Он может быть любым целым десятичным числом от 1 до 2047 включительно. Операторы программы можно вводить в произвольном порядке: система сама расставит их в порядке возрастания номеров.

Оператор. Оператор программы предписывает, какие действия следует выполнить интерпретатору в момент исполнения программы, либо служит для связи некоторой комбинации переменных ***cnaf*** с определен-

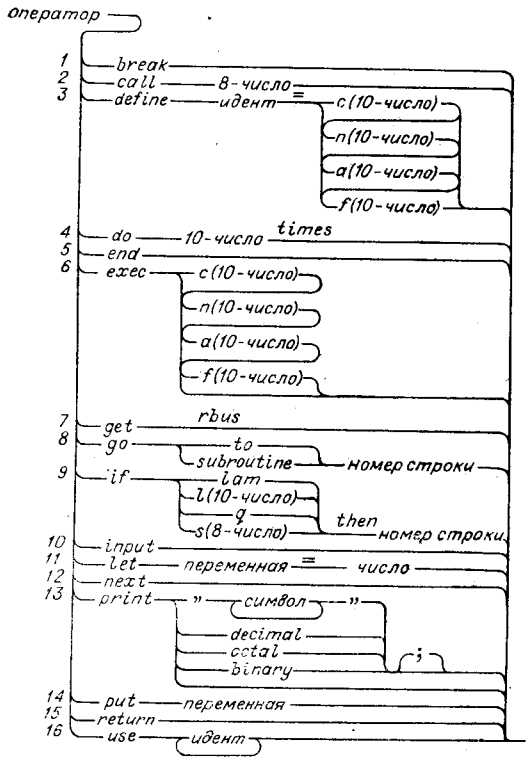

Рис. 3.

ным идентификатором. Оператор отделяется от номера оператора одним или несколькими пробелами.

Конец оператора. Каждый оператор программы должен заканчиваться маркером конца оператора.

Директива. Директива — это строка, введенная пользователем с клавиатуры терминала, начинающаяся с буквы.

С помощью директив пользователь дает задание операционной системе на выполнение определенных действий, например запуск программы, вывод программы и т. п.

ЯЗЫК ОПЕРАТОРОВ

Синтаксис языка операторов содержится в диаграммах. В этом пункте мы приводим семантику операторов языка. Операторы перечисляются в алфавитном порядке. Номера линий диаграммы «оператор» и номера абзацев, описывающих семантику операторов, совпадают. Описание сопровождается простыми примерами и замечаниями, которые относятся к конкретной реализации интерпретатора.

1. ***break***. Оператор ***break*** приостанавливает исполнение программы. При этом на печать выводится сообщение:

break in line ****

Здесь **** означает номер оператора ***break***, приостановившего исполнение программы. Для того чтобы продолжить исполнение программы, следует воспользоваться директивой ***continue***.

Пример: 10 ***break***

2. ***call***. Этот оператор передает управление на непосредственно исполняемую (т. е. неинтерпретируемую) подпрограмму, написанную в машинных кодах. После исполнения подпрограммы управление передается оператору, следующему за оператором ***call***. В качестве параметра оператора ***call*** указывается абсолютный адрес точки входа в подпрограмму (восьмеричное число).

Пример: 20 ***call*** 140

Вызываемая подпрограмма оформляется стандартным образом:

точкавхода, 0

.../тело подпрограммы

JMP I точкавхода

Замечание. Использование оператора ***call*** таит в себе потенциальную опасность, поскольку управление передается непосредственно исполняемой программе.

3. ***define***. Оператор ***define*** используется для присвоения имени (идентификатора) некоторой комбинации значений переменных ***cnaf***. Значениями переменных могут быть только десятичные числа.

Пример: 15 ***define adc=c(0)n(5)***
20 ***define read=a(0)f(0)***

(В 15-й строке модулю, размещенному в пятой позиции крейта нуль, присвоено имя ***adc***. В строке 20-й нулевому субадресу и операции с кодом нуль присвоено имя ***read***.)

4. ***do***. Оператор цикла ***do*** позволяет повторить заданное число раз группу операторов, следующих за оператором ***do***. Замыкает эту группу оператор ***next***. Число повторений задается десятичным числом в диапазоне от 0 до 4095. После выполнения указанного числа повторений управление будет передано на оператор, следующий за ***next***. Допускается вложенность операторов ***do***. Глубина вложенности неограниченна.

Пример: 10 ***do*** 10
. . .
40 ***do*** 20
. . .
100 ***next***
. . .
140 ***next***

(Участок программы от 40-й до 100-й строки будет исполнен 200 раз; от 10-й до 40-й и от 100-й до 140-й строки — 10 раз).

5. ***end***. Оператор ***end*** завершает исполнение программы. На печать выводится сообщение ***ready***. Система переходит в состояние ожидания ввода директивы или оператора с терминала.

6. ***exec***. Этот оператор предписывает исполнить САМАС операцию с переменными ***cnaf***. В качестве значений переменных используются только десятичные числа. Любой набор значений переменных ***cnaf*** может быть указан в операторе явно. Если значение какой-либо переменной опущено, то берется старое ее значение. В частности, могут быть опущены все значения

Пример: 40 ***exec c(0)n(4)a(10)f(19)***
41 ***exec n(5)***
42 ***exec n(7)***

(Если в этом примере комбинация ***a(10)f(19)*** означает «Разрешение модуля», то в результате исполнения операторов 40, 41, 42 будут разрешены модули 4.5 и 7 в крейте с номером нуль).

7. ***get***. По этому оператору состояние *R*-шин нулевого крейта, которое было зафиксировано в момент исполнения последнего САМАС цикла, переносится в виртуальный регистр. Старое содержимое виртуального регистра теряется.

Пример: 45 ***get rbus***

8. ***go***. Возможны две конструкции оператора перехода ***go***.

8.1. В результате исполнения конструкции ***go — to*** осуществляется безусловный переход к выполнению оператора с номером, который указан в качестве параметра.

Пример: 50 ***go to*** 40

8.2. В результате исполнения конструкции ***go — subroutine*** осуществляется вызов подпрограммы, написанной на входном языке. Параметр оператора указывает на номер первого оператора подпрограммы. Выход из подпрограммы осуществляется по оператору ***return***. По выходе управление будет передано на оператор, непосредственно следующий за оператором ***go — subroutine***. Допускается вложенность вызова подпрограмм. Глубина вложенности неограниченна.

Пример: 52 ***go subroutine*** 1000

9. ***if***. Операторы условного перехода (***if***-операторы) приводят к пропуску или выполнению некоторых операторов программы. В зависимости от типа условия возможны четыре конструкции оператора ***if***.

9.1. Конструкция ***if — lam***. Эта конструкция предназначена для выяснения, есть ли запросы от модулей, расположенных в нулевом

крейте. Если хотя бы один модуль выставил запрос, то управление будет передано на оператор с указанным номером. Если запросов нет, то управление будет передано на оператор, непосредственно следующий за оператором ***if — lam.***

Пример: 60 ***if lam then*** 105

9.2. Конструкция ***if — l.*** Эта конструкция предназначена для выяснения, выставил ли конкретный модуль, расположенный в нулевом крейте, запрос. Если к моменту исполнения ***if — l*** оператора модуль, номер позиции которого указан в качестве значения переменной ***l***, выставил запрос, то управление будет передано на оператор с номером, указанным в качестве параметра. Если указанный модуль запроса не выставил, то управление будет передано на оператор, непосредственно следующий за оператором ***if — l.*** В качестве значения переменной ***l*** может быть указано десятичное число из диапазона $1 \leqslant l \leqslant 23$.

Пример: 70 ***if l*** (5) ***then*** 110

9.3. Конструкция ***if — q.*** Эта конструкция предназначена для выяснения, присутствовал ли сигнал на шине *Q* в последнем цикле САМАС. (Имеется в виду шина *Q* нулевого крейта). Если сигнал *Q* был, то управление будет передано на оператор с указанным номером. В противном случае управление будет передано на оператор, непосредственно следующий за оператором ***if — q.***

Пример: 80 ***if q then*** 200

9.4. Конструкция ***if — s.*** Эта конструкция предназначена для выполнения условного перехода по значению кода, установленного на ключевом регистре вычислительной машины. Если в момент исполнения оператора ***if — s*** значение кода совпадает со значением переменной ***s***, заданным в операторе ***if — s***, то управление будет передано на оператор с номером, указанным в качестве параметра ***if — s*** оператора. В противном случае управление будет передано на оператор, непосредственно следующий за оператором ***if — s.*** В качестве значения переменной ***s*** может быть указано восьмеричное число из диапазона $0 \leqslant s \leqslant 7777$.

Пример: 90 **if s** (77) **then** 1000

10. ***input.*** Оператор ***input*** предназначен для ввода с клавиатуры одного числа в момент исполнения этого оператора. Это число может быть представлено в десятичной, восьмеричной или двоичной форме. При исполнении оператора ***input*** происходит следующее: приостанавливается исполнение программы и на печать (экран) выводится напоминающий символ >. Система переходит в состояние ожидания ввода с клавиатуры терминала.

Непосредственно за символом > пользователь может ввести число в любой форме представления. (О представлении чисел в той или иной форме смотри пункт «основные понятия».) Ввод числа завершается по вводу маркера конца оператора, при этом двоичный эквивалент введенного числа размещается на виртуальном регистре (старое значение виртуального регистра теряется). Исполнение программы продолжается. Следующим исполняемым оператором будет оператор, непосредственно следующий за оператором ***input.***

Пример: 100 ***input***

Замечание. Если пользователь в ответ на символ > вводит строку, начинающуюся с буквы ***s***, то на терминал будет выведено сообщение ***stop***, затем ***ready***. Исполнение программы прекращается.

11. ***let.*** Оператор ***let*** служит для присвоения значения переменным ***driver***, ***wbus***, ***c***, ***n***, ***a*** и ***f***. Значением переменной может быть десятичное, восьмеричное или двоичное число, которое отделяется от имени переменной разделителем (знаком равенства или пробелом). Старое значение переменной теряется.

Пример: 202 ***let f***=25
210 ***let wbus***=1

12. ***next***. Оператор ***next*** завершает группу операторов, исполняемых в цикле. (Подробности смотри в описании оператора ***do***.)

Пример: 70 ***next***

13. ***print***. Оператор ***print*** служит для вывода на печать (экран) содержимого виртуального регистра или любой, заранее определенной последовательности символов. Символ точка с запятой (;) в конце оператора ***print*** указывает на то, что по завершении печати не следует осуществлять перевод строки, возврат каретки. Вывод осуществляется с текущей позиции каретки (маркера). Возможны пять конструкций оператора ***print***.

13.1. ***print — «text»***. Эта конструкция служит для вывода любой последовательности символов, заключенной в кавычки.

Пример: 150 ***print «this is example»***

13.2 ***print — decimal***. Вывод содержимого виртуального регистра в десятичной форме (8 десятичных цифр). Незначащие нули подавляются. Содержимое регистра сохраняется.

Пример: 160 ***print decimal***

13.3 ***print — octal***. Вывод содержимого виртуального регистра в восьмеричной форме (8 восьмеричных цифр). Незначащие нули подавляются. Содержимое регистра сохраняется.

Пример: 165 ***print octal***

13.4. ***print — binary***. Вывод содержимого виртуального регистра в двоичной форме (24 двоичных цифры). Незначащие нули подавляются. Содержимое регистра сохраняется.

Пример: 170 ***print binary***

13.5. ***print***. Осуществляется перевод строки, возврат каретки.

Пример: 180 ***print***

14. ***put***. В результате исполнения оператора ***put*** переменной ***bdriver***, ***wbus***, ***c***, ***n***, ***a*** или ***f*** в качестве значения присваивается содержимое виртуального регистра. Старое значение переменной теряется. Содержимое виртуального регистра сохраняется.

Пример: 200 ***put wbus***
210 ***put f***

15. ***return***. По оператору ***return*** осуществляется выход из подпрограммы. Управление передается на оператор, непосредственно следующий за оператором перехода к подпрограмме (***go — subroutine***).

Пример: 250 ***return***

16. ***use***. Этот оператор предписывает исполнить САМАС операцию со значениями переменных ***cnaf***, определенных с помощью идентификаторов, которые, в свою очередь, определены в операторах ***define***.

Пример: 300 ***use read adc***

(В этом примере оператор ***use*** использует идентификаторы ***read adc***, которые были определены операторами ***define*** (смотри пример к оператору ***define***). В результате его исполнения в крейте 0 в модуле, расположенном на позиции с номером 5, по субадресу 0 будет выполнена САМАС операция с кодом 0).

ЯЗЫК ДИРЕКТИВ

Управление операционной системой происходит с помощью директив. Директивы вводятся с клавиатуры с первой позиции строки. Перед директивой не ставится никакого номера. С помощью директив можно ввести программу с перфоленты (***ptape***), запустить программу (***run***), вывести текст программы на терминал (***list***) или на перфоленту (***plist***), продолжить исполнение программы после ее приостановки (***continue***), уничтожить программу (***kill***). Директивы ***run***, ***list*** и ***plist*** могут иметь параметры, суть которых — номера операторов. Для опи-

сания директив с параметрами принята следующая нотация: параметры, которые можно опускать, указаны в квадратных скобках.

1. ***ptape***. Для ввода программы, подготовленной на перфоленте, служит директива ***ptape***. Перфолента с текстом программы перфорируется в коде *ASCII* по стандартным правилам подготовки. Общий вид директивы:

ptape

2. ***run***. Для запуска программы служит директива ***run***. Она имеет один необязательный параметр, который указывает на номер того оператора, с которого следует начать исполнение программы. Общий вид директивы ***run***:

run [*p*]

По директиве ***run*** производятся следующие действия:

1. Значения переменных ***wbus***, ***bdriver***, ***c***, ***n***, ***a*** и ***f*** устанавливаются в нуль.
2. Инициируется крейт-контроллер и выдается сигнал ***z***.

Затем начинается исполнение программы.

Процесс исполнения приостанавливается по оператору ***break***, при этом на терминал выводится сообщение ***break in line*** ****

break in line ****

Здесь **** означает номер строки оператора ***break***. Текущее состояние всех регистров и переменных сохраняется. Можно продолжить исполнение программы с помощью директивы ***continue***.

Процесс исполнения прекращается по оператору ***end***, при этом выводится сообщение ***ready***.

Прервать исполнение программы можно нажатием любой клавиши на клавиатуре терминала, при этом выводится сообщение ***ready***. После сообщения ***ready*** продолжить исполнение программы нельзя, но можно заново возобновить ее исполнение по директиве ***run***.

Если при исполнении оператора ***exec*** или ***use*** с конкретным значением переменных ***cnaf*** по шине *X* не было получено ответа, подтверждающего законность операции, то на терминал будет выведено сообщение:

no x. line ****. ***c***(*)***n***(**)***a***(**)***f***(**)

Здесь **** указывает на номер строки оператора, во время исполнения которого не был получен сигнал ***x***; в скобках указывается текущее значение переменных ***cnaf***. Исполнение программы после вывода этого сообщения продолжается.

3. ***continue***. Директива ***continue*** служит для продолжения исполнения приостановленной программы. Общий вид директивы:

continue

4. ***list***. Вывод программы на терминал осуществляется по директиве ***list***. Общий вид директивы:

list[***p***1[, ***p***2]]

Эта директива имеет два необязательных параметра ***p***1 и ***p***2, которые определяют, какую часть программы следует выводить. Значениями параметров являются номера операторов. Если параметры отсутствуют, то выводится текст всей программы. Если указан один параметр, то выводится программа, начиная со строки с номером ***p***1. Если указаны оба параметра, то выводится текст программы, начиная со строки с номером ***p***1 и до строки с номером ***p***2 включительно.

5. ***plist***. Вывод исходного текста программы на перфоленту осуществляется с помощью директивы ***plist***. Общий вид директивы:

plist [*p*1[, *p*2]]

Эта директива имеет два необязательных параметра *p*1 и *p*2, значения и смысл которых совпадают с приведенными в описании директивы ***list***. Перфорируется программа в коде *ASCII*, начало и конец перфоленты оформляются лидерами.

6. ***kill***. Для того чтобы уничтожить в памяти машины текст всей программы, следует воспользоваться директивой ***kill***. Общий вид директивы:

kill

СООБЩЕНИЯ ОБ ОШИБКАХ

При обнаружении ошибки, произошедшей при вводе программы или при ее исполнении, на терминал выводится сообщение о типе ошибки. Форма сообщения такова:

error ** ***in line*** ****

Здесь ** означает код ошибки, **** — номер неверного оператора.

Если ошибка обнаружена в директиве, то сообщение выглядит так:

error **

Сообщение об ошибке выводится сразу же, при ее обнаружении. Если ошибка была допущена при вводе оператора, то этот оператор теряется. Полная таблица ошибок приведена в приложении.

Приложение

ТАБЛИЦА ОШИБОК

error 1 — переполнение памяти;
error 2 — переполнение при переводе числа;
error 3 — не опознана директива;
error 4 — не опознан оператор;
error 5 — неверный диапазон параметров;
error 6 — при переводе числа встретилась цифра, большая, чем основание системы счисления;
error 7 и ***error*** 8 — эти ошибки возникают при несовпадении числа операторов ***do***, ***next*** или ***go*** — ***subroutine***, ***return***;
error 9 — директиве ***continue*** не предшествовало исполнение оператора ***break***;
error 10 — в операторе ***go*** или ***if*** параметр перехода указывает на номер несуществующего оператора;
error 11 — номер оператора больше 2047;
error 12 — идентификатор дважды определен операторами ***define***;
error 13 — в операторе ***use*** используется неопределенный идентификатор;
error 14 — переменные ***cnaf***, значения которых ранее были определены операторами ***define***, перекрываются в операторе ***use***, т. е. значение одной и той же переменной определено несколькими идентификаторами.