граммирования для CAMAC [16] придерживаются взглядов [15], эта работа содержит ряд конкретных примеров.

Нам приятно поблагодарить А. Н. Гинзбурга, который стимулировал эту работу, Ю. К. Постоенко за ценные обсуждения и замечания, а также Н. Г. Щербакову, которая выполнила значительную часть переводов иностранной литературы,

ЛИТЕРАТУРА

1. EUR 4100. Revised Description and Specification. ESOME Committee, 1972.
2. EUR 4600. Specification of the Branch Highway and CAMAC Grate Controller Type A. ESONE Committee, 1972.
3. F. R. Golding, A. C. Peatfield, K. Spurling. CAMACRO-an Aid to CAMAC Interface Programming.— CAMAC Bulletin, 1972, N 5.
4. CAMAC package: CAMPACK. STAP-Schumberger, October, 1972.
5. CAMCONV. Conversational Language for Writing and Executing CAMAC Functions. STAP-Schumberger, October, 1972.
6. R. M. Keyser. COMP11, a CAMAC-Oriented Monitor for the PDP11.— CAMAC Bulletin, 1973, N 7.
7. F. May, H. Halling, K. Petreczek. FOCAL Overlay for CAMAC Data and Command Handling.— CAMAC Bulletin, 1971, N 1.
8. F. May, W. Marschik, H. Halling. A. FOCAL Interrupt Handler for CAMAC.— CAMAC Bulletin, 1973, N 6.
9. H. Halling, K. Zwoll, W. John. CAMAC Averlay for Single-User BASIC and Modification of 8-user BASIC for the PDP11. CAMAC Bulletin, 1973, N 6.
10. I. Bals, E. Agostino. An Extendes BASIC Language for CAMAC Programming.— CAMAC Bulletin, 1973, N 7.
11. R. F. Thomas, Specifications for Standard CAMAC Subroutines.— CAMAC Bulletin, 1973, N 6.
12. M. Sarquiz, P. Valois. An Approach to a CAMAC Language.— CAMAC Bulletin, 1971, N 2.
13. P. Wilde. A CAMAC Language.— CAMAC Bulletin, 1972, N 3.
14. Proposal for a CAMAC Language.— CAMAC Bulletin (Supplement), 1972, N 5.
15. P. Wilde. A Survey Paper.— ESONE-Software Working Group, RHEL, Chilton, 14 November, 1970.
16. Ю. Ф. Рябов. В. П. Хомутников. О входных языках для систем, использующих CAMAC. Препринт ЛИЯФ, № 24, Л., 1973.

Поступила в редакцию 19 февраля 1974 г.

УДК 681.3.06 : 51

П. М. ПЕСЛЯК, Э. А. ТАЛНЫКИН

(Новосибирск)

ЯЗЫК СИСТЕМНОГО ПРОГРАММИРОВАНИЯ ДЛЯ МИНИ-ЭВМ

Введение. Авторы этого сообщения на протяжении трех лет занимались вопросами построения математического обеспечения ЭВМ различных классов. В круг задач входило: обеспечение нестандартных внешних устройств (графопостроителей, дисплеев, аналого-цифровых и цифроаналоговых преобразователей, устройств считывания полутоновых изображений и т. п.), а также систем для сбора и обработки данных в реальном масштабе времени.

Основная часть всех этих работ проводилась с использованием языков ассемблера, хотя неадекватность этого инструмента целям разарботки достаточно очевидна. Использование макрогенераторов дает лишь некоторое механическое облегчение, но не освобождает от необходимости помнить о машинных кодах.

В данной статье предлагается язык для написания системных программ (далее ЯСП или язык) для ЭВМ типа М-6000 (HP2116, HP2100). Разработка языка и реализация проводились с учетом основных требований к языку системного программирования, сформулированных в [1]. ЯСП избавляет программиста от использования машинных кодов и в то же время в полной мере отражает архитектуру, систему команд и структуру базового математического обеспечения ЭВМ. В языке отражены все команды машины, кроме команд, изменяющих последовательность выполнения операций, которые используются неявно в виде богатого набора управляющих конструкций (циклов, условных выражений и т. п.).

Структуры данных не зафиксированы жестко, а определяются программистом, что повышает гибкость языка и крайне необходимо при составлении системных программ. В целях повышения эффективности транслятор производит локальную экономию памяти (на уровне отдельной функции или программы) и экономию времени уничтожением излишних косвенных адресаций, а также за счет того, что некоторые выражения вычисляются во время трансляции (например, $A+3+5$ будет оттранслировано в $A+8$).

Настоящая статья не является формальным описанием языка, и в отдельных местах, чтобы не затруднять чтение, полный синтаксис не приводится.

1. Основные концепции. Основными величинами, которыми оперируют все языковые конструкции, являются «значения». На множестве значений определены бинарные и унарные операции, в результате которых появляются новые значения. Кроме того, имеются «ячейки» (переменные), обладающие свойством принимать значения. С некоторыми ячейками может быть связано одно или несколько имен, представляемых идентификаторами в тексте программы. В реальной машине значения представляются совокупностью двоичных наборов, способных разместиться в одном машинном слове, а «ячейка» есть просто ячейка оперативной памяти. Операции над значениями обеспечиваются системой команд машины. Некоторые из значений можно интерпретировать как указатели на определенную ячейку — абсолютный машинный адрес.

Имя переменной в контексте исполняемой конструкции программы представляет значение, принимаемое этой переменной, кроме случая, когда

$$\uparrow абв$$

представляет значение, являющееся указателем на ячейку с именем «*абв*», т. е. ее абсолютный адрес. Символ «↑» не является операцией. Так, например, конструкции

$$\uparrow 5 \quad \uparrow(x+1) \quad \uparrow\uparrow x$$

ничего не выражают уже семантически — мы можем создать указатель на ячейку, а не на значение! В языке есть операция противоположного смысла, а именно «Взять содержимое». Эта одноместная операция обозначается символом «.» и имеет семантику — взять значение ячейки, указателем на которую является значение операнда. Реализация этой операции обеспечивается косвенной адресацией в системе команд машины. В данном случае конструкции

$$.5 \quad .(\uparrow x+1) \quad .x$$

законны. Значением первой будет значение, принимаемое ячейкой с абсолютным адресом 5, второй — значение, принимаемое ячейкой, следующей за ячейкой с именем x, третьей — значение, принимаемое ячейкой, указателем на которую является значение ячейки с именем x.

Из соображений удобочитаемости введены скобки излечения значения, которые эквивалентны операции «.» и обозначаются «⟨» и «⟩». Таким образом, вместо выражения

$$.(.(.x+7)+.(..y+5))$$

можно записать

$$\langle\langle.x+7\rangle+\langle..y+5\rangle\rangle$$

Эта форма записи представляется нам более наглядной.

Основное свойство вычислительного процесса состоит в изменении значений переменных. В языке это реализуется двухместной операцией «⇒» — «Заслать в», несущей семантику: значение левого операнда заслать в ячейку, указателем на которую является значение правого операнда. Эта операция в некотором смысле представляет оператор присваивания АЛГОЛа в языке. Так, $A:=B$ запишется в виде $B \Rightarrow \uparrow A$.
Конструкция же типа $B \Rightarrow A$; $B \Rightarrow .A$; $B \Rightarrow (A+1)$ не имеют аналога в АЛГОЛе.

В ЯСП нет операторов. Каждая исполняемая конструкция языка есть выражение, т. е. нечто, поставляющее значение, которое может быть использовано далее для конструирования более сложных выражений. Таким образом можно, например, складывать и перемножать блоки, циклы и т. п.

Как уже отмечалось, в языке нет типов данных, как **целый и вещественный** в АЛГОЛе, вернее, есть всего один тип — «двоичное изображение», которое интерпретируется в соответствии с семантикой производимых над ним операций.

В языке нет также зафиксированных структур данных, как массив в АЛГОЛе. Вместо этого программисту дается возможность определять свои собственные структуры данных. Это означает, что описывается алгоритм доступа и привязывается к переменной. Область памяти, на которую во время исполнения указывает значение этой переменной, приобретает структуру, определяемую алгоритмом доступа. Привязка алгоритма доступа к переменной происходит статически, а засылка указателя может происходить как статически, так и динамически во время исполнения программы. Снабдить указатель алгоритмом доступа можно также непосредственно на момент одного обращения. (Подробнее см. п. 12).

В ЯСП нет операторов перехода, и соответствие между статическим текстом программы и динамическим вычислительным процессом, порождаемым этой программой, устанавливается цепочкой текстуальных индексов и динамических указателей [2]. В ЯСП текстуальный индекс может быть снабжен «текстуальным значением». Текстуальное значение в языке можно сравнить с регистром ЭВМ, содержащим результат предыдущей операции, причем оно остается неизменным при проверках условия в условном выражении, явных и неявных проверках условия выхода из цикла, при вызове функций и т. п.

2. Лексика. Идентификаторы в программе на ЯСП представляются следующим образом:

⟨идентификатор⟩ : :=⟨буква⟩|⟨идентификатор⟩⟨буква⟩|
⟨идентификатор⟩⟨цифра⟩|
⟨идентификатор⟩—

Некоторые идентификаторы зарезервированы под служебные слова и не могут быть использованы под имена ячеек, функций, структур и т. п. Список служебных слов здесь не приводится, так как мы не описываем синтаксис языка полностью. Это могут быть разделители **начало, конец, если** и т. д., одноместные и двухместные операции ***and, or, not, neg, zero*** и т. д. Обозначение служебных слов обычными идентификаторами обусловлено спецификой печатающих устройств и уст-

ройств подготовки, тем не менее, в этой заметке мы будем выделять служебные слова (полужирным шрифтом).

В тексте программы могут встречаться также разделители

$$\langle|\rangle|=|\neq|,\ |.|(|)|[|]|\Rightarrow|;\ |+|-|\uparrow|\ \text{и т. п.}$$

Любая последовательность символов, кроме комбинации (*/), заключенная между двухсимвольными разделителями (/*) и (*/), рассматривается как комментарий и игнорируется транслятором. Комментарии могут встречаться между двумя любыми лексически полными единицами текста программы.

3. Значения. Значения в программе на ЯСП представляются десятичными, восьмеричными и составными числами:

⟨значение⟩ : : =⟨простое число⟩|⟨текст⟩|⟨составное число⟩
⟨простое число⟩ : : =⟨число 8⟩|⟨число 10⟩
⟨число 10⟩ : : =⟨цифра⟩|⟨цифра⟩⟨число 10⟩
⟨число 8⟩ : : =⟨цифра 8⟩*B*|⟨цифра 8⟩⟨число 8⟩
⟨цифра⟩ : : =⟨цифра 8⟩|8|9
⟨цифра 8⟩ : : =0|1|2|3|4|5|6|7
⟨составное число⟩ : : =!⟨знак⟩⟨составное без знака⟩
⟨знак⟩ : : =пусто|+|−
⟨составное без знака⟩ : : =⟨простое число⟩|(⟨список полей⟩)
⟨список полей⟩ : : =⟨поле⟩|⟨список полей⟩, ⟨поле⟩
⟨поле⟩ : : =⟨длина⟩/⟨значение поля⟩
⟨длина⟩ : : =⟨простое число⟩
⟨значение поля⟩ : : =⟨простое число⟩
⟨текст⟩ : : =«любая последовательность печатных символов»

Десятичные и восьмеричные числа интерпретируются обычным образом. В случае если значение не помещается в ячейку, старшие разряды отбрасываются и выдается диагностика. Составное число представляется в общем виде:

$$!\pm(\text{П1/ЗП1, П2/ЗП2, ..., П}n\text{/ЗП}n)-$$

иинтерпретируется как описано ниже.

Вначале вычисляются значения полей, т. е. двоичные наборы, продолженные влево нулями. Слово разбивается на поля, начиная с крайнего левого разряда, и условно предполагается неограниченным справа. Далее значения полей заносятся в соответствующие поля с отбрасыванием старших разрядов. Результатом является двоичный набор, разместившийся в одном слове, с обратным знаком, если в представлении составного числа был знак (—). В случае если поля не покрывают слово, остаток слова заполняется нулями перед сменой знака. Если суммарная длина полей превышает размер слова или значение не помещается в поле, выдается диагностика.

Текст интерпретируется следующим образом: если в кавычках стоит один символ, то значением будет число, равное его коду, если же более одного, то символы упаковываются в формате, принятом в данной ЭВМ. Если в текст необходимо вставить символ (»), то его следует повторить два раза.

Примеры:

10 99 77*B* !—5 !—30*B* !—(5/7, 3/2, 8/177*B*)

4. Память. Программа оперирует одним или несколькими сегментами памяти. Сегмент состоит из некоторого числа последовательных ячеек (слов), каждая из которых состоит, в свою очередь, из определенного числа двоичных разрядов (16 в нашей реализации). Несколько последовательных разрядов ячейки могут быть выделены в поле. Каждая ячейка сегмента, как и сам сегмент, может быть поименована. Доступ же имеется к любому полю внутри ячейки, т. е. можно, например, к содержимому поля одной ячейки прибавить содержимое поля

другой ячейки и заслать результат в некоторое поле третьей ячейки.

Отведение сегментов памяти, инициализация их содержимого и привязка имен происходят статически (во время трансляции и загрузки). (Динамическое распределение памяти в ЭВМ, на которой производилась реализация, повлекло бы необходимость в административной системе, что нежелательно для системных программ и исключило бы возможность для оттранслированных программ работать без всякого окружения.

Определение сегментов происходит посредством описаний, например:

локальный $M[1000]$, P, $X=1$, $U[5]=(1, 2, «ABC»)$
глобальный $R[100]$, K
внешний Z, Y
функция $F(A, B)=.A+.B$
глобальная функция $G(X, Y)=$
(если $.X>.Y$ **, то возврат** 0 **иначе** $.X \Rightarrow Y$)
структура Массив $[N, I, Y]=$(Массив$+(.N+.I)+.Y$)
глобальная структура $M1(I)=(M1+.I)$
локальный T **синоним** $\uparrow M+500$, $H[10]$, $W=\uparrow M$
значение один$=1$, два$=2$, сто$=100$,
код_операции$=!(1/0, 4/17B)$

Описание **локальный** определяет сегмент указанной длины; и первой ячейке сегмента приписывается имя, локализованное в блоке, содержащем это описание. Имя функции или структуры тажке локализуется в блоке.

Описания с приставками **глобальный** и **внешний** могут появляться только в самом внешнем блоке и служат для коммуникации отдельно оттранслированных программ. Если, например, в одной программе описан **глобальный** X [100], а во второй — **внешний** X, то при условии, что программы будут вместе загружены в память, место под сегмент X будет отведено один раз, а имя X в обеих программах будет привязано к этому сегменту.

Описание **синоним** не определяет сегмента, а только привязывает имя к ячейке уже определенного сегмента. Выражение в описании **синоним** должно вычисляться во время трансляции.

Описание **значение** также не определяет сегмента и даже не именует никакую ячейку, а просто текстуально привязывает идентификатор к значению, и появление этого идентификатора далее эквивалентно появлению самого значения. Это описание также локализуется в блоке.

Если в описании, определяющем сегмент, длина явно не указана, то сегменту отводится одна ячейка. Сегмент-функциям и структурам отводится столько ячеек, сколько потребует тело функции или структуры. Сегмент памяти во время исполнения программы содержит или данные, или программные коды (в случае функций и структур). Содержимое сегмента может быть инициализировано, т. е. прямо во время трансляции ячейкам сегмента приписываются некоторые значения. Сегменты функций и структур всегда инициализируются.

5. Выражения. Выражение является основной конструкцией языка и описывается следующим синтаксисом:

⟨выражение⟩ ::= ⟨пусто⟩ | ⟨терм⟩ |
⟨выражение⟩⟨одноместная операция⟩ |
⟨выражение⟩⟨двухместная операция⟩⟨терм⟩
⟨терм⟩ ::= ⟨значение⟩ | ⟨значение переменной⟩ |
⟨терм⟩ | ⟨составное выражение⟩ |
⟨блок⟩ | ⟨структурное значение⟩ |
↑⟨имя переменной⟩ | ⟨цикл⟩ |
⟨условное выражение⟩ | ⟨выражение выбора⟩ |
⟨⟨составное выражение без скобок⟩⟩ |
⟨конструкция выхода⟩

Из синтаксиса видно, что все двухместные операции выполняются слева — направо, т. е. нет никакого старшинства. Это отражает структуру выполнения операций в машине с сумматором, где последний всегда является одним из операндов и после выполнения содержит результат.

Одноместные операции отражают команды машины, в которых всего один операнд — сумматор, а результат операции — снова на сумматоре.

Среди двухместных можно выделить арифметические операции: сложение (+), вычитание (—), умножение (×), деление (/) и выделение остатка (***rel***); сдвиги: логические сдвиги (***lsl*** — влево, ***lsr*** — вправо), арифметические сдвиги (***asl*** — влево, ***asr*** — вправо) и циклические сдвиги (***rtl*** — влево, ***rtr*** — вправо); логические операции: поразрядное логическое умножение (***and***), поразрядное логическое сложение (***or***) и поразрядное сложение по ***mod***2 (***xor***); операции сравнения: $>$, $<$, $\geqslant$, $\leqslant$, $=$, $\neq$. Все перечисленные операции имеют традиционную семантику. Операции сравнения вырабатывают значения: 0 — да и (—1) — нет. Двухместная операция «⇒» — «Заслать в» уже обсуждалась выше.

Среди одноместных операций есть типичные: ***abs*** — взять абсолютную величину значения, ***neg*** — взять с обратным знаком, ***not*** — поразрядная логическая операция НЕ, ***zero*** — занулить результат. Кроме того, имеется богатый набор (более 200) операций над значением обеспечиваемых системой команд машины, на которой производилась реализация.

6. Составные выражения и блоки.

⟨составное выражение⟩ ::= (⟨составное выражение без скобок⟩)
⟨составное выражение без скобок⟩ ::= ⟨выражение⟩|
⟨составаное выражение без скобок⟩
⟨выражение⟩|
⟨составное выражение без скобок⟩; ⟨выражение⟩
⟨блок⟩ ::= **начало** ⟨описания⟩ ⟨составное выражение без скобок⟩
конец

Значением составного выражения или блока при нормальном выходе является текстуальное значение перед замыкающим символом «)» или **конец**, т. е., попросту, значение последнего исполняемого выражения. При искусственном выходе значение определяется в соответствии с п. 11.

7. Условные выражения.

⟨условное выражение⟩ ::= **если** ⟨выражение⟩ **то** ⟨выражение⟩|
если ⟨выражение⟩ **то** ⟨выражение⟩
иначе ⟨выражение⟩

Выражение **если** *Выр1* **то** *Выр 2* интерпретируется стандартным образом. Вычисляется значение *Выр1* и, если оно истинно, вычисляется значение *Выр2*, которое и становится значением условного выражения. Если же значение *Выр1* — ложь, то значением условного выражения является текстуальное значение, существовавшее при входе.

Значением выражения **если** *Выр1* **то** *Выр2* **иначе** *Выр3* является значение *Выр2* или значение *Выр3* в зависимости от того, истина или ложь значение *Выр1*.

8. Циклы. В языке есть семь типов циклов:

⟨цикл⟩ ::= **повторять** ⟨выражение⟩|
пока ⟨выражение⟩ **делать** ⟨выражение⟩|

пока не ⟨выражение⟩ **делать** ⟨выражение⟩|
делать ⟨выражение⟩ **пока** ⟨выражение⟩|
делать ⟨выражение⟩ **пока не** ⟨выражение⟩|
увеличивая ⟨идентификатор⟩ **от** ⟨выражение⟩
до ⟨выражение⟩ **делать** ⟨выражение⟩|
уменьшая ⟨идентификатор⟩ **от** ⟨выражение⟩
до ⟨выражение⟩ **делать** ⟨выражение⟩

Первая конструкция **повторять** выражение означает неограниченное повторение исполнения выражения — тела цикла. Если заглянуть вперед и учесть конструкции выхода, этот тип цикла является наиболее общим. Он представляет цикл как повторяющийся процесс, причем условие завершения явно не указывается, оно может быть любым и в любом месте тела цикла. В остальных кострукциях явно указывается условие завершения и место его проверки — до или после выполнения тела цикла.

Вторая конструкция **пока** *Выр1* **делать** *Выр2* может быть представлена: **повторять если** *Выр1* **то** *Выр2* **иначе выхцикл.**

Третья конструкция **пока не делать** *Выр1 Выр2* представляется: **повторять если** *Выр1* ***not*** **то** *Выр2* **иначе выхцикл.**

Четвертая и пятая конструкции цикла отличаются от второй и третьей только тем, что сначала выполняется тело цикла, а затем происходит проверка условия выхода.

Итеративный цикл **увеличивая** *I* от *Выр1* **до** *Выр2* **делать** *Выр3* может быть представлен:

начало
локальные *I*, *граница* *значение*
( ) ⇒ *значение*;
(*Выр1*) ⇒ ↑ *I*;
(*Выр2*) ⇒ ↑ *граница*
повторять
если *I* > *граница* **то выхблок** *значение*
иначе (*значение*; (*Выр3*) ⇒ ↑ *значение* *I* + 1 ⇒ *I*)
конец

Конечно, не следует думать, что указанные действия действительно выполняются в рабочей программе. Такое представление приведено, чтобы явно показать, что происходит с текстуальным значением при выполнении цикла. Заметим, что переменная цикла локализуется в теле цикла, т. е. при выходе из цикла она лексически отсутствует.

Итеративный цикл с отрицательным шагом почти ничем не отличается от только что разобранной конструкции.

Общим значением для всех циклов является значение последней (во времени) исполняемой операции в теле цикла, или текстуальное значение при входе, если таковой не было.

9. Выражения выбора.

⟨выражение выбора⟩ : : = **выбор** ⟨список выражений⟩ **из**
⟨составное выражение без скобок⟩ **кв**|
выбор ⟨список выражений⟩ **из**
⟨составное выражение без скобок⟩
иначе ⟨выражение⟩ **кв**

В наиболее общей форме и выражении выбора:

выбор *Вр1*, *Вр2*, . . ., *Врn* **из**
Выр1
Выр2
.
.
.

Выр к **иначе** *Выраж* **кв**

сначала вычисляется значение *Вр1*, затем *Выр* с индексом, равным этому значению. Если такого индекса нет, то вычисляется *Выраж* и вся эта процедура повторяется для *Вр2*, ..., *Врп*. Значением выражения выбора является значение последней (во времени) исполняемой операции в выражении *Выр1*, *Выр* к или *Выраж*. Выражение выбора гораздо удобнее переключателя в АЛГОЛе, так как оно избавляет от необходимости использовать много меток и переходов и в совокупности с другими возможностями языка дает очень полезные конструкции, например:

выбор *I* **из**

↑*F*1; ↑*F*2; ↑*F*3 /* это указатели на сегменты функций*/

кв (*x*)

10. Функции. С каждой функцией связан сегмент памяти, отведенный ей в соответствии с ее описанием. Описание функции выглядит следующим образом:

Функция *F*(*П*1, *П*2, ..., П*n*) = *Выражение*, где *П*1, *П*2, ..., П*n* — формальные параметры. Вызов же функции выглядит как

Выр0 (*Выр1*, *Выр2*, ..., *Вырn*), где значение *Выр0* — указатель на сегмент функции, а *Выр1*, ..., *Вырn* — фактические параметры.

При описании тела функции формальные параметры *П*1, *П*2, ..., *Пn* нужно рассматривать как переменные, значения которых служат указателями на действительные параметры (сегменты памяти). При передаче параметров действует единый механизм: если фактический параметр есть имя переменной или сегмента, то передается указатель на «действительный объект», если же параметром является выражение, то вычисленное значение засылается во временную ячейку и передается указатель на нее. Таким образом, при желании можно избегать побочных эффектов применения функций. Пусть, например, имеется фрагмент программы:

локальный *x*, *y*

функция *F*(*A*, *B*) = (*A* ⇒ *B*)

и пусть имеем для вызова функции

$$\uparrow F(x, y) \text{ и } \uparrow F(x, (y)).$$

Как в первом, так и во втором случаях значением функции будет значение переменной *x*. В первом случае будет наблюдаться побочный эффект, заключающийся в том, что переменная *y* примет значение переменной *x*. Во втором случае побочного эффекта нет.

11. Конструкции выхода. В языке имеется возможность выйти из любой описанной выше конструкции до ее полного завершения:

⟨конструкция выхода⟩ ::= **возврат** ⟨выражение⟩ |
⟨тип выхода⟩ ⟨число уровней⟩
⟨выражение⟩

⟨число уровней⟩ ::= [⟨число 10⟩]

⟨тип выхода⟩ ::= **выход** | **выхцикл** |
выхблок | **выхусл** |
выхсост | **выхвыб**

возврат означает возврат из функции со значением, равным значению выражения. Во всех остальных случаях тип выхода означает тип конструкции, из которой должен произойти выход, число уровней показывает число вложенных конструкций данного типа, которые нужно покинуть, а значение выражения будет значением на выходе. Смысл типов выхода следующий: **выхцикл** — выход из цикла, **выхблок** — выход из блока, **выхусл** — выход из условного выражения, **выхсост** — выход из составного выражения, **выхвыб** — выход из выражения выбора, **выход** включает любой из перечисленных выше типов выходов.

12. Структуры. В языке нет жестко зафиксированных структур дан-

ных, но есть возможность определять свои собственные структуры. Для этого необходимо определить структуру посредством описания, т. е. задать алгоритм доступа, например:

структура *Массив* [*I*] = (*Массив*+.*I*—1)

В описании структуры сам индентификатор структуры используется как переменная, значением которой служит указатель на область памяти, а параметры в алгоритме доступа рассматриваются как указатели на ячейки, содержащие фактические параметры обращения, аналогично параметрам функции. Значением выражения в описании должен быть указатель на ячейку, к которой определился доступ. После определения структуры как алгоритма доступа нужно привязать этот алгоритм к некоторой переменной. Область памяти, на которую во время исполнения будет указывать значение переменной, автоматически приобретает нужную структуру. Эта процедура называется «Навести структуру на указатель» и проделать ее можно статически:

навести *Массив* **на** *x*, *y*, *z*;

Если раньше в программе присутствовали описания

локальный ПАМ1 [1000], ПАМ2 [100],

x=↑ПАМ1, *y*, *z*;

то законны следующие конструкции:

.*x*[*I*+5] ⇒ *x*[100]

↑ПАМ2 ⇒ ↑*y*

.*y*[20] ⇒ *x*[5]

(↑ПАМ2+50) **как** *Массив* [30]

В первой строке значение (*I*+5)-й ячейки сегмента ПАМ1 засылается в сотую ячейку того же сегмента. Здесь указатель на область определен статически. Во второй строке в ячейку *y* засылается указатель на сегмент ПАМ2, после чего *y* [20] будет означать указатель на двадцатую ячейку сегмента ПАМ2. В четвертой строке вычисляется указатель на 50-ю ячейку сегмента ПАМ2 и на него на момент обращения наводится структура «Массив».

Так как в языке адресуются только целые ячейки, структурировать область памяти можно также с точностью до ячейки, сохраняя двусторонний доступ (чтение, занесение). Если же нужна структура с односторонним доступом (чтение), то можно наводить более «тонкие» структуры, например:

```
структура ТЕТРАДА [I] =
начало
    локальный R
    ⟨ТЕТРАДА+((.I—1 ⇒ ↑R) lsr 2)⟩
  lsr(12—(R and 3B*4)) nd 17B
    /* здесь lsr — логический сдвиг вправо,
а and — поразрядное логическое умножение */
конец
```

определяет алгоритм доступа к памяти структурированной по тетрадам (имеется в виду 16-разрядная длина слова). Теперь если имеем описание

локальный ПАМ [100], ТЕТР=↑ПАМ

навести ТЕТРАДА **на** ТЕТР

то конструкция

ТЕТР [25]

даст значение 25-й тетрады, а не указатель. Ниже в качестве примера приведен фрагмент программы на ЯСП:

```
начало
структура Матрица [I, J] = (матрица+(. I*10)+. J),
Строка [I] = (Строка + . I),
Столбец [J] = (Столбец+(. J*10))
⋮
```

```
локальный x[100], y[100], z[100],
          Mx=↑x, My=↑y, Mz=↑z
навести Матрица на Mx, My, Mz
/* Фрагмент, вычисляющий матричное произведение z=x+y*/
увеличивая I от 0 до 9 делать
начало
локальный стрx, стру
навести Строка на стрx, стру
        Mx[I, 0] ⇒ ↑стрx
        Mx[I, 0] ⇒ ↑стрz
увеличивая J от 0 до 9 делать
начало
локальный стлу
навести Столбец на стлу
        My[0, J] ⇒ стлу
        0+(увеличивая K от 0 до 9 делать
        (     )+(.стрx[K]+.стлу[K])) ⇒ стрz [J]
    конец
конец
.
.
.
конец
```

13. Реализация. Структурно транслятор состоит из трех блоков: лексического анализатора, синтаксического анализатора с семантическими подпрограммами и генератора кода (по существу, являющимся макрогенератором). Выходом транслятора служит правильная программа на языке ассемблера, получаемая за один просмотр исходной программы.

В качестве метода трансляции выбран несколько модернизированный метод диаграмм переходов [3, 4]. При построении транслятора выдерживался принцип: как можно больший процент кодов перенести в постоянную часть, т. е. таблицы, диаграммы и т. п.

Синтаксис языка и его семантика закодированы в диаграмме, и работа синтаксического анализатора состоит в передвижении по диаграмме, исполняя по пути расставленные там семантические подпрограммы. Семантические подпрограммы работают в очень конкретном контексте, так что большая их часть сводится к выдаче некоторой макрокоманды. Макрокоманда, попадая на макрогенератор, разворачивается далее в команды ассемблера. Логика лексического анализатора также закодирована в своей диаграмме, и имеются свои семантические подпрограммы. Таким образом, синтаксический и лексический анализаторы отличаются диаграммами и семантическими подпрограммами, а также уровнем понятий, с которыми они оперируют. Синтаксический анализатор получает от лексического конструкции типа идентификатор, число, служебное слово, разделитель и т. п., причем числа уже переведены в двоичный формат, а служебные слова и разделители заменены внутренними кодами. Лексический анализатор работает с объектами типа буква, цифра, которые он получает от программы более низкого уровня.

В процессе разработки транслятор сначала был написан на своем входном языке, затем на языке ассемблера было реализовано некоторое подмножество языка и на нем написан полный транслятор, на котором уже была оттранслирована первая версия. После этого производилось еще несколько проходов с целью модернизации подпрограмм ввода и вывода.

При кодировании транслятора ни разу не ощущалась необходимость в операторе перехода или в машинном коде.

Заключение. Как сам язык, так и методы реализации представляют собой модернизацию и адаптацию к конкретным условиям существующих методов и концепций. Язык может использоваться для построения системных программ широкого назначения (компиляторов, интерпретаторов, мониторных систем и т. п.). В настоящее время разрабатывается математическое обеспечение магистральной модульной системы для сбора и обработки экспериментальных данных, построенной на принципах САМАС. Большое влияние на структуру языка оказали работы [1, 2, 4—6]. При реализации много идей заимствовалось из [1, 3, 4, 7].

Хочется выразить благодарность С. В. Бредихину и В. А. Мелешихину, оказавшим неоценимую помощь при реализации. Продолжительные дискуссии с В. А. Марковым помогли в определении основных концепций языка. А. Н. Гинзбургу мы всецело обязаны появлением этой статьи.

ЛИТЕРАТУРА

1. А. П. Ершов. Технология разработки систем программирования.— «Системное и теоретическое программирование». Сборник трудов. Новосибирск, 1972.
2. E. W. Dijkstra. Go to Statement Considered Harmful.— Comm. of he ACM, 1968, v. 11, N 3.
3. Е. М. Конвей. Проект делимого компилятора, основанного на диаграммах перехода. Современное программирование. Языки для экономических расчетов. М., «Советское радио», 1967.
4. N. Wirth. The Design of a PASCAL Commpiler.— Software-Practice and Experience, 1971, v. 1, 309—333.
5. N. Wirth. PL360 — A Programming Language for 360 Computers.— Journal of the ACM, 1968, v. 15, N 1.
6. W. A. Wulf, D. B. Russell, A. N. Habermann. BLISS: A Language for Systems Programming.— Comm. of the ACM, 1971, v. 14, N 12.
7. АЛЬФА — система автоматизации программирования. Под ред. А. П. Ершова. Новосибирск, «Наука», 1967.

Поступила в редакцию 7 февраля 1974 г.