**Разработка общего программного обеспечения**

МАКРОЯЗЫК И МАКРОПРОЦЕССОР

Оператор, работающий в какой-либо системе, часто встречается с необходимостью повторять некоторые последовательности действий много раз? Такая последовательность может, например, состоять из ввода некоторой текстовой последовательности, нажатии определенной последовательности клавиш, выполнении однотипного ряда каких-либо арифметических операций. В подобных случаях часто можно воспользоваться аппаратом макрокоманд.

*Макрокоманды* (часто называемые *макро* или *макрос*) являются однострочными сокращениями для группы команд. Используя макрокоманду, программист по существу определяет одну “команду” для представления некоторой последовательности команд.

Определяя соответствующие макрокоманды, оператор может удобным для себя образом вводить свои собственные средства более высокого уровня, не заботясь о структуре системы. Он может достигнуть краткости и простоты управления системой, не теряя при этом основных преимуществ использования исходной системы, такой, как например язык ассемблера. Крупные макрооперации упрощают пользование, отладку и модификацию программ, и облегчают стандартизацию. Многие разработчик вычислительных машин используют макрокоманды для автоматизации составления “подходящих” операционных систем в процессе, называемом *генерацией системы*

МАКРОКОМАНДЫ

В своей простейшей форме макрокоманда представляет собой сокращение для обозначения последовательности операций

Рассмотрим следующий набор команд, взятый из макроязыка IDE для Borland C++ версии 3.1 (TEMC). Рассмотрим следующую программу, написанную с помощью этих операций

Пример 1

.

.

.

SetPrevPos;

FixScreenPos;

PageScreenUp;

FixCursorPos;

.

.

.

SetPrevPos;

FixScreenPos;

PageScreenUp;

FixCursorPos;

.

.

.

В приведенной программе последовательность команд

SetPrevPos;

FixScreenPos;

PageScreenUp;

FixCursorPos;

встречается дважды.

Аппарат макрокоманд позволяет присвоить этой последовательности имя и использовать это имя вместо нее. Можно также определить некоторый макроязык, позволяющий рассматривать данную конструкцию, как определение и в дальнейшем использовать это определение.

Фактически, макропроцессор представляет собой отдельный языковой процессор со своим собственным языком.

Форматы макроопределений в различных системах может отличаться друг от друга. В данном случае последовательность команд, определяющая макрокоманду имеет следующий формат

MACRO <macro name>

-----------------

-----------------

-----------------

END;

Псевдокоманда MACRO - первая строка определения - определяет следующий за ней идентификатор, как *имя макрокоманды*. Вслед за этой строкой располагается последовательность команд, называемых “телом макроопределения”. Определение заканчивается строкой с псевдокомандой END.

Если макрокоманда определена, то использование имени соответствующей макрокоманды в качестве мнемоники кода в программе эквивалентно использованию соответствующей последовательности команд. Если повторяющейся последовательности команд дать имя “MacPageUp”, то наш пример можно будет переписать следующим образом:

|  |  |
| --- | --- |
| *Исходный текст* | *Расширение исходного текста* |
| MACRO MacPageUp  SetPrevPos;  FixScreenPos;  PageScreenUp;  FixCursorPos;  END;  .  .  .  MacPageUp  .  .  .  MacPageUp  .  .  . | .  SetPrevPos;  FixScreenPos;  PageScreenUp;  FixCursorPos;  .  .  SetPrevPos;  FixScreenPos;  PageScreenUp;  FixCursorPos;  .  .  . |

В данном случае макропроцессор заменяет каждую макрокоманду (макровызов) строками:

SetPrevPos;

FixScreenPos;

PageScreenUp;

FixCursorPos;

Такой процесс замены называется *расширением* макрокоманды. Заметим, что само макроопределение не появляется в расширении исходного текста. Определение сохраняется макропроцессором. Вхождение в исходную программу имени макрокоманды как мнемоники операции называется *макровызовом.*

ОПЕРАТОРЫ МАКРОКОМАНД

Аппарат макрокоманд в том виде, как он был описан до сих пор, позволяет подставлять последовательности команд вместо макровызовов, причем все обращения к макроопределению будут заменены идентичными последовательностями команд. Такой аппарат недостаточно гибок: в макровызове нет средств модифицировать коды, которые его заменяют. Существенное расширение возможностей макросредств достигается добавлением операндов (параметров) макрокоманд.

Рассмотрим следующую программу:

Пример 2:

.

.

.

SetPrevPos;

MoveToMark(1);

CenterFixScreenPos;

.

.

.

SetPrevPos;

MoveToMark(2);

CenterFixScreenPos;

.

.

.

В данном случае последовательности команд очень похожи, но не абсолютно идентичны. В первой последовательности используется операнд “1”, а во втором - операнд “2”. Можно считать, что они выполняют одну и ту же операцию с переменным параметром, или операндом. Такой параметр называют “*операндом макрокоманды”* или *“формальным параметром”,* и он обычно объявляется в той же строке, где и имя макроса. В случае работы с языком макроассемблера, он обычно помечается символом &, что отличает его как символ макроязыка от символов ассемблера. В нашем случае, фирма Борланд не предусмотрела в своем макроязыке работы с макрооперандами, однако можно предположить, что если бы макроопределения в языке TEMC могли бы обрабатывать подобную ситуацию, то формат макроопределения мог бы выглядеть следующим образом:

MACRO <macro name>(<paramlist>)

-----------------

-----------------

-----------------

END;

где <paramlist> это перечисление через запятую всех операндов макроса.

Предыдущая программа в таком случае может быть переписана следующим образом:

|  |  |
| --- | --- |
| *Исходный текст* | *Расширение исходного текста* |
| MACRO MacGoto(labelno)  SetPrevPos;  MoveToMark(labelno);  CenterFixScreenPos;  END;  .  .  .  MacGoto(1)  .  .  .  MacGoto(2)  .  .  . | .  .  .  SetPrevPos;  MoveToMark(1);  CenterFixScreenPos;  .  .  .  SetPrevPos;  MoveToMark(2);  CenterFixScreenPos;  .  .  . |

Следует заметить, что макрокоманда может иметь и более одного операнда. Каждый операнд должен при этом соответствовать формальному параметру в строке определения имени макроса.

Рассмотрим следующий пример:

Пример 3:

.

.

.

ScrollScreenDown;

ScrollScreenLeft(0);

FixCursorPos;

.

.

.

ScrollScreenUp;

ScrollScreenLeft(1);

FixCursorPos;

.

.

.

В данном случае операнды в исходных последовательностях команд различны, как и команды. Эта программа может быть переписана так:

|  |  |
| --- | --- |
| *Исходный текст* | *Расширение исходного текста* |
| MACRO MacMove(cmd,left)  cmd;  ScrollScreenLeft(left);  FixCursorPos;  END;  .  .  .  MacMove(ScrollScreenDown,0)  .  .  .  MacMove(ScrollScreenUp,1) | .  .  ScrollScreenDown;  ScrollScreenLeft(0);  FixCursorPos;  .  .  .  ScrollScreenUp;  ScrollScreenLeft(1);  FixCursorPos;  .  .  . |

Следует отметить, что существует два основных способа задания операндов. Первый способ - *позиционный* показан в вышестоящем примере. Существует также способ указания с применением *ключевых операндов*, который позволяет обращаться к формальным операторам, как по именам так и по позиции. Ссылаясь на формальные операторы в определении MacMove можно также использовать следующий вызов:

MacMove(cmd=ScrollScreenUp,left=1)

УСЛОВНОЕ МАКРОРАСШИРЕНИЕ

Иногда возникают потребности изменять порядок команд макрорасширения внутри макроса.

Рассмотрим следующую программу:

Пример 4:

.

.

.

ScrollScreenLeft(1);

LiteralChar('-');

.

.

.

ScrollScreenDown;

ScrollScreenLeft(2);

LiteralChar('\*');

.

.

.

В этом примере не только параметры, но и количество команд - переменная величина. Эта программа может быть записана следующим образом:

.

.

.

MACRO Mac1(p1,left,chr)

IF p1==1 THEN

ScrollScreenDown;

ENDIF

ScrollScreenLeft(2);

LiteralChar('\*');

END

|  |  |
| --- | --- |
| .  .  .  Mac1(1,2,’-’)  .  .  .  Mac1(0,1,’\*’)  .  .  . | .  .  .  ScrollScreenLeft(1);  LiteralChar('-');  .  .  .  ScrollScreenDown;  ScrollScreenLeft(2);  LiteralChar('\*');  .  .  . |

Комбинация IF...THEN...ELSE является *макрометками* или *символами следования* и не появляются в выходном тексте макропроцессора. В макроязыке также могут быть предусмотрены псевдокоманды *условного* и *безусловного перехода* на псевдо-метку, с которой макропроцессор продолжит обработку текста программы. Точно так же, как и в случае выполнения программы, операторы переходов служат для указания выполнения операторов программы, операторы макро-переходов служат для указания порядка компиляции текста программы. Это дает возможность в процессе расширения получать конкретные варианты последовательностей команд, соответствующие данному случаю применения макрокоманды. Выполнение переходов и проверок внутри выполняемого кода увеличивает его размер и время выполнения, в то время, как проверка и переходы в макросах происходят на стадии компиляции и поэтому не требуют затрат времени при выполнении кода. Эта возможность избирательной выборки нужных частей текста является одним из самых мощных средств в системном программировании.

МАКРОВЫЗОВЫ ВНУТРИ МАКРООПРЕДЕЛЕНИЯ

Поскольку макроопределения являются, по сути, “сокращениями” для последовательности команд, то полезным свойством была бы возможность производить такие “сокращения” внутри самих макроопределений.

Пример 5:

.

.

.

MACRO Mac1

SetPrevPos;

CursorCharRight;

END

.

.

.

MACRO Mac2

Mac1

Mac1

END

.

.

.

Внутри макроопределения Mac2 дважды происходит ссылка на макроопределение Mac1. Это помогло нам уменьшить длину макроопределения Mac2 и сделало его более легким для понимания. Такое использование макросредств приводит к макрорасширениям на нескольких уровнях вложенности, например:

|  |  |  |
| --- | --- | --- |
| *Исходный текст* | *Расширение исходного текста (уровень 1)* | *Расширение исходного текста (уровень 2)* |
| .  .  .  MACRO Mac1  SetPrevPos;  CursorCharRight;  END  MACRO Mac2  Mac1  Mac1  END  .  .  .  Mac2  .  .  . | MACRO Mac2  SetPrevPos;  CursorCharRight;  SetPrevPos;  CursorCharRight;  END  Mac2  .  .  . | SetPrevPos;  CursorCharRight;  SetPrevPos;  CursorCharRight;  .  .  . |

Макровызовы, внутри макроопределений могут включать несколько уровней. Например, команда Mac2 могла бы быть выполнена внутри другого макроопределения. Фактически, такие средства, как макро-переходы дают возможность любое число раз обращаться к любому макроопределению, и даже к самому себе. Такие вызовы называются *рекурсивными.*

МАКРООПРЕДЕЛЕНИЯ В МАКРООПРЕДЕЛЕНИЯХ

Мы рассматривали макрокоманды, как обобщенные сокращения для последовательности команд. Представляется разумным разрешить использование в теле макроопределения любых допустимых синтаксисом предложений, в том числе и другие макроопределения.

Необходимо, однако, понимать, что внутреннее макроопределение не будет определено до тех пор, пока не произойдет вызов внешнего макроса. Это - следствие метода реализации макроопределений. Например, пусть пользователь хочет определить группу макроопределений для обращения к подпрограммам с помощью какой-то стандартизированной вызывающей последовательности. Приведенный ниже пример определяет макрокоманду DEFINE, которая при указании в качестве ее операнда имени подпрограммы определяет соответствующий этому имени макрос. Отдельные генерируемые макроопределения получают имена связанных с ними подпрограмм.

Пример 6:

.

.

.

МАСRO DEFINE sub

.......

.......

MACRO sub(param)

......

......

sub(param)

......

......

END

......

......

END

Пользователь может обратиться к этому макроопределению следующим образом:

DEFINE(cos)

определяя таким образом новое макроопределение с именем cos, к которому впоследствии можно обращаться следующим образом:

cos(х)

и макропроцессор сгенерирует соответствующую последовательность вызова функции.

РЕАЛИЗАЦИЯ

Таким образом, нами был описан вариант реализации макроязыка. Рассмотрим метод реализации макроязыка. В качестве примера возьмем классический язык макроассемблера.

ПОСТАНОВКА ЗАДАЧИ

Любой процессор макрокоманд должен решать следующие четыре основные задачи:

*Распознавать макроопределения.* Процессор макрокоманд должен распознавать макроопределения, выделяемые соответствующими псевдокомандами. В языке макроассемблера этими псевдооператорами являются псевдокоманды MACRO и MEND. Эта задача может быть усложнена тем, что внутри макроопределений могут встречаться также другие макроопределения. Когда макроопределения *вложены,* как было продемонстрировано выше, макропроцессор должен правильно распознавать вложения и сопоставить начало и конец макроса. Весь вложенный текст, включая и другие макроопределения определяет отдельную макрокоманду.

*Запоминать макроопределения.* Процессор должен запомнить определения макрокоманд, которые будут впоследствии использоваться для расширения макровызовов

*Распознавать вызовы.* Необходимо также и распознавать макровызовы, представленные в виде мнемонического кода операции. Это предполагает, что имена макрокоманд обрабатываются на тех же самых основаниях, как и один из кодов операции.

*Выполнять расширение макрокоманд и подстановку фактических параметров.* Вместо формальных параметров макроопределения макропроцессор должен подставить соответствующие операнды макрокоманды. Этот текст, в свою очередь может содержать как макрокоманды так и макроопределения.

Таким образом, макропроцессор должен распознавать и обрабатывать макроопределения и макрокоманды.

Что же касается формальных параметров, то тут нужно принять несколько решений. Необходимо определить - могут ли они встречаться в качестве кода операции, каков синтаксис допустимых параметров. В разных реализациях макроязыков могут встречаться разные варианты методы реализации подобных ситуаций, поэтому можно только дать некоторые разумные варианты, покрывающие большую часть возможных реализаций. Формальные параметры могут встречаться в макроопределении где угодно, в том числе и в команде и в коде операции. Мы хотим, чтобы была обеспечена возможность конкатенации формальных параметров макроопределения с фиксированными символьными строками. В таком случае встает вопрос о некоем разделительном символе, обеспечивающем конкатенацию формальных параметров и заданных пользователем символьных последовательностей.

Например, если из один из параметров должен быть соединен с другим (macro[x,y] = xy), то возможен синтаксис x&y, что означает конкатенацию формального параметра x с формальным параметром y. Этот случай не вызывает больших трудностей. Гораздо сложней обрабатывается случай, когда речь идет о подстановке параметра внутри символьной строки. В таком случае возможным выходом будет конкатенация по умолчанию двух последовательно друг за другом идущих символьных строк, а также преобразование формального параметра, заключенного в скобки к символьной строке. Таким образом, если мы хотим, чтобы в макросе фигурировала строка вида “blablabla[x]xxxxx”, где [x] должно заменяться формальным параметром вполне возможно заменить строку такого вида строкой типа “blablabla”(x)”xxxxx”.

Надо заметить, что множество замечательных идей по реализации подобных макроязыков реализовано в языке REXX, поддерживаемом на системном уровне операционной системой OS/2 компании IBM.

Также для выполнения функций условных переходов должны вычисляться некоторые арифметические выражения (возьмем в пример хотя бы обыкновенных счетчик). Таким образом часто оказывается полезной возможность использования псевдо-переменных времени компиляции внутри макросов.

ДВУПРОСМОТРОВЫЙ АЛГОРИТМ

Начнем с некоторых упрощающих предположений. Будем считать, что наш макропроцессор функционально независим от основного компилятора и его текст будет передаваться этому компилятору. Сначала не разрешим макровызовы и макроопределения внутри макроопределений.

Макропроцессор, как и язык ассемблера, просматривает и обрабатывает строки текста. Но в языке все строки связаны адресацией - одна строка может ссылаться на другую при помощи адреса или имени, которое должно быть “известно” ассемблеру. Более того, адрес присваеваемый каждой отдельной строке зависит от содержимого, количества и адресов предшествующих строк. Если рассматривать макроопределение, как единый объект, то можно сказать, что строки нашего макроопределения не так сильно взаимосвязаны. Макроопределения не могут ссылаться на объекты вовне этого макроопределения. Предположим, что в теле макроопределения есть строка INCR X, причем перед этой командой параметр Х получил значение 10. Макропроцессор не производит синтаксический анализ, а производит простую текстовую подстановку вместо “Х” подставляется “10”.

Наш алгоритм будет выполнять 2 систематических просмотра входного текста. В первый проход будут детерминированы все макроопределения, во второй проход будут открыты все ссылки на макросы. Так же, как и язык ассемблера не может выполнить ссылку на символ до того момента, как он встретит этот символ, язык макрокоманд не может выполнить расширение до тех пор, пока не встретит соответствующее макроопределение. Во время первого просмотра проверяется каждый код операции, макроопределения запоминаются в таблице макроопределений, а копия исходного текста без макроопределений запоминается во внешней памяти, для использования ее при втором проходе. Помимо таблицы макроопределений во время первого прохода будет также таблица имен, во второй проход она будет использоваться для выделения макроопераций и расширения их до текста соответствующего макроопределения.

ДАННЫЕ ДЛЯ ПЕРВОГО ПРОСМОТРА

1. ВХТ - Входной текст

2. ВЫХ1 - Выходная копия текста для использования во второй проход.

3. МДТ - таблица макроопределений, в которой хранятся тела макроопределений

4. МНТ - таблица имен, необходимая для хранения имен макрокоманд, определенных в МНТ

5. МДТС - счетчик для таблицы МДТ

6. МНТС - счетчик для таблицы МНТ

7. АЛА - массив списка параметров для подстановки индексных маркеров вместо формальных параметров, перед запоминанием определения.

ДАННЫЕ ДЛЯ ВТОРОГО ПРОСМОТРА

1. ВЫХ1 - Выходная копия текста после первого прохода

2. ВЫХ2 - Выходная копия текста после второго прохода

3. МДТ - таблица макроопределений, в которой хранятся тела макроопределений

4. МНТ - таблица имен, необходимая для хранения имен макрокоманд, определенных в МНТ

5. МДТС - счетчик для таблицы МДТ

6. МНТС - счетчик для таблицы МНТ

7. АЛА - массив списка параметров для подстановки индексных маркеров вместо формальных параметров, перед запоминанием определения.

АЛГОРИТМ

Ниже приведена формальная запись соответствующих алгоритмов обработки макроопределений двухпросмотровым способом.

Каждый из алгоритмов осуществляет построчный просмотр входного текста.

ПЕРВЫЙ ПРОСМОТР - МАКРООПРЕДЕЛЕНИЯ: Алгоритм первого просмотра проверяет каждую строку входного текста. Если она представляет собой псевдооперацию MACRO, то все следующие за ней строки запоминаются в ближайших свободных ячейках МДТ. Первая строка макроопределения - это имя самого макроса. Имя заносится в таблицу имен МНТ с индексом этой строки в МДТ. При этом происходит также подстановка номеров формальных параметров, вместо их имен. Если в течение просмотра встречается команда END, то это означает, что весь текст обработан, и управление можно передавать второму просмотру для обработки макрокоманд.

ВТОРОЙ ПРОСМОТР - РАСШИРЕНИЕ МАКРОКОМАНД: Алгоритм второго просмотра проверяет мнемонический код каждого предложения. Если это имя содержится в МНТ, то происходит обработка макропредложения по следующему правилу: из таблицы МНТ берется указатель на начало описания макроса в МДТ. Макропроцессор готовит массив списка АЛА содержащий таблицу индексов формальных параметров и соответствующих операндов макрокоманды. Чтение производится из МДТ, после чего в прочитанную строку подставляются необходимые параметры, и полученная таким образом строка записывается в ВЫХТ2. Когда встречается директива END, текст полученного кода передается для компиляции ассемблеру.

*Первый просмотр*

Начало алгоритма

МДТС = 0

МНТС = 0

ФЛАГ ВЫХОДА = 0

цикл пока (ФЛАГ ВЫХОДА == 0) {

чтение следующей строки ВХТ

если !(операция MACRO) {

вывод строки в ВЫХТ1

если (операция END) ФЛАГ ВЫХОДА = 1

}

иначе {

чтение идентификатора

запись имени и индекса в МНТ

МНТС ++

приготовить массив списка АЛА

запись имени в МДТ

МДТС ++

цикл {

чтение следующей строки ВХТ

подстановка индекса операторов

добавление в МДТ

МДТС ++

} пока !(операция MEND)

}

}

переход ко второму проходу

конец алгоритма

*Второй просмотр*

Начало алгоритма

ФЛАГ ВЫХОДА = 0

цикл пока (ФЛАГ ВЫХОДА == 0) {

чтение строки из ВЫХТ1

НАЙДЕНО = поиск кода в МНТ

если !(НАЙДЕНО) {

запись в ВЫХТ2 строки

если (операция END) {

ФЛАГ ВЫХОДА = 1

}

}

иначе {

УКАЗАТЕЛЬ = индекс из МНТ

Заполнение списка параметров АЛА

цикл {

УКАЗАТЕЛЬ ++

чтение след. строки из МДТ

подстановка параметров

вывод в ВЫХТ2

} пока !(операция MEND)

}

}

переход к компиляции

конец алгоритма

ОДНОПРОСМОТРОВЫЙ АЛГОРИТМ

Предположим, что мы допускаем реализацию макроопределения внутри макроопределений. Основная проблема здесь заключена в том, что внутреннее макро определено только после того, как выполнен вызов внешнего. Для обеспечения использования внутреннего макро нам придется повторять как просмотр обработки макроопределений, так и просмотр обработки макрокоманд. Однако существует и еще одно решение, которое позволяет произвести распознавание и расширение в один просмотр.

Рассмотрим аналогию с ассемблером. Макроопределение должно обрабатываться до обработки макрокоманд, поскольку макро должны быть определены для процессора раньше, чем макрокоманды обращения к ним. Однако, если мы наложим ограничение, что каждое макроопределение должно быть определено до того, как произойдет обращение к нему, мы устраним основное препятствие для однопросмотровой обработки. Заметим, что то же самое может быть верно и для символических имен в ассемблере, но такое требование было бы неоправданным ограничением для программиста. В случае же макрорасширения может быть вполне естественно потребовать, чтобы объявления макро предшествовали вызовам. Это не накладывает очень существенных ограничений на использование аппарата макрокоманд. Этот механизм даже не запрещает обращение макро к самому себе, поскольку обращение ведется в тот момент, когда имя макроса уже определено. Расширение же макроса идет не в процессе разбора макроса, а в процессе последующего вызова.

Предложенный ниже алгоритм объединяет два вышеприведенных алгоритма для двупросмотрового макроассемблера в один.

АЛГОРИТМ

*Однопросмотровый макроассемблер*

Начало алгоритма

МТДС = 0

МНТС = 0

ФЛАГ ВЫХОДА = 0

цикл пока !(ФЛАГ ВЫХОДА) {

чтение следующей строки ВХТ

НАЙДЕНО = поиск кода в МНТ

если (НАЙДЕНО) {

МДИ = 1

УКАЗАТЕЛЬ = индекс из МНТ

Заполнение списка параметров АЛА

цикл {

УКАЗАТЕЛЬ ++

чтение след. строки из МДТ

подстановка параметров

вставка во ВХТ

} пока !(операция MEND)

иначе если !(операция MACRO) {

вывод строки в ВЫХТ1

если (операция END) ФЛАГ ВЫХОДА = 1

}

иначе {

чтение идентификатора

запись имени и индекса в МНТ

МНТС ++

приготовить массив списка АЛА

запись имени в МДТ

МДТС ++

цикл {

чтение следующей строки ВХТ

подстановка индекса операторов

добавление в МДТ

МДТС ++

} пока !(операция MEND)

}

}

конец алгоритма

ОПИСАНИЕ АЛГОРИТМА

данный алгоритм является упрощением алгоритма приведенного в [1], глава 4.3.2. Различие состоит в том, что современные средства интеллектуализации программирования дают нам возможность осуществлять вставки и удаления из крупных массивов с минимальными затратами процессорного времени, что было невозможно при использовании перфокарт. Кроме того, скорость работы современных процессоров настолько велика, что позволяет производить прямые вставки и удаления в массивах данных средней величины (скажем, до 64 килобайт) в режиме реального времени. Таким образом, расширение исходного макроса может быть напрямую вставлено в массив исходного текста и обработано в расширенном виде. Такая технология позволяет значительно упростить алгоритм обработки макроязыка.

РЕАЛИЗАЦИЯ ВНУТРИ АССЕМБЛЕРА

Макропроцессор, описанный нами предназначался для обработки текста в режиме препроцессора, то-есть он выполнял полный просмотр входного текста, до того, как передать управление ассемблеру. Но макропроцессор также может быть реализован внутри первого прохода ассемблера. Такая реализация позволяет исключить промежуточные файлы, и позволяет достичь на порядок большей интеграции макропроцессора и ассемблера путем объединения сходных функций. Например, возможно объединение таблиц имен макросов и имен кода операции; специальный признак может указывать на то макро это или встроенная операция.

Основные преимущества включения макропроцессора в первый просмотр состоят в следующем:

Многие функции не надо реализовывать дважды (например, функции ввода-вывода, проверки на тип, и.т.п.)

В процессе обработки отпадает необходимость создавать промежуточные файлы или массивы данных.

У программиста появляются дополнительные возможности по совмещению средств ассемблера (например, команды EUQ) совместно с макрокомандами.

Основные недостатки:

Программа должна требовать больше оперативной памяти, что критично на некоторых типах ЭВМ, не имеющих много оперативной памяти.

Реализация подобного типа задачи может оказаться на порядок сложнее, чем отдельная реализация ассемблера и макропроцессора.

Отдельно от рассмотрения реализации аппарата макросредств в ассемблер лежит рассмотрение дополнительного просмотра, используемого многими программами для выявления определенных характеристик исходной программы, таких как типы данных. Располагая таким макропроцессором, можно использовать команды условной компиляции, позволяющие поставить расширение макрокоманд в зависимость от определенных характеристик программы.

ВЫВОДЫ

Макроязыки и соответствующие им макропроцессоры представляют собой самостоятельную форму языков программирования. При использовании вместе с ассемблером, макропроцессор является для программиста полезным инструментом и по существу, позволяет ему самому определять свой язык “высокого” уровня.

Существуют четыре основных задачи, решаемых макропроцессором:

Распознавание макроопределений

Хранение макроопределений

Распознавание макрокоманд

Расширение макрокоманд и подстановка параметров

Макропроцессор в ассемблере может быть реализован несколькими способами:

Независимый двухпросмотровый ассемблер

Независимый однопросмотровый ассемблер

Процессор, совмещенный с первым проходом стандартного двухпросмотрового ассемблера.