Использование операторного языка для реализации операций, обеспечивающих специализированные исследовательские системы

А. А. Рубан, email: andreygrigorev0@gmail.com
Н. Е. Балакирев, email: balakirev1949@yandex.ru
А. Э. Майдан, email: alexeymaidan2003@gmail.com

Московский авиационный институт
(национальный исследовательский университет)

Аннотация. *В данной работе продемонстрированы преимущества операторного языка, используемого для реализации алгоритмов для прикладной области исследования и распознавания информационного содержания волн, что не исключает возможность применения его и в других предметных областях. Представленный операторный язык оптимизирует использование компьютерных ресурсов, позволяет облегчить разработку требуемых алгоритмов в терминах затрагиваемой предметной области решать и обеспечивает его расширение.*

Ключевые слова: *Функциональный оператор, семантический оператор, предметная область, синонимия, макрообработка*

Введение

В рамках работ по исследованию и распознаванию информационного содержания волн [1] создание программных средств ведется как на языках высокого уровня, так и на ассемблере. Использование макроязыка позволило уйти от трудоемкости написания программ в терминах команд компьютера и разработать специализированный язык операторов. Постепенное расширение набора операторов привело к созданию целой системы инструментов, которые по своему функционалу стали близки к языкам высокого уровня и, кроме того, обеспечили гибкость в именовании и оформлении операторов. Таким образом, появилась возможность конструировать специализированные алгоритмы в терминах предметной области и в то же время определять «смысл» производимых действий на содержательном уровне в терминах этой предметной области.

При разработке языка операторов был выбран язык FASM [1], который обеспечил возможность создавать любой необходимый набор операторов, в том числе и возможность именования операторов на

русском языке. Создание такого рода языка в рамках исследований в области извлечения информационного содержания волн позволяет нам решать следующие задачи:

1. Формализовать специализированные операции работы с данными через набор функциональных операторов
2. Оформить такие специализированные операции, опирающиеся на формализованные операторы, в виде синтаксических конструкций, понятных по содержанию и удобных в применении. Семантическая «нагрузка» на такие синтаксические конструкции может быть переопределена через синонимию таких конструкций.
3. Обеспечить систематизацию используемых системных библиотек и оптимизацию используемых ресурсов при написании программ.
4. Предоставить возможность упрощения представления основных системных директив при оформлении программ (Например, Начало_Программы или же Конец_Программы).
5. Разрешить проблемы совместного использования различных кодировок при оформлении программ.

Решение поставленных задач достигается через:

1. Множество форм представления операторов.
2. Предоставление сервиса для систематизации и оптимизации используемых ресурсов.
3. Упрощение представления основных системных директив при оформлении программ.
4. Обеспечению сервиса по совместному использованию различных кодировок при оформлении программ.

1. Множество форм представления операторов.

Фактически, наличие макроязыка ассемблера позволило нам создать набор операторов, который со временем стал расширяться, что и привело к созданию инструментария в виде языка структурных операторов. Все созданные операторы в таком языке имеют два взаимосвязанные формы:

1. Функциональная форма.
2. Семантическая форма.

Функциональный оператор ориентирован на оптимальную реализацию обобщенного алгоритма и по форме подобен вызову процедуры с параметрами. Функциональный оператор представляет из себя последовательность команд, которая появится после развертывания макроса с учетом входных параметров, также он представляет из себя

последовательность команд, которые обеспечат действия, декларируемые в семантической части с момента их выполнения.

Например, имеется следующая задача:

Необходимо установить отношение между тремя беззнаковыми переменными и перейти с учетом установленного отношения в одну из 13 точек программы для дальнейшей обработки. Значения, между которыми требуется установить отношение, будут находится по адресам AX, BX, CX. А 13 точек перехода будут именоваться P1, P2,…., P13. Тогда под именем UC!ABCJmp мы имеем функциональную форму оператора, которое будет реализовывать это действие через определенную последовательность команд.

При исследовании и распознавании информационного содержания волн часто используется оператор установления отношения между тремя переменными.

Пример обращения к функциональной форме такого оператора для решения поставленной задачи продемонстрирован на рис.1.

```
UC!ABCJmp   AX,BX,CX,P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13,ExiPloxo
```

Рис. 1. Обращение к функциональной форме оператора

При использовании только лишь функциональной формы такого оператора возникают трудности, такие как:

- Слабая наглядность передаваемых параметров оператору при обращении.
- При большом количестве возможных адресов для передачи управления в программе, возникает потребность все время держать в голове или где-то смотреть при каком условии произойдет передача управления по тому или иному адресу.

Семантический оператор реализует наиболее удобную для разработчика программы форму обращения к функциональному оператору. Имя семантического оператора может отражать смысловое содержание предметной области. Обращение к параметрам семантического оператора может содержать не используемые в функциональном операторе комментарии, вводимые при помощи символа «$», дающие информацию о предназначении, используемых при обращении к семантической части оператора, параметров и другую описательную информацию.

Механизм макрообработки позволяет относительно одного функционального оператора иметь множество синонимов

семантических операторов, таким образом привязывая лингвистически функциональный оператор к предметной области.

Близость к естественно – языковому описанию алгоритмов в терминах той предметной области, к которой он относится, облегчает отслеживание смысла обработки данных и по факту является взаимно-однозначным описанием того, что в конечном итоге исполняется в программе. Любые другие описания принятые в технологии производства программного продукта будут лишь некоторой проекцией реального хода обработки информации. Кроме этого, при необходимости разбора содержания проще разбираться в терминах предметной области, данные которой и обрабатываются соответствующей программой. Пример обращения к семантической части оператора для решения той же самой задачи продемонстрирован на рис.2.

```
1  УчитываяОтношенияWПерейти\
2          A?B?C?\
3                  $<\>Real=>                P1\
4                   $</>Real=>               P2\
5                    $<-->Real=>             P3\
6                     $</``>Real=>           P4\
7                      $<_/>Real=>           P5\
8                       $<``\>Real=>         P6\
9                        $<\_>Real=>         P7\
10                        $</\>Real=>        P8\
11                         $</\.>Real=>      P9\
12                          $<./\>Real=>     P10\
13                           $<\/>Real=>     P11\
14                            $<`\/>Real=>   P12\
15                             $<\/`>Real=>  P13\
16                              $bad=>       ПлохойВыход
```

Рис. 2. Обращение к семантической форме того же оператора с вводимыми комментариями

Как видно на рис.2 при обращении к семантической форме оператора установления отношения между тремя беззнаковыми переменными, после указания основного семантического ключа «УчитываяОтношенияWПерейти», по которому идет обращение к семантической форме оператора, указываются три переменные, между которыми требуется установить отношение. В данном семантическом ключе присутствует символ «W», что означает то, что отношение устанавливается между словными (2 байта) переменными. После

семантического ключа идут 13 точек перехода, куда будет передано управление в программе при установлении определенного класса отношений между тремя переменными. Вводимые комментарии при помощи символа «$» позволяют наглядно увидеть при каких исходных значениях будет передано управление по тому или иному адресу. Например, на рис.2 в строке номер 3 после символа «$» указывается описательная информация, которая говорит о том, что передача управления в программе по адресу, указанному после «=>», «P1» произойдет в том случае, если (A>B) & (B>C), например, такой переход произойдет, если, например, (A=9), (B=8), (C=6).

В строке номер 10 на рис.2 находится описательная информация, которая дает понять о том, что передача управления в по адресу, указанному после «=>», «P8» произойдет в том случае, если (A<B) & (B>C) & (A=C). Такой переход произойдёт, если, например, (A=5), (B=8), (C=5). Графическое обозначение введенного комментария продемонстрировано на рис.3.

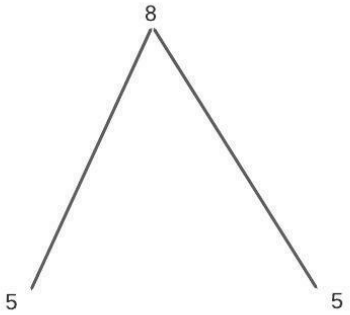

Рис. 3. Графическое обозначение комментария при переходе по адресу P8

В примере обращение к семантической форме оператора на рис.2 использовался символ «\» для переноса частей оператора при обращении к ним на следующую строку. Такой способ позволяет структурировать обращение к оператору и сделать его более наглядным и понятным для восприятия.

2. Использование языка операторов для систематизации и оптимизации используемых ресурсов при написании программ

Предлагаемый язык операторов позволяет производить систематизацию и минимизацию используемых ресурсов при составлении программы, посредством особого набора операторов, представляющих из себя макроопределения, которые позволяют подключать или отключать наборы библиотек операторов, представленных в виде обособленных групп.

Отключение или включение операторов происходит через редактирование Общего Списка Операторов, в котором все операторы поделены на группы и находятся в упорядоченном состоянии. Набор операторов можно упорядочить в любом желаемом порядке (по алфавиту, номерам и т.д.) Общий список операторов предоставляется каждому пользователю языка, и он может быть индивидуальным, в зависимости от потребностей пользователя.

При начале написания программы следует подключить нулевой набор операторов, который предоставляет доступ к функционалу для минимизации используемых ресурсов. Внутри такого набора происходит подключение общего списка операторов, а также подключение списка системных операторов, доступных для редактирования с точки зрения подключения или отключения требуемых операторов. Данный набор операторов позволяет:

1. Подключать или отключать необходимые библиотеки в оперативном режиме без изменений в файле INCALL.inc.
2. Подключать или отключать необходимые системные библиотеки в оперативном режиме без изменений в файле SYSALL.inc.

При помощи оператора, начинающегося с «!», мы имеем возможность при выборе ресурсов, который нам будут нужны при составлении программы, выбрать только те ресурсы, которые нам требуются. В рамках данного оператора после «!» указывается:

- Наименование оператора, которое включает файлы, имеющие данное имя оператора.
- Порядковый номер оператора.
- Символы «+», «-» или «*». Нужный оператор будет подключен, если присутствует знак «+» в строке с его именем, где также указан путь доступа к макроопределению с таким именем. Наличие знака минус «-» означает запрет на подключение данного оператора. Знак «*» означает, что библиотека оператора уже подключена другой строкой с ОПЕРАТОРОМ, входящим в эту же библиотеку.

– Путь к файлу, где находится сам оператор.

Пример использования общего списка операторов для подключения или отключения нужных ресурсов продемонстрирован на рис4.

```
;*****************************************************************************
! УпрятатьРегистры       1+      'C:\FASM\BIBLMCN\PUSHe.inc'
! ВосстановилиРегистры   2*      'C:\FASM\BIBLMCN\PUSHe.inc'
! УпрятатьРегистры       3*      'C:\FASM\BIBLMCN\PUSHe.inc'
! ВостановитьРегистры    4*      'C:\FASM\BIBLMCN\PUSHe.inc'
! CrPUSH                 5*      'C:\FASM\BIBLMCN\PUSHe.inc'
! CrPOP                  6*      'C:\FASM\BIBLMCN\PUSHe.inc'
;*****************************************************************************
! PUSHe                  7-      'C:\FASM\BIBLMCN\PUSHREG\CrPUSHs.inc'
! POPe                   8-      'C:\FASM\BIBLMCN\PUSHREG\CrPUSHs.inc'
! CrPUSHs                9-      'C:\FASM\BIBLMCN\PUSHREG\CrPUSHs.inc'
! CrPOPs                 10-     'C:\FASM\BIBLMCN\PUSHREG\CrPUSHs.inc'
! CrPUSHe                11-     'C:\FASM\BIBLMCN\PUSHREG\CrPUSHs.inc'
! CrPOPe                 12-     'C:\FASM\BIBLMCN\PUSHREG\CrPUSHs.inc'
;*****************************************************************************
```

Рис. 4. Фрагмент списка операторов

На рис.4 продемонстрировано, что оператор с порядковым номером 1 подключен, так как после порядкового номера указан символ «+», а вместе с ним и подключены операторы под номерами 2-6, так как данные операторы находятся в одном файле и при подключении одного – подключаются и остальные, на что указывает символ «*» после порядковых номеров операторов. Так же в данном фрагменте присутствуют операторы под номерами 7-12, которые являются английскими вариантами операторов 1-6, так как данный набор на момент создания фрагмента не использовался, то он был отключен путем установления символа «-» после порядкового номера в строке.

Такая технология помогает существенно минимизировать ресурсы, используемые в программах.

Списку операторов должен предшествовать «нулевой» набор операторов, который обеспечивает работу с такими списками. Такая система работы позволяет, не обращаясь к документации, получить информацию о возможных действиях по управлению подключением библиотек.

Использование только необходимого набора операторов для любой программы экономит память и обеспечивает удобство при разборе и получении листинга вашей программы.

3. Упрощение представления основных системных директив в языке структурных операторов

Одна из главных идей создания такого языка операторов – приближение именования используемых операторов при написании

текста программы к предметной области, для которой пишется текст программы, а, соответственно, и приближение описания программы к более человечному понимаю, что в свою очередь дает программисту возможность выразить интерпретацию алгоритма человеческим языком. Немало важным моментом для воплощения в жизнь данной идеи является упрощение представления основных системных директив, которые задают трафарет программы, или же можно сказать «шапку» программы.

Использования языка FASM позволило нам реализовать системные операторы, направленные на облегчение использование некоторых системных директив, например, начала программы или объявления сегмента команд и сегмента данных. В реализованных операторах, любой параметр, с которым мы обращаемы к шаблону, сопровождается ключевым словом, которому присваивается конкретное значение. Пример трафарета программы продемонстрирован на рис.5.

```
include 'C:\FASM\MZERO.inc'

НАЧАЛО_Программы Форма=PE,Вывод=Console,Имя_Входа=start
СЕКЦИЯ_ДАННЫХ Имя=data,Чтение=+,Запись=+

; Данные пишутся здесь

СЕКЦИЯ_КОМАНД

; Текст программы пишется здесь

СЕГМЕНТ_ИМПОРТА
```

Рис. 5. Трафарет программы

4. Обеспечение совмещения различных кодировок при оформлении программ.

В ходе написания текста программы и исполнения написанных программ, когда используются операторы, которые именуются на русском языке, при выдаче разного рода распечаток на русском языке при работе операторов, а также при задании констант, именуемых на русском языке, наблюдается конфликт между кодировками, которые используются в разных текстовых редакторах, таких как (Notepad ++) и кодировкой, которая используется операционной системой при выдаче текста в терминал. В связи с таким конфликтом кодировок наблюдаются «некорректные» символы при выдаче текста на консоль или же может наблюдаться несогласованность имен макроопределений (операторов) при их макровызове.

Для распечатки русского текста из программы требуется кодировка «OEM 866». Если изменить кодировку файла, допустим, при помощи редактора «Notepad ++» и при этом использовать операторы, именуемые на русском языке, то транслятор ассемблера FASM будет неправильно интерпретировать имена операторов. Также при использовании разных кодировок в «Notepad ++» для разных файлов, которые взаимно используются, тоже произойдет конфликт. Из чего следует, что при использовании в одном проекте разных файлов, все они должны иметь одну кодировку.

Из всего выше сказанного появляется задача перевода «на лету» из кодировки «Windows-1251» в кодировку «OEM 866» для текстового редактора «Блокнот», а также из кодировки «utf-8» в кодировку «OEM 866» для «Notepad ++» для выше описанных случаев.

Существует возможность применения «ручного» кодирования текстов на русском языке при помощи задания русских символов через константы, которые соответствуют кодировке «OEM-866» (при помощи таблицы кодировок). Но при большом объеме текстовой информации, которую нужно выдать в терминал, такой способ является весьма затруднительным.

Для решения данной проблемы были разработаны служебные операторы, которые переводят из кодировки «Windows-1251» или «utf-8» в кодировку «OEM 866» при представлении имен в тексте программы на уровне трансляции, так же задании констант, содержащих текст на русском языке для его распечатки в процессе выполнения программы.

При задании русского текста в кодировке «Windows-1251» достаточно лишь уменьшить коды всех используемых символов на 64 (промежуток от 0C0h до 080h) и будет получена кодировка «OEM 866». А вот для редактора «Notepad ++» при задании кодировки «Windows-1251» текстовый редактор кодирует каждую русскую букву 2-ух байтовым кодом «utf-8», что требует перекодирования из 2-ух байтового кода в однобайтовый код «OEM 866».

Было реализовано 4 служебных оператора. Для текстового редактора «Notepad ++»:

- Оператор «dbr» - для задания текстовых констант.
- «Print!Rfall» - для представления возможности печати во время трансляции.

Для текстового редактора «Блокнот»:

- «dbb» - для задания текстовых констант.
- «Print!Bfall» - для представления возможности печати во время трансляции.

При задании и распечатки текстовых констант (или другое применение), имеются издержки по использованию памяти, так как остаток от 2-х байтового кода приходится заполнять пробелами. Всё это происходит из-за ограничений транслятора FASM, который не позволяет сместить счетчик текущего байта назад.

Для применения реализованных операторов достаточно знать лишь их имя. Например, для задания текстовой константы достаточно лишь написать «dbr ‘Распечатка’». Использование таких операторов упростило технологию создания языка операторов на русском языке и было включено в практикум по курсу «Машинно-ориентированные языки».

Список литературы

1. Рубан, А. А. Методы создания специализированных языковых и процедурных инструментов для оптимальной реализации алгоритмов для прикладных предметных областей/ А. А. Рубан, Н. Е. Балакирев, М. В. Зеленова, М. М. Фадеев, В. С. Родионов // Информатика: проблемы, методы, технологии : Материалы XXI Международной научно-методической конференции (Воронеж, 11-12 февраля 2021 г.). – Воронеж, 2021. – С. 139-146.

2. Flat assembler. Assembly language resources [Электронный ресурс] : официальный сайт. – Режим доступа : https://flatassembler.net