В.И.Гололобов Б.Г. Чеблаков Г.Д.Чинин

Описание языка ЯРМО

МАШИННО—НЕЗАВИСИМОЕ ЯДРО

247

ПРЕПРИНТ

НОВОСИБИРСК • 1980

Академия наук СССР Сибирское отделение
Вычислительный центр

В.И.Гололобов, Б.Г.Чеблаков, Г.Д.Чинин

ОПИСАНИЕ ЯЗЫКА ЯРМО

МАШИННО-НЕЗАВИСИМОЕ ЯДРО

Препринт
247

Новосибирск 1980

АННОТАЦИЯ

Описывается замкнутое машинно-независимое подмножество языка ЯРМО, предназначенного для написания комплексов системных программ.

ПРЕДИСЛОВИЕ

В 1973 г. в Новосибирском филиале Института точной механики и вычислительной техники АН СССР были начаты работы по созданию языка программирования высокого уровня с целью применения его в практическом системном программировании. В рамках этого направления были разработаны язык программирования для ЭВМ БЭСМ-6 [1], получивший название ЯРМО (язык реализации машинно-ориентированный), и базирующаяся на его основе система программирования.

В течение 1974-1977гг. эта система интенсивно применялась в практических и экспериментальных разработках НФ ИТМ и ВТ и ВЦ СОАН СССР. С ее помощью были реализованы несколько трансляторов компилирующего и интерпретирующего типов, большие компоненты систем машинной графики, аналитических преобразований, АСУ и т.д. Всем этим задачам характерно умеренное требование к эффективности рабочих программ. Точнее сказать - эффективность рабочих программ достигается в них, в основном, за счет алгоритмов функционирования, а не качества их объектного кода. Ясно, что на таких задачах проявились главным образом положительные стороны языка реализации высокого уровня. Развитые изобразительные средства дали возможность формулировать программы в терминах, близких к существу задачи и алгоритму ее решения. Механизмы конструирования и статического контроля данных, регулярность структур управления, защита имен и аппарат процедур позволили ускорить процессы написания и отлад-

ки программ.

С другой стороны, разработка на основе языка ЯРМО операционной системы для ЭВМ БЭСМ-6 и спецпроцессора МВК "Эльбрус-I" наряду с положительными моментами применения языка ЯРМО отчетливо показала и главный его недостаток - невысокую эффективность рабочих программ.

В 1976 г. с учетом опыта применения языка ЯРМО была разработана вторая его версия, в значительной степени свободная от недостатков первой. Обсуждение внешних требований к языку, обусловленных потребностями разработки программ, и обзор его свойств можно найти в работах [2, 3, 4, 5].

Вторая версия языка послужила реализационной базой для ряда разработок практического и экспериментального характера. В частности, на его основе заново реализована упомянутая выше операционная система [6]. Результаты пробной эксплуатации системы на ЭВМ БЭСМ-6 свидетельствуют о значительно возросшей эффективности по сравнению с первой реализацией.

Вместе с тем в процессе разработок отмечены недостатки языка и системы программирования, устранение которых должно повысить эффективность и расширить применимость языка. С этой целью разрабатывается третья версия языка. Не вдаваясь в существо предполагаемых изменений, отметим лишь, что основным практическим источником соображений по улучшению (устранению недостатков) языка является опыт разработки операционной системы.

В работе над языком и системой программирования кроме авторов настоящего документа принимали участие А.Ю. Бондарь, Б.И. Байсер, В.С. Нумеров, В.Б. Разгулин, Н.П. Терновая, Т.Ф. Чеблакова. Ф.Р. Цанг оказал существенное влияние на разработку языка в части придания ему черт, необходимых для написания программ операционных систем.

I. ВВЕДЕНИЕ

Язык Реализации, Машинно-Ориентированный на ЭВМ БЭСМ-6 (ЯРМО), предназначен для написания комплексов системных программ. ЯРМО является языком высокого уровня, предоставляющим программисту также и возможность прямого использования БЭСМ-6. Описание языка построено по строго последовательному принципу, т.е. изложение в любом месте текста опирается только на ранее приведенные факты и понятия. Каждая глава начинается с краткого обзора описываемых понятий.

Синтаксис определяемого понятия описывается в виде группы из одной или более возможных альтернатив, разделяемых двоеточиями. Альтернатива представляет собой последовательность конституент, каждая из которых является либо терминальным символом языка, либо обозначением некоторого понятия, либо последовательностью конституент, заключенной в скобки. Так, например, правило, определяющее понятие целого

целое::=положительное:отрицательное

говорит о том, что целое может быть или положительным, или отрицательным, причем и для того, и для другого имеются свои определяющие правила. Обозначение понятия в синтаксических правилах выглядит либо как слово, либо как группа слов, отделенных друг от друга дефисами. Восклицательный знак, сопровождающий конституенту, означает, что данная конституента может быть повторена один или более раз. Вопросительный знак показывает, что предшествующая ему конституента может отсутствовать. Для обозначения серий однотипных объектов используется конституента вида (а/в), являющаяся сокращением последовательности **а в а ... в а**, где а и в - произвольные конституенты. В качестве примера можно привести правило:

описание-переменных::=
    переменные (описание-переменной/;)

которое означает, что описание переменных начинается с терминального символа переменные, после которого следуют описания переменных, каждое из которых отделяется от предыдущего точкой с запятой.

Для некоторых синтаксических понятий в данном описании приводится несколько правил, причем каждое последующее является либо дополнением, либо расширением предыдущих. Для удобства чтения после имени определяемого понятия в скобках указывается номер раздела, в котором это понятие в очередной раз расширяется. В сводном синтаксисе, описанном в приложении, приводятся номера всех разделов, в которых объясняется указанное понятие. Некоторые понятия снабжаются комментариями, обрамленными процентами (%). Некоторые терминальные символы языка представляются подчеркнутыми словами. Подчеркиванием в данном описании выделяются также терминальные символы языка для отличия от метасимволов с аналогичным изображением (круглые скобки, двоеточие и наклонная черта).

При описании семантики языка предполагается наличие некоторого вычислителя, способного исполнять каждую описываемую конструкцию указанным способом.

По отношению к этому вычислителю все средства языка можно разбить на три группы. При использовании средств первой группы не требуется знания внутренней структуры вычислителя. Средства первой группы образуют замкнутое подмножество языка, которое может быть рекомендовано для начального обучения и использования. Этот подъязык описан в гл. 2-6. Ко второй группе относятся средства открытых подстановок, не имеющие никакого отношения к вычислителю. Открытые подстановки играют значительную роль для программного задания структур данных и процедурных механизмов. Их описание приводится в гл. 7. И, наконец, к третьей группе относятся средства, предполагающие конкретность вычислителя, которым является ЭВМ БЭСМ-6 в некотором операционном окружении. Эти возможности описаны в гл. 8.

2. ЗНАЧЕНИЯ

Значение реализуется в языке словом, представляющим собой последовательность битов, длина которой фиксируется для конкретного вычислителя. Для наглядности последующего изложения описание объектов языка, как правило, сопровождается сопоставлением внешних изображений с их внутренними значениями, при этом размер слова для определенности считается равным 48 битам. Слова могут размещаться в ячейках, обладающих адресами и имеющих соответствующее количество разрядов.

Значения не принадлежат к зафиксированным типам, а трактуются в соответствии с семантикой производимых над ними операций. Существуют различные способы внешнего изображения значений (см. 2.I). Можно именовать группы постоянных или переменных значений с помощью соответствующих описаний (см. 2.2). Для работы с частями значений (полями) в языке имеются специальные структуры (см. 2.I.3) и операции.

2.I. Изображения значений

```
изображение-значения (3.I) ::= число : набор :
    составное : строка
```

2.I.I. Числа

```
число ::=целое : рациональное
целое ::=положительное : отрицательное
положительное ::=цифра!
отрицательное ::=¯положительное
цифра ::=восьмеричная-цифра : 8 : 9
восьмеричная-цифра ::=
    0 : I : 2 : 3 : 4 : 5 : 6 : 7
рациональное ::=
```

р ~?((положительное?.положительное :
положительное) (10целое)?:10целое)

Для получения внутреннего значения, соответствующего положительному целому числу, число представляется в двоичном виде. Отрицательные числа представляются в дополнительном коде. Внутреннее значение рационального является записью числа в двоичном плавающем виде.

2.I.2. Наборы

```
набор ::=шестнадцатеричный : восьмеричный :
    двоичный : текстовый
шестнадцатеричный ::=
    ( ш : шл )(цифра : A : B : C : D : E :F )!
восьмеричный ::=('в : вл )восьмеричная-цифра!
двоичный ::=( д : дл )(0 : I)!
текстовый ::=( т : тл ) "литера!"
```

Для получения внутреннего значения правого (левого) набора к двоичному представлению набора слева (справа) приписывается необходимое число нулей. Служебное слово для левого набора соответствующего типа содержит букву л , по которой и можно отличить левый набор от правого. Шестнадцатеричным элементам A, B, C, D , E, F сопоставляются двоичные представления IOIO, IOII, IIOO, IIOI, IIIO, IIII, соответственно. Двоичные кодировки текстовых элементов (литер) определяются реализацией. Для представления кавычки в текстовом наборе ее необходимо удвоить.

2.I.3. Записи

```
составное (8.7) ::=
    запись%для составного% [(изображение-значения /,)]
запись%для составного% (2.2.I) ::=
    группа-параллельных-полей%для составного%
группа-параллельных-полей (2.2.I) ::= [(поле /,)]
```



```
поле%для составного% ::= слог
слог ::= граница : граница
граница (2.2.I) ::= положительное
```

Запись выделяет поля смежных разрядов в ячейке. Значения границ полей интерпретируются как целые положительные числа и должны быть строго положительными. Число полей в записи должно соответствовать числу изображений, с помощью которых формируется составное значение. Для формирования значения сначала все разряды заполняются нулями. Затем каждое из составляющих значений накладывается на формируемое значение таким образом, что первый разряд совмещается с разрядом, указанным правой границей соответствующего поля. Слово, получившееся в результате таких действий над всеми составляющими значения, и будет представлять собой составное значение.

В языке имеется возможность работы с именами записей. Это достигается при помощи соответствующего описания.

```
описание-записей ::= записи (описание-записи/;)
описание-записи ::=имя%записи% = запись
запись (2.2.I) ::= имя%записи% : изображение-записи
изображение-записи (2.2.I) ::=
    группа-параллельных-полей
имя ::= буква (буква : цифра)!?
буква ::=
    А : Б : В : Г : Д : Е : Ж : З : И : Й : К : Л : М :
    Н : О : П : Р : С : Т : У : Ф : Ц : Ч : Ш : Щ : Ы :
    Ь : Э : Ю : Я : D : F : G : I : J : L : N : Q : R :
    S : U : V : W : Z : X
```

2.I.4. Строки

```
строка ::= "литера!?"
```

Представление последовательности символов, образую-

щих строку, и внутреннее значение изображения строки, являющееся указателем на представление строки, определяются реализацией.

Кавычки в строке изображаются двойными кавычками.

2.I.5. Примеры

Изображения

3328, ш D 00, шл 000000000D ,
в 6400, д IIOI00000000, т ", 0",
[I2:9, 8:I][I3, 0]

обладают одинаковым внутренним значением

0000000000000000000000000000000000IIOI00000000.

Последнее (составное) значение можно изобразить с помощью имени записи. Например:

ИЗ [I3,0]

при условии, что ранее описана соответствующая запись

запись ИЗ = [I2:9, 8:I];

Изображение рационального числа

р 5.0

обладает значением

I0000II0I0I0000000000000000000000000000000000000

2.2. Константы и переменные

Для работы с именами значений имеется возможность путем описания связывать некоторое имя с группой постоянных (при описании констант) или переменных (при описании переменных) значений. При этом под группу значений отводится группа ячеек, последовательных в том смысле, что адрес каждой последующей на единицу больше адреса предыдущей. Для того чтобы дать возможность обращаться к каждому отдельному значению из этой группы или к полю значения, можно при описании связывать имя с соответствующей записью. Для этого следует расширить понятие записи, введенное в 2.I.3.

2.2.I. Записи

Запись определяет структуру группы последовательных ячеек, разбивая эту группу на именованные и/или неименованные поля; каждое из полей, в свою очередь, может быть структурировано аналогичным образом, благодаря чему достигается иерархия структуры.

```
запись ::= имя%записи% : изображение-записи
изображение-записи ::= ⟀ размер-записи ⟀ :
    ⟀ множитель ⟀ х компонента-массива:
    группа-параллельных-полей:
    группа-последовательных-полей
компонента-массива ::= запись
группа-последовательных-полей ::= ⟨ ( поле /,) ⟩
группа-параллельных-полей ::= [( поле /,)]
поле ::= имя%поля% (=( запись : слог ))?:
    изображение-записи : слог
слог ::= граница : граница
размер-записи ::= константа
множитель ::=константа
граница ::= константа
константа (2.2.2) ::= изображение-значения
```

Разбиение, задаваемое записью (которая может являться полем объемлющей записи), можно описать следующим образом.

I) Запись, задаваемая указанием ее размера, определяет структуру для группы последовательных ячеек. Полями такой записи являются ячейки, которые идентифицируются внутри структуры порядковыми номерами от единицы до номера, равного размеру записи. Если такая запись с размером, равным единице, участвует в описании именованного поля, ее можно опускать.

2) Поле, определяемое границами, выделяет группу смежных разрядов в ячейке, включая граничные разряды. Левая и правая границы могут быть указаны в произвольном порядке. Размер такого поля (называемого слогом) полагается равным единице.

3) Запись можно задать в виде массива записей с помощью множителя. Компонента подобного массива идентифи-

цируется номером, пробегающим значения от единицы до величины множителя. Размер такой записи равен произведению размера компоненты на величину множителя.

4) Запись, задаваемая списком полей, заключенным в квадратные скобки, определяет разбиение для каждого поля из списка с одной и той же начальной ячейки разбиения. Среди размеров параллельных полей выбирается максимальный, он и является размером записи.

5) Группа последовательных полей определяет разбиение для каждого из полей таким образом, что начальная ячейка разбиения некоторого поля располагается непосредственно за последней ячейкой разбиения предшествующего поля. Начальные ячейки разбиения всей такой записи и ее первого поля совпадают. Размер записи в данном случае равен сумме размеров всех составляющих полей.

Поля записи могут быть именованы, что позволяет получить доступ к такому полю с помощью селектора записи. Кроме имен в селекторе записи могут появляться номера, идентифицирующие внутренние компоненты поля указанным выше образом.

```
селектор-записи ::=
    ( изображение-значения : имя%поля% )
    (.селектор-записи )?
```

Если запись представлена своим размером, то можно выделить любую ячейку структуры, указав в качестве селектора номер этой ячейки, отсчитываемый от начала разбиения.

2.2.2. Константы

```
описание констант ::=
    константы ( описание-константы /;)
описание-константы (8.5) ::=
    имя%константы% ( : запись )? = ( константа /,)
константа (8.7) ::=изображение-значения :
    имя%константы% (.селектор-записи )?
```


Если размер записи в описании константы равен единице, запись можно опустить. При описании константы под нее отводится группа последовательных ячеек в количестве, равном размеру записи; это число является также и размером константы. Отведенные ячейки заполняются значениями, приведенными в списке, располагаемом в описании константы справа от знака равенства. Количество элементов списка должно быть в точности равно размеру константы. Структура для данной группы значений определяется записью таким образом, что начальная ячейка разбиения записи совмещается с первой из отведенных под константу ячеек. При отсутствии записи в описании константы ее размер считается равным количеству элементов списка значений.

Использующее вхождение имени константы с селектором записи (если он является единицей, его можно опустить) определяет значение следующим образом:

I) Если подзапись, определяемая селектором записи, является слогом, то содержимое слога займет младшие разряды формируемого значения, т.е. первый разряд слога совмещается с первым разрядом значения. Все остальные разряды заполняются нулями.

2) Если подзапись определяет ячейку, то ее содержимое и является определяемым значением.

3) Значение, которое определяется подзаписью, имеющей структуру, равно содержимому начальной ячейки разбиения этой структуры.

2.2.3. Переменные

```
описание-переменных ::=
    переменные ( описание-переменной /;)
описание-переменной (2.3.I) ::=
    имя%переменной% ( : запись )?
переменная (2.3.I) ::=
    имя%переменной% (.селектор-записи)?
```

Как и в п. 2.2.2, первые два правила относятся к определяющим вхождениям, а третье - к использующим. Описание переменной инициирует отведение группы ячеек в количестве, равном размеру переменной. Значения, содержащиеся в этих ячейках, могут меняться в процессе исполнения программы. Задание структуры с помощью записи определяется так же, как и у константы. Значение переменной с селектором записи определяется способом, описанным ранее (см. 2.2.2).

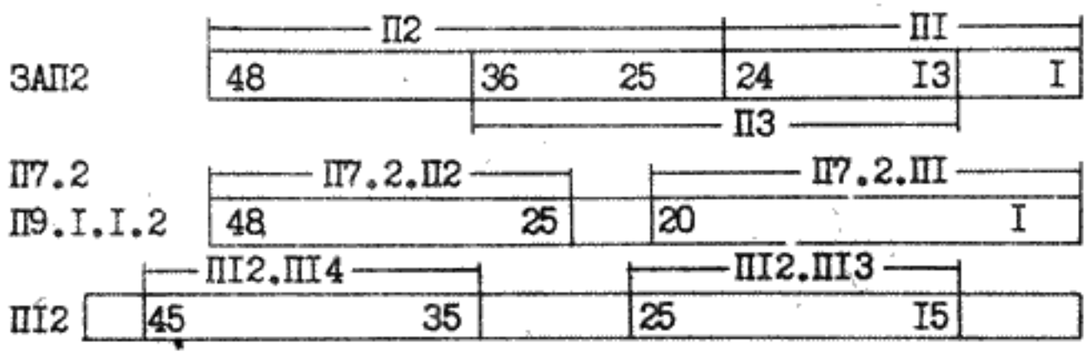
2.2.4. Примеры

Пусть имеется описание двух записей

```
записи ЗАП1=/2/х[П1=1:20, П2=25:48],

    ЗАП2=( [П1=1:24, П2=25:48, П3=13:36],
    [П4=/2/, П5=10:40, П6],
    [(П7=ЗАП1, П8), П9=/3/х/2/х/3/],
    [П10=[10:20], П11=30:40],
    П12=[П13=15:25, П14=35:45] );
```

Размер структуры, определенной описанием записи ЗАП2, равен 23 ячейкам. Ниже в качестве примера приведена разбивка первой, пятой и последней ячеек со всеми возможными селекторами подзаписей. Селекторы, выписанные слева, именуют всю ячейку.



При описании переменной, связанной с этой записью, отведется 23 ячейки:

```
    переменная ПЕР:ЗАП2;
```

При описании константы, имеющей структуру ЗАП2, список

инициализации должен состоять из 23 констант:

```
константа КОН:ЗАП2= ш FAIBC20,
        шл АВО,
        [1:2, 3:5][3, 7],
        р 42.3,
        [20:25, 27:48]  [ш 1В, 3],
        7,
        -17,
        дл 101,
        [1:5, 6:15]  [д 111, в 23],
        1309,
        в 341,
        д 11010000101,
        ш D0001,
        1,
        [1:4, 5:6]  [д 1110, 1],
        в 77,
        р .401,
        401,
        00001,
        ш ABCDEF,
        13,
        ш 10000008,
        [25:35]  [ш 201];
```

При этом будут справедливы, например, следующие соотношения:

```
КОН          =  в  1750336040
КОН.П1       =  в  50336040
КОН.П2       =  в  17
КОН.П3       =  в  175033
КОН.П7.2     =  в  1466000000
КОН.П7.2.П1  =  в  2000000
КОН.П7.2.П2  =  в  14
КОН.П12      =  в  1001000000000
КОН.П12.П13  =  в  2000
КОН.П12.П14  =  в  0
```

2.3. Векторы

2.3.1. Описание и использование векторов

Для того чтобы иметь возможность работать с массивами структур, понятие переменной соответствующим образом расширяется.

```
описание-переменной (3) ::=
    имя%переменной% ( [ константа] )?(: запись )?
переменная (7.2) ::=
    имя%переменной% ( [предложение ] )?
    (.селектор-записи )?
предложение (3) ::= константа : переменная
```

Переменные, описанные как одномерные массивы, будут иногда называться векторами.

При описании вектора отводится группа последовательных ячеек в размере, равном произведению количества элементов вектора (задаваемого константой) на размер элемента (равный размеру записи).

При обращении к вектору номер элемента поставляется предложением, а поле элемента указывается селектором. Результатом такого обращения является значение, выбранное из указанного поля.

2.3.2. Примеры

Пусть имеется описание векторов:

```
переменные ВЕКТ1[3];
           ВЕКТ2[2] : [ДЛ=16:48, АДР=1:15];
           ВЕКТ3[2] : ЗАП1;
```

При таком описании в памяти отведется три группы последовательных ячеек. Ниже показано разбиение первого элемента третьего вектора с указанием всех возможностей доступа.

	ВЕКТ 3[1].1.П2		ВЕКТ 3[1].1.П1
ВЕКТ 3[1]			
ВЕКТ 3[1].1	48 25		20 1
	ВЕКТ 3[1].2.П2		ВЕКТ 3[1].2.П1
ВЕКТ 3[1].2	48 25		20 1

3. ОПЕРАЦИИ

Для обработки и изменения значений и слогов значений в языке имеется некоторое множество операций. Считается, что результат применения операции размещается в специальной ячейке, называемой сумматором. Перед началом исполнения программы содержимое сумматора (далее просто сумматор) считается неопределенным. Значения для операций поставляются первичными, которые, в свою очередь, могут обладать внутренней структурой. Последовательность первичных и операций, построенная по определенным правилам, образует предложение.

```
предложение (4.3) ::= ( формула /;)
формула ::= первичное ?( операция первичное:
    выделение-слога : изменение-слога :
    изменение-слога-с-засылкой : засылка )!?
первичное (4) ::= константа : адрес : ( предложение )
адрес (8.1) ::= переменная
```

Исполнение формулы заключается в последовательном слева направо применении бинарных операций. Результирующее значение (результат), полученное применением какой-либо операции, остается на сумматоре. Результат последней исполненной операции считается результатом исполнения формулы. Результатом исполнения предложения является результат последней исполненной формулы. Порядок вычисления операций формулы можно задать соответствующей расстановкой круглых скобок. В тех случаях, ког-

да вычисление операнда имеет побочный эффект, следует с повышенным вниманием отнестись к остальным возможным вхождениям изменяемой величины в формулу.

Результат формулы может использоваться для присвоения начальных значений при описании переменных. С учетом возможности такой инициализации описание переменной выглядит следующим образом:

описание-переменной (8.I) ::=
    имя%переменной% ([константа])?
    (: запись)?(=(формула /,))?

При этом количество формул в списке инициализаций не должно превышать размера переменной.

Значением левого операнда, как правило, является содержимое сумматора. К исключению из этого правила следует отнести первую операцию формулы, начинающейся с первичного. В этом случае левым операндом является данное первичное, причем порядок вычисления операндов не определен. Последнее важно иметь в виду, когда вычисление операнда имеет побочный эффект.

В том случае, когда формула состоит из одного первичного, значение последнего является результатом исполнения формулы.

3.I. Операции над значениями

операция (8.3) ::=
    + : - : x : дел : сл : вч : умн :
    делр : < : ⩽ : > : ⩾ : м : мр :
    б : бр : = : ≠ : ∨ : ∧

Значения операндов трактуются языком в соответствии с операциями. Внутренние представления соответствующих классов значений приведены в разделе 2.I.I. Результат операции гарантируется при условии, что указанная трактовка операндов и результата возможна. Например, если указано, что операнды и результат положительны, то это

означает следующее:

I) старшие восемь разрядов значений операндов равны нулю;

2) младшие 40 разрядов представляют собой двоичное представление целого положительного числа;

3) результат операции не превышает $2^{40}-1$.

При выполнении этих условий результат в соответствующем внутреннем представлении сохраняется на сумматоре, в противном случае результат операции не определен.

Добавляется еще один класс значений, называемых логическими. Ниже приведены изображения этих значений.

изображение-значения ::= истина : ложь

В приводимой ниже таблице определяется семантика операций и указывается интерпретация значений операндов и результата для каждой операции.

интерпретация значений		операция	результат
операндов	результата		
цел	цел	+	сумма
		-	разность
полож	полож	x	произведение
		дел	целая часть
рац	рац	сл	сумма
		вч	разность
		умн	произведение
		делр	частное
цел	лог	< ⩽ > ⩾	результат имеет значение истина, если отношение выполняется, и ложь -в противном случае
рац	лог	м (меньше) мр (меньше или равно) б (больше) бр (больше или равно)	

произв	лог	= ≠	
произв	произв	∨	поразрядная логическая сумма
		∧	поразрядное логическое произведение

Операции логического сложения и умножения над логическими значениями выполняются в смысле Булевской алгебры, согласно следующей таблице.

значение левого операнда	значение правого операнда	результат	
		∨	∧
истина	истина	истина	истина
истина	ложь	истина	ложь
ложь	истина	истина	ложь
ложь	ложь	ложь	ложь

3.2. Операции над слогами

Синтаксис, описывающий операции над слогами, содержит понятие записи. В данном случае имеется в виду частный случай записи с размером, равным единице. Именно с помощью таких записей осуществляется работа с частями слов.

3.2.1. Выделение слога

```
выделение-слога ::= * запись
```

Исполнение описываемой операции состоит в следующем. Из значения левого операнда выделяется группа разрядов, определяемая записью. Содержимое выделенного слога занимает младшие разряды формируемого значения. Остальные разряды заполняются нулями. Получившееся значение размещается на сумматоре и является результатом операции.

3.2.2. Изменение слога

```
изменение-слога ::= =>запись * первичное
изменение-слога-с-засылкой ::= ->запись * адрес
```

Для изменения слога выполняются следующие действия. Сначала в значении, поставляемом первичным, обнуляются разряды, соответствующие выделенному слогу. Затем значение левого операнда размещается в выделенном слоге ранее описанным способом, т.е. первый разряд значения левого операнда совмещается с правым граничным разрядом выделенного слога. Получившееся значение является результатом операции изменения слога. В случае, если это изменение сопровождается засылкой (->), выполняется еще один шаг: засылка сформированного значения по соответствующему адресу. Для переменной этим адресом является начальная ячейка разбиения подзаписи, определяемая селектором записи переменной.

3.3. Засылки

```
засылка ::= простая-засылка : совмещенная-засылка
простая-засылка ::= ->адрес
совмещенная-засылка ::= операция -> адрес :
   * (-> запись * адрес : => запись * первичное ):
    операция -> запись * адрес
```

Простая засылка позволяет изменить значение содержимого ячейки с указанным адресом. Исполнение совмещенной засылки можно описать следующим образом (порядок вариантов соответствует синтаксису)

1) Сначала применяется операция, причем значение правого операнда поставляется первичным, находящимся в позиции получателя (адрес). Затем результат операции размещается по адресу, указанному получателем.

2) Во-первых, происходит выделение слога (определяемого записью) из значения левого операнда. Далее, по

ранее определенным правилам (см. 3.2.2), производится операция изменения слога.

3) Вначале поставляется значение правого операнда для операции. Для этого производится выделение слога, определяемого записью из значения, которое поставляется первичным, находящимся в позиции получателя. Затем исполняется операция, и получившийся результат является значением левого операнда для операции описываемой совмещенной засылки.

Суммируя вышеизложенное, нужно подчеркнуть, что основным свойством засылки любого типа является частичное или полное обновление содержимого некоторой ячейки, за исключением случая совмещения операции с изменением слога без засылки. Важно отметить, что результатом исполнения засылки является полное слово, размещенное в обновленной ячейке.

3.4. Примеры

Ниже приводится ряд формул и результаты их исполнения. Изображения значений результатов даются в соответствии с синтаксическими правилами языка (см. изображение значения). Операнды, встречающиеся в формулах, представлены либо изображениями значений, либо переменными и константами из предшествующих примеров (см. 2.2.4). Порядок расположения формул в примерах существенен в связи с имеющимися в них засылками.

Формула	Значение
3x2-> ПЕР^1	0
1+КОН.П2 дел 4-> ПЕР	в 4
3<ПЕР ∧ (р 2 б р 1)	истина
КОН.П4.2-2-> ПЕР.П2*[1:3]∧2=0	истина
истина +1∨(в 6670-> ПЕР.ПП0)	не определено
р 5.0=5	ложь
р 401x5	не определено
ПЕР.П2-> [6:10]*ПЕР.П1	в 3500001644
в 16∧-> ПЕР	в 4
в 11600 *⇒ [1:9] * (ПЕР+ в 3001 2)	в 30600
д 111∧-> [1:9]* ПЕР.ПП0	в 6677

4. СТРУКТУРЫ УПРАВЛЕНИЯ

Структуры управления в языке позволяют управлять последовательностью вычислений (см. 4.1) и задавать необходимые повторения действий в ходе вычислительного процесса (см. 4.2). Исполнение структур управления может завершиться нормально, в соответствии с описанной семантикой конструкции. Наряду с этим, язык дает возможность принудительного завершения исполнения структуры (см. 4.3). Структуры управления расширяют понятие первичного.

первичное (4.3) ::= альтернативное : циклическое

4.1. Альтернативы

```
альтернативное ::=
    условное : выбирающее : перебирающее
условное ::=
    если условие
    то предложение
    ( иначе предложение)?
    все
выбирающее ::=
    выбор предложение
    из (предложение/,)
    ( иначе предложение)?
    все
перебирающее ::=
    перебор предложение
    из ((предложение/,) то предложение /,)
```

(иначе предложение)?
все
условие (8.I) ::=предложение (ист : неист)?

При последующем чтении следует учесть два факта. Первый касается равноправности использования разделителя то и двоеточия в перебирающем. Второй касается описания семантики, в котором идентификация предложений происходит с помощью предшествующего разделителя, например, иначе-предложение.

I) Исполнение условного первичного происходит в следующем порядке. Сначала исполняется предложение из условия, причем результат трактуется как логическое значение. Если предложение сопровождается символом ист (который можно опускать), то полученное логическое значение будет результатом всего условия. Если же предложение сопровождается символом неист , то к полученному результату применяется операция логического отрицания и полученное после этого значение считается результирующим значением условия. Если это значение есть истина , выполняется то-предложение, результат которого является в этом случае результатом исполнения условного. В том случае, когда условие вырабатывает ложь , результат условного получается после исполнения иначе-предложения (при его наличии), либо (при его отсутствии) результат условного не определен.

2) Исполнение выбирающего первичного состоит из двух этапов. Сначала в результате исполнения выбор-предложения получается значение, трактуемое как целое число и означающее номер нужной альтернативы в списке. Если список альтернатив содержит предложение с таким номером, то оно и подлежит исполнению, а его результат является результатом исполнения всей конструкции. Альтернативные предложения нумеруются, начиная с единицы. Если вычисленный номер меньше единицы или больше числа предложений в списке, то исполняется иначе-предложение (при его наличии), в противном случае результат выбирающего первичного не определен.

3) Перебирающее первичное составлено из перебор-предложения, вычисляющего ключ, и списка пар, левая часть которых состоит из набора предложений, предназначенных для сопоставления с ключом. Исполнение перебирающего первичного состоит из нескольких шагов. Сначала вычисляется ключ. Затем, начиная с первой пары, происходит следующий процесс. Вычисляются предложения из левой части (до разделителя то) по очереди до тех пор, пока либо результат вычисленного предложения совпадет с ключом, либо будут исчерпаны все предложения. В случае совпадения вычисляется то-предложение, и его результат является результатом исполнения первичного. В случае исчерпания предложений из левой части испытываемой пары процесс повторяется для следующей пары. Если список пар исчерпан, то выполняется иначе-предложение, и его результат в этом случае является результатом описываемой конструкции. В том случае, когда список пар исчерпан, а иначе-предложения нет, значение первичного не определено.

4.2. Циклы

циклическое (8.I) :=
пока условие
цикл предложение
(иначе предложение)?
все :
повторять предложение
(когда условие
(иначе предложение)?)?
все

I) Исполнение пока-цикла заключается в многократном вычислении условия и цикл-предложения. Эта последовательность действий прерывается в тот момент, когда условие выработает в качестве результата ложь . Далее исполняется иначе-предложение (в случае его наличия), которое поставляет результат исполнения цикла.

2) Исполнение повторять-цикла отличается от исполнения пока-цикла только двумя деталями. Во-первых, при отсутствии условия повторение цикла может прерваться только в случае принудительного завершения (см. 4.3). Во-вторых, изменен порядок исполнения повторяющихся действий, описанных для пока-цикла, т.е. тело цикла (повторять-предложение) исполнится хотя бы раз в любом случае.

4.3. Завершения

В языке отсутствует возможность для произвольных передач управления. Разрешается нарушать порядок исполнения структуры управления только путем завершения исполнения структуры на любом уровне вложенности. Синтаксис в связи с этим расширяется следующим образом.

```
предложение (8.6) ::= (формула завершитель ?/;)
завершитель (5.2) ::=
    завершить имя%уровня%?( при условие все )?
первичное (5.I) ::=
    ( : имя%уровня% : )?(альтернативное : циклическое)
```

Порядок получения результата исполнения структур уже описан ранее (см. 4.I и 4.2), но здесь стоит обратить внимание на следующую фразу:"Результатом исполнения предложения является результат последней исполненной формулы" (см. 3). При отсутствии конструкций завершения последней формулой всегда является текстуально последняя формула. Конструкция завершения позволяет прекратить вычисления в любом месте предложения и выйти с полученным результатом на указанный уровень. Другими словами, завершается исполнение соответствующей именованной структуры. В том случае, когда имя завершаемого уровня не указано, происходит завершение исполнения непосредственно объемлющей структуры.

Кроме безусловного завершения, возможно условное завершение. Завершение исполнения структуры происходит,

если результат условия есть истина . При этом исполнение условия не влияет на результат формулы.

4.4. Примеры

Ниже дается таблица, аналогичная приводимой ранее (см. 3.4). В ней содержится ряд предложений вместе с результатами их исполнения.

предложение	результат исполнения
если 3->ПЕР>0 то I иначе 0 все	в I
выбор ПЕР из 0 ,I ,2 все	в 2
0->ПЕР; I->ПЕР.П5; пока I+->ПЕР<I0 цикл 2х->ПЕР.П5 все	не определено
0->ПЕР; I->ПЕР.П5; повторять 2х->ПЕР.П5 когда I+->ПЕР<3 иначе ПЕР.П5 все	в 20
перебор ПЕР из если ПЕР>0 то 0 иначе I все	

: в I6→ПЕР.П5 ,4 : если ПЕР.П5>0 то I→ПЕР все все	в I
пока I+→ПЕР<КОН.П3 цикл завершить при ПЕР=20 все иначе ПЕР все	не определено
0→ПЕР; повторять I+→ПЕР; завершить при ПЕР=I0 все все	в I2
если выбор ПЕР из ш 50FVКОН.ПII , в I00I иначе 0 все →ПЕР.П7=КОН.ПII то ш 7FF иначе ш IF все →ПЕР	в 3777
перебор ПЕР>КОН.ПII из ПЕР.ПIV ш FFFF :I ,ПЕР.ПI∧ ш FFFF :0 иначе 2 все →ПЕР.ПII	не определено
0→ПЕР; :MI: пока ПЕР<I00 цикл 0→ПЕР.П5;	

пока ПЕР.П5≠I0 цикл I+→ПЕР.П5; завершить MI при ПЕР+ПЕР.П5=80 все иначе ПЕР.П5 все +→ПЕР иначе ПЕР все	в I2

5. БЛОКИ

блок ::= начало описания вход предложение конец

Одним из важнейших средств структурирования программы являются блоки. С их помощью вводятся также новые уровни локализации объектов программы.

Описание объекта задает область доступности для имени объекта. Эта область состоит из данного блока, из которого удалены все вложенные блоки, содержащие описание одноименных объектов. Это правило имеет исключение для переменных в некоторых случаях (см. 5.2).

Недопустимы описания одноименных объектов на одном блочном уровне. Время существования значений переменных зависит от местоположения блока. В зависимости от контекста блоки называются статическими (см. 5.I), динамическими (см. 5.2) или модульными (см. 5.3). Исполнение любого типа блока заключается в инициализации переменных и исполнении вход-предложения, результат которого является результатом исполнения блока. Описания представляют собой последовательность описаний различных объектов. Ранее были приведены правила для описания записей, констант и переменных. По мере появления будут приводиться описания и других объектов. Описания обязательны для всех имен, кроме имен уровней структур. Поиск имени уровня (см. 4.3) начинается с самой внутренней из структур, объемлющих использующее вхождение

искомого имени. Если эта структура не именована или ее имя не совпадает с искомым, то происходит переход на следующий уровень и т.д. При переходе на уровень динамического или модульного блока поиск прекращается, причем эта попытка завершения недоступного или вовсе несуществующего уровня квалифицируется как ошибка.

5.I. Статические блоки

```
программа ::= ( : имя%уровня% : )? блок
первичное (5.2) ::= ( : имя%уровня% : )? блок
```

Как видно из синтаксиса, программа представляет собой блок, возможно, с именем уровня. Блок может также занимать позицию первичного. Блоки, занимающие указанные синтаксические позиции, называются статическими. Все, что изложено выше о блоках вообще, верно для статических блоков без исключения. Описание переменной на уровне статического блока инициирует отведение памяти для значений переменной (см. 2.2). В момент завершения исполнения блока эти значения перестают существовать.

5.2. Процедуры

```
описание-процедур (5.3) ::=
    процедуры (описание-процедуры /;)
описание-процедуры (8.I) ::= имя%процедуры%
    ( ( (формальный-параметр/,) ) )?= формула
формальный-параметр ::= имя%переменной% ( : запись)?
первичное (5.3) ::= вызов-процедуры
вызов-процедуры ::=
    имя%процедуры% ( ( (предложение/,) ) )?
завершитель ::= возврат ( при условие все )?
```

Действия, задаваемые описанием процедуры, определяются телом процедуры, которое является динамическим блоком и конструируется следующим образом. Раздел описаний этого блока состоит из формальных параметров, а

вход-предложение задается формулой. Область доступности для переменных, описанных внутри динамического блока, сужается путем удаления из тела процедуры всех вложенных динамических блоков. Формальный параметр можно рассматривать как переменную с размером записи, равным единице. Исполнение процедурного блока инициируется путем вызова процедуры, появляющегося в позиции первичного. Это исполнение заключается в следующем.

I) Отводится память под формальные параметры и под переменные, описанные внутри области процедурного блока, из которой удалены все вложенные динамические блоки.

2) Исполняется по очереди каждый фактический параметр, и результат исполнения становится значением соответствующего формального параметра.

3) Порождается процесс, исполнение которого определяется процедурным блоком. Завершение исполнения может произойти либо естественным образом, либо с помощью возврата, который играет ту же роль, что и завершитель структур управления и статического блока. Т.е., если формула кончается символом возврат , то завершается исполнение самого внутреннего из объемлющих динамических блоков.

4) Результат исполнения процедурного блока является результатом вызова процедуры. На этом исполнение соответствующего первичного кончается.

Время существования переменных совпадает со временем исполнения процесса, порожденного вызовом процедуры.

Ясно, что исполнение процесса, порожденного вызовом процедуры, может прерываться для исполнения новых процессов, порождаемых с помощью новых вызовов той же самой (рекурсивное обращение) или каких-нибудь других процедур. Таким образом, в отличие от статического блока, процедуре может соответствовать в один и тот же момент исполнения программы несколько процессов со своими комплектами переменных.

5.3. Модули

описание-модулей ::= модули (описание-модуля/;)
описание-модуля (8.5) ::=
    имя%модуля%.(((формальный-параметр/,)))?= блок
первичное (7.I) ::= вызов-модуля
вызов-модуля ::= имя%модуля% (((предложение/,)))?

Принципиальной особенностью модульного блока является то, что описанные в нем объекты не перестают существовать при завершении исполнения модульного блока. Они будут существовать во все время исполнения самого внутреннего из всех объемлющих немодульных блоков. Этот блок (статический или динамический) будет называться ниже областью существования для объектов, описанных в модуле. Таким образом, модульный блок влияет только на локализацию имен, но не на время существования объектов, которые им соответствуют. Для некоторых объектов модуля (например, процедур) существует возможность расширить область доступности в рамках области существования. Для этого вводится понятие виртуальных описаний, которые позволяют обращаться к объектам модуля вне модуля.

описание-процедур ::=
    общие ? процедуры (описание-процедуры/;):
    виртуальные процедуры
    (описание-виртуальной-процедуры/;)
описание-виртуальной-процедуры (8.I) ::=
    имя%процедуры% (* анкета%виртуальной процедуры%)?
анкета ::=[(раздел-анкеты/,)]
раздел-анкеты%виртуальной процедуры% ::=
    чиспар = константа

С помощью виртуальных описаний часть имен объектов, описанных внутри модуля, становтся доступной для использования вне модуля. Для других имен модульная локализация служит защитой от внешнего использования.

Доступность имени модуля аналогична доступности имени переменной. Но модуль нельзя вызывать внутри самого этого модуля. Исполнение вызова модуля аналогично исполнению вызова процедуры за исключением того, что не может существовать более одного процесса, порожденного вызовом одного и того же модуля в один и тот же момент исполнения программы.

Понятие анкеты, появившееся в этом пункте, будет использоваться и далее для довольно разнообразных целей. Поэтому имеет смысл перечислить некоторые особенности этого понятия.

Во-первых, наличие раздела анкеты не всегда обязательно, и для этих случаев определяется правило умолчания.

Во-вторых, ни один раздел анкеты не должен повторяться дважды в одной и той же анкете.

В-третьих, анкеты общего и виртуального описаний одного и того же объекта не должны противоречить друг другу.

5.4. Примеры

5.4.I. Иллюстрация определения областей доступности

```
начало
   переменные j;i;
   виртуальная процедура ПРОЦ *
            [чиспар =I];
   модуль М=
    начало переменная i;
      процедура П (X)=
         начало переменная j;
               процедура П2=
                начало
                  переменная i;
                вход 0→i;
                конец ;
          вход
            П2;0→j;
```

```
            X->i;
          конец ;
        общая процедура ПРОЦ (X)=
          начало переменная j;
          вход 0->j->i;
            ПI (X);
          конец
      вход
      конец
вход 0->i;
      ПРОЦ (j);
      начало переменная i;
      вход ПРОЦ (j);I->i;
      конец ;
      I->i;
      начало переменная i;
      вход 0->i;
      конец ;
      ПРОЦ (i);
конец ;
```

5.4.2. Наибольший общий делитель

```
процедура НОД (L,N)=
    начало
    вход
       если  L < N
       то НОД (N,L)
       иначе если N≠0
              то НОД (N,L-(L дел N x N))
              иначе L
              все
       все
    конец ;
```

5.4.3. Модуль работы со стеком

```
модуль РАБОТА СО СТЕКОМ =
      начало
         переменные СТЕК [100];
```



```
                  УКСТЕКА;
            общие процедуры ЗАПИСЬ В СТЕК (X) =
                  (X-> СТЕК [УКСТЕКА];
                   I+-> УКСТЕКА);
                ЧТЕНИЕ ИЗ СТЕКА =
                   СТЕК [УКСТЕКА-I-> УКСТЕКА];
                СТЕК ПУСТ =
                   УКСТЕКА = I;
вход
I-> УКСТЕКА
конец ;
```

5.4.4. Модуль работы с таблицей идентификаторов.

```
модуль РАБОТА С ТИ =
   начало
      переменные ТИ [1000]:[ИМЯ=13:48,
                            СЛЕД ЭЛЕМ ЦЕПОЧКИ=1:12];
                 ОГЛАВЛЕНИЕ [100];
                 УК ТИ; УК ОГЛ;
процедура ФУНКЦИЯ РАССТАНОВКИ (X:[П1=1:8,П2=9:16,
                                  П3=17:24,П4=25:32])=
X.П1+X.П2+X.П3+X.П4-(X.П1+X.П2+X.П3+X.П4 дел 100x100);
   общая процедура ПОИСК В ТИ (ДАННОЕ ИМЯ)=
      начало переменные УКОГЛ;
                        ТЕК ЭЛЕМ ЦЕПОЧКИ;
      вход
           если ОГЛАВЛЕНИЕ[ФУНКЦИЯ РАССТАНОВКИ
            (ДАННОЕ ИМЯ)-> УКОГЛ]-> ТЕК ЭЛЕМ ЦЕПОЧКИ≠0
            то повторять
                 если ТИ [ТЕК ЭЛЕМ ЦЕПОЧКИ].ИМЯ=
                                 ДАННОЕ ИМЯ
                 то ТЕК ЭЛЕМ ЦЕПОЧКИ; возврат
                 все
               когда ТИ [ ТЕК ЭЛЕМ ЦЕПОЧКИ ].СЛЕД
                           ЭЛЕМ ЦЕПОЧКИ -> ТЕК ЭЛЕМ
           все;
      все ;
       ОГЛАВЛЕНИЕ[УК ОГЛ]-> ТИ [I+-> УКТИ-> ОГЛАВЛЕНИЕ
```

```
                [УК ОГЛ]]. СЛЕД ЭЛЕМ ЦЕПОЧКИ;
      ДАННОЕ ИМЯ-> ТИ [УК ТИ].ИМЯ;УК ТИ;
    конец ;
вход
   0-> УК ТИ-> УК ОГЛ;
   пока I+-> УК ОГЛ<100
   цикл 0-> ОГЛАВЛЕНИЕ [УК ОГЛ]
   все
конец
```

6. ПРАКТИЧЕСКИЙ ПРИМЕР ИСПОЛЬЗОВАНИЯ ЯЗЫКА

В главах 2-5 дано описание подмножества языка, достаточного для программирования почти всего класса алгоритмов, реализуемых средствами полного языка. Приводимый в этой главе пример иллюстрирует некоторые возможности, предоставляемые языком. Следует иметь в виду, что с помощью средств, описанных в последующих главах, можно добиться существенного повышения эффективности приведенного алгоритма.

Пример представляет собой реализацию на языке ЯРМО известного алгоритма динамического распределения памяти, основанного на системе " близнецов".

6.I. Описание алгоритма

Система близнецов обеспечивает резервирование и освобождение блоков памяти по запросу.

Применение этого алгоритма ведет к "потере" одного бита в каждом выделяемом блоке и требует, чтобы все эти блоки имели размер (в словах) I,2,4,8,I6 и т.д.

Система близнецов задается следующим образом. Весь объем распределяемого пространства состоит из $2^м$слов. Все свободные блоки связываются в списки отдельно для каждого размера $2^к$, $0<к<м$. Первоначально свободным

является весь блок размера $2^м$. Далее, когда требуется блок из $2^к$слов, а свободных блоков такого размера нет, расщепляется на две равные части блок большего размера; в конце концов появится блок размера $2^к$. Когда один блок расщепляется на два, эти два блока называются близнецами. Позднее , когда оба близнеца освобождаются, они опять объединяются в один блок: таким образом, процесс может продолжаться до тех пор, пока не исчерпается все пространство.

Ключевым фактом, определяющим практическую ценность этого метода, является то обстоятельство, что если известен адрес блока X (т.е. адрес первого слова в блоке) и если известен также размер этого блока , то известен и адрес его близнеца, который вычисляется следующим образом:

$$\text{БЛИЗНЕЦ}\ (X,к)=\begin{cases} X+2^к, & \text{если } X \bmod 2^{к+1}=0; \\ X-2^к, & \text{если } X \bmod 2^{к+1}=2^к. \end{cases}$$

Система близнецов использует одноразрядное поле признака в каждом блоке:

$$\text{ПРИЗНАК}\ (P)=\begin{cases} 0, & \text{если блок с адресом P свободен} \\ I, & \text{если блок с адресом P занят} \end{cases}$$

Кроме поля ПРИЗНАК, которое имеется во всех блоках, в свободных блоках есть еще два поля связи, которые образуют обычный список с двумя связями: вперед и назад; там же имеется поле, в котором указывается значение к, если размер блока равен $2^к$.

Головы списков свободной памяти с размерами блоков I,2,4,..., $2^м$составляют таблицу из м+I элемента. В них содержатся два указателя: связь с концом списка и связь с началом списка.

Как уже отмечалось выше, первоначально в системе есть единственный свободный блок длины $2^м$, а списки для свободных блоков размера 2 при к<м-пусты.

Ниже на языке ЯРМО приводится описание модуля, который формирует такую систему близнецов и обеспечивает ее функционирование.

6.2. Модуль работы с системой близнецов

Система близнецов, задаваемая описанием модуля СИСТЕМА БЛИЗНЕЦОВ, распределяет память общим объемом в 1024 слова, т.е. максимальный ранг блока, который может быть выделен, равен 10.

Процедуры РЕЗЕРВИРОВАТЬ БЛОК и ОСВОБОДИТЬ БЛОК являются единственными объектами, доступными вне модуля, и именно посредством обращения к этим процедурам осуществляется динамическое распределение памяти в рамках заданной области.

```
модуль  СИСТЕМА БЛИЗНЕЦОВ=
   начало
      константы
         СВОБОДЕН=0;
         ЗАНЯТ   =1;
         ОБЪЕМ РАСПРЕДЕЛЕМОЙ ПАМЯТИ=1024;
         МАКС РАНГ БЛОКА=10;
         РАЗМЕР ТАБЛ ЗАГОЛОВКОВ=МАКС РАНГ БЛОКА;
      переменные
         ПУЛ[ОБЪЕМ РАСПРЕДЕЛЯЕМОЙ ПАМЯТИ +
              РАЗМЕР ТАБЛ ЗАГОЛОВКОВ]:[ПРЕД=1:11,
                                       СЛЕД=12:22,
                                       РАНГ=23:26,
                                       ПРИЗНАК=48:48
                                      ];
процедура  СТЕПЕНЬ(РАНГ)=
   (1;
    повторять
       завершить
       при ¯1+->РАНГ<0
       все x2
    все
   );
процедура  АДР ЗАГ СПИСКА СВОБ(РАНГ)=
   ОБЪЕМ РАСПРЕДЕЛЯЕМОЙ ПАМЯТИ+РАНГ+1;
процедура  ЕСТЬ СВОБОДНЫЙ БЛОК(РАНГ)=
```



```
      ПУЛ[АДР ЗАГ СПИСКА СВОБ(РАНГ)].СЛЕД=
          АДР ЗАГ СПИСКА СВОБ(РАНГ);
   процедура  ИСКЛЮЧИТЬ БЛОК ИЗ СВОБ ПАМЯТИ(РАНГ)=
начало
   переменные
      АДР БЛОКА ЛОК;
      АДР ЗАГ ЛОК;
вход
   ПУЛ[ПУЛ[АДР ЗАГ СПИСКА СВОБ(РАНГ)
   ->АДР ЗАГ ЛОК].ПРЕД
   ->АДР БЛОКА ЛОК].ПРЕД->ПУЛ[АДР ЗАГ ЛОК].ПРЕД;
   АДР ЗАГ ЛОК->ПУЛ[ПУЛ[АДР БЛОКА ЛОК].ПРЕД].СЛЕД;
   ЗАНЯТ->ПУЛ[АДР БЛОКА ЛОК].ПРИЗНАК;
   АДР БЛОКА ЛОК
конец ;
   процедура  ВКЛЮЧИТЬ В СПИСОК СВОБ(АДРЕС,РАНГ)=
начало
   переменные АДР ЗАГ ЛОК;
вход
   АДР ЗАГ СПИСКА СВОБ(РАНГ)->АДР ЗАГ ЛОК
   ->ПУЛ[АДРЕС].СЛЕД;
   ПУЛ[АДР ЗАГ ЛОК].ПРЕД->ПУЛ[АДРЕС].ПРЕД;
   АДРЕС->ПУЛ[ПУЛ[АДР ЗАГ ЛОК].ПРЕД].СЛЕД;
   АДРЕС->ПУЛ[АДР ЗАГ ЛОК].ПРЕД;
   РАНГ->ПУЛ[АДРЕС].РАНГ;
   СВОБОДЕН->ПУЛ[АДРЕС].ПРИЗНАК
конец ;
   процедура  АДРЕС БЛИЗНЕЦА(АДР,РАНГ)=
начало
   переменные  РАЗМ ЛОК;
вход
   СТЕПЕНЬ(РАНГ)->РАЗМ ЛОК;
  если  АДР-(АДР дел (РАЗМ ЛОКx2)x(РАЗМ ЛОКx2))=0
   то     АДР+РАЗМ ЛОК
   иначе  АДР-РАЗМ ЛОК
   все
конец ;
   процедура  БЛИЗНЕЦ СВОБОДЕН(АДРЕС,РАНГ)=
```

```
РАНГ≠МАКС РАНГ БЛОКА
∧(ПУЛ[АДРЕС БЛИЗНЕЦА(АДРЕС,РАНГ)].ПРИЗНАК=СВОБОДЕН)
∧(ПУЛ[АДРЕС БЛИЗНЕЦА(АДРЕС,РАНГ)].РАНГ=РАНГ);
     процедура  ОБЪЕДИНИТЬ С БЛИЗНЕЦОМ(АДРЕС,РАНГ)=
начало
   переменные
      СЛЕД ЛОК;
      ПРЕД ЛОК;
      АДР БЛИЗНЕЦА ЛОК;
вход
   АДРЕС БЛИЗНЕЦА(АДРЕС,РАНГ)→ АДР БЛИЗНЕЦА ЛОК;
   ПУЛ[АДР БЛИЗНЕЦА ЛОК].ПРЕД→ ПРЕД ЛОК
   →ПУЛ[ПУЛ[АДР БЛИЗНЕЦА ЛОК].СЛЕД→ СЛЕД ЛОК].ПРЕД;
   СЛЕД ЛОК→ ПУЛ[ПРЕД ЛОК].СЛЕД;
   если  АДР БЛИЗНЕЦА ЛОК>АДРЕС
   то    АДРЕС
   иначе  АДР БЛИЗНЕЦА ЛОК
   все
конец ;
     общая процедура  ОСВОБОДИТЬ БЛОК(АДРЕС,РАНГ)=
      пока   БЛИЗНЕЦ СВОБОДЕН(АДРЕС РАНГ)
      цикл   ОБЪЕДИНИТЬ С БЛИЗНЕЦОМ(АДРЕС,РАНГ)
             →АДРЕС;
             I+→ РАНГ
      иначе  ВКЛЮЧИТЬ В СПИСОК СВОБ(АДРЕС,РАНГ)
      все ;
    общая процедура   РЕЗЕРВИРОВАТЬ БЛОК(РАНГ)=
начало
   переменные
      РАНГ ЛОК=РАНГ;
      АДР БЛОКА ЛОК;
вход
:М: повторять
      если   ЕСТЬ СВОБОДНЫЙ БЛОК(РАНГ ЛОК)
      то     ИСКЛЮЧИТЬ БЛОК ИЗ СВОБ ПАМЯТИ(РАНГ ЛОК)
         →АДР БЛОКА ЛОК;
         пока   РАНГ ЛОК≠РАНГ
         цикл   ВКЛЮЧИТЬ В СПИСОК СВОБ(АДР БЛОКА ЛОК
                +СТЕПЕНЬ( I+→ РАНГ ЛОК),РАНГ ЛОК)
         все ;
         АДР БЛОКА ЛОК;
         завершить М
      все
   когда   I+→ РАНГ ЛОК<МАКС РАНГ БЛОКА
   иначе ложь
   все
конец ;
   вход
      начало
         переменные
            РАНГ ЛОК=0;
            АДР ЛОК;
         вход
            пока   I+→ РАНГ ЛОК<МАКС РАНГ БЛОКА
            цикл   АДР ЗАГ СПИСКА СВОБ(РАНГ ЛОК)→ АДР ЛОК
                   →ПУЛ[АДР ЛОК].ПРЕД;
                   АДР ЛОК→ ПУЛ[АДР ЛОК].СЛЕД
            все ;
            ВКЛЮЧИТЬ В СПИСОК СВОБ(I,МАКС РАНГ БЛОКА)
      конец
конец
```

Литература

I. Гололобов В.И., Чеблаков Б.Г., Чинин Г.Д. Машинно-ориентированный язык высокого уровня для ЭВМ БЭСМ-6. -В кн.:Развитие программного обеспечения БЭСМ-6. М.,Б.и.,I975,с.50-5I.-В надзаг.:ВЦ АН СССР.

2. Чеблаков Б.Г. Представление структур данных в машинно-ориентированном языке высокого уровня. -В кн.:Труды всесоюзного симпозиума по методам .реализации новых алгоритмических языков. Новосибирск, Б.и., I975,с.I60-I76. -В надзаг.:Сиб. отд-ние АН СССР, ВЦ.

3. Чинин Г.Д. Языковые аспекты реализации больших программных систем. -В кн.:Теория программирования и методы трансляции. Новосибирск,Б.и.,I977.с.6-26 -В надзаг.:Сиб. отд-ние АН СССР, ВЦ.

4. Цанг Ф.Р. Реализация операционной системы на языке высокого уровня. -В кн.:Теория программирования и методы трансляции. Новосибирск,Б.и.,I977, с.27-33. -В надзаг.:Сиб. отд-ние АН СССР, ВЦ.

5. Цанг Ф.Р., Чинин Г.Д. Принципы построения и реализации операционной системы спецпроцессора МВК "Эльбрус-I". М.,Б.и.,I978.(Препринт/ИТМ и ВТ АН СССР;22)

6. Цанг Ф.Р. Операционная система спецпроцессора МВК "Эльбрус-I" совместимая с ОС ДИСПАК. -М.,Б.и., I978.(Препринт/ИТМ и ВТ АН СССР;20)

Оглавление.

В.И.Гололобов, Б.Г.Чеблаков, Г.Д.Чинин

ОПИСАНИЕ ЯЗЫКА ЯРМО
МАШИННО-НЕЗАВИСИМОЕ ЯДРО

П р е п р и н т
247

Технический редактор В.С.Сергеев

Художник-оформитель И.Г.Бархатова

Подписано в печать 29.10.80. МН 06995
Формат бумаги 60x90 1/16 Объем 2.2 п.л.Уч.-изд.л.2.4
Тираж 250 экз. Заказ № 454

Ротапринт ВЦ СО АН СССР, Новосибирск 90