УДК 518.5

Распределение регистров в рабочей программе

А. Н. Терехов

В статье рассматривается метод распределения регистров, позволяющий получить эффективную рабочую программу. Уточняются задачи, стоящие перед системой распределения регистров. Хотя нахождение оптимаьлного варианта в принципе требует полного перебора, предлагаются простые алгоритмы, которые в большинстве случаев дают результат, близкий к оптимальному. Метод основан на автоматическом сравнении многих вариантов программирования.

В [1] предложен метод синтеза эффективной рабочей программы в компиляторах для вычислительных машин, допускающих разнообразные способы представления информации. Этот метод позволяет использовать информацию о последующем тексте программы с помощью запуска многих вариантов программирования. При этом транслируемый текст просматривается строго последовательно, а информация о конкурирующих вариантах программирования отражается специальным представлением значений в транслирующей системе. Создание вариантов осуществляется с помощью "предсказателей" — специальных логических переменных, значения которых выбираются системой так, чтобы минимизировать "стоимость" рабочей программы. Полного перебора программы при этом не происходит, так как многие варианты "ортогональны" друг другу (т.е. решения в различных точках программы принимаются независимо друг от друга), кроме того, каждый вариант программирования касается определенной конструкции и, вообще говоря, не влияет на программу после окончания этой конструкции.

1. ПОСТАНОВКА ЗАДАЧИ

В статье будем вести изложение для определенности в применении к архитектуре ЕС ЭВМ. Основной особенностью системы команд ЕС ЭВМ, приводящей к появлению альтернативных, но не равноценных вариантов программирования одной и той же последовательности операций, является возможность размещать величины или их адреса либо в оперативной памяти, либо на регистрах, причем при использовании регистров также возможны различные варианты. Поэтому наиболее трудной задачей при построении эффективной рабочей программы является рациональное использование регистров.

Система распределения регистров должна решать по крайней мере следующие задачи:

1) выделение базовых регистров для адресации команд текущей программной секции (командные регистры);
2) выделение базовых регистров для адресации статически доступных величин, т.е. величин, относительные адреса которых вычисляются во время трансляции (статические регистры);

"Программирование", №1, 1977, стр. 37-41

3) выделение регистров для хранения промежуточных результатов (анонимные регистры);
4) выделение регистров для постоянного использования в определенной роли в пределах некоторого блока (специальные регистры), например, выделение регистров для программирования циклов;
5) обеспечение многократного использования значений в регистрах и возможности использования информации, оставшейся в регистрах после предшествующих действий (остаточная информация).

В соответствии с принятой структурой рабочей программы статические и командные программы выделяются в начале трансляции процедуры, но при этом длина процедуры и ее статической секции еще не известны, поэтому не известно, сколько регистров потребуется для этой цели. В связи с этим каждый статический и каждый командный регистр, кроме первых, выделяется условно, для чего с каждым таким регистром связывается новый предсказатель.

При программировании различных конкретных конструкций входного языка применяется техника абстрактного стека и запросов на размещение. Внешняя конструкция выдает внутренним конструкциям запросы, описывающие допустимые способы размещения выдаваемых ими значений, а каждая внутренняя конструкция в ответ помещает на абстрактный стек характеристику размещения выдаваемого ею значения. Программа трансляции каждой конструкции оформляется в форме макроопределения на специальном языке, причем язык и реализующая его система выбраны так, чтобы максимально упростить написание макроопределений. При этом многие функции перекладываются на предсказатели, в частности в этой системе внутренняя конструкция никогда не будет освобождать для себя анонимный регистр, занятый внешней конструкцией, так как одновременно с вариантом, где этот регистр был занят, будет рассматриваться вариант, в котором этот регистр или не был занят, или был освобожден, занявшей его конструкцией.

С другой стороны, из-за того, что занятие и освобождение регистра является условным действием, состояние каждого конкретного регистра, если регистры выделять по конкретным номерам, будет зависеть от многих предсказателей, а условное выражение, определяющее номер свободного регистра, окажется еще сложнее. В связи с этим информацию, используемую для распределения регистров, нужно представить таким образом, чтобы по возможности уменьшить использование условных значений. Одним из средств для этого является использование вместо номеров регистров (которые могут быть и условными) символических имен этих регистров.

Если не стремиться к использованию остаточной информации в регистрах, то для выделения регистров можно использовать следующий простой алгоритм: ведется счетчик (условный), указывающий наибольший номер регистра; при отведении нового регистра значение этого счетчика увеличивается на единицу и выдается регистр с полученным номером, причем этому номеру сразу присваивается символическое обозначение (при помощи команды *EQU*); при освобождении регистра счетчик уменьшается на единицу. Если при увеличении на единицу счетчик выходит за пределы допустимых номеров регистров, то данный вариант бракуется (ему приписывается бесконечная стоимость). Разумеется, все операции над счетчиком являются условными. Аналогичным образом можно обеспечить выделение двойных регистров для команд типа *M*, *SLDL*, *BXLE* и т.п. Такие регистры можно выделять со стороны больших

номеров, уменьшая каждый раз соответствующий счетчик на два до встречи (условной!) с первым счетчиком.

Для обеспечения многократности использования значений в регистрах и использования остаточной информации требуются более сложные приемы. Уточним, какого рода информацию предполагаем использовать. Для того чтобы при необходимости помещения на регистр некоторого значения можно было узнать, что такое значение уже находится на каком-либо регистре, нужно использовать какую-то содержательную *характеристику* этого значения. Предлагается учитывать следующие характеристики:

1) значение, которым обладает некоторый идентификатор;
2) результат разыменования значения типа 1);
3) результат выборки поля из значений типов 1), 2);
4) значение является константой.

Будем называть такие значения *узнаваемыми*. Задача использования остаточной информации состоит из определения "времени жизни" остаточного значения и задания по возможности одного и того же регистра для различных использований значений с одинаковой характеристикой.

Отметим, что время жизни остаточного значения ограничивается следующими обстоятельствами:

1) загрузка в тот же регистр другого значения;
2) встреча с меткой, на которую управление могло быть передано до появления на регистре интересующего нас значения;
3) встреча с меткой, на которую могло быть передано управление из последующего текста, в котором данное остаточное значение прекращает существование;
4) вход в цикл, в теле которого данное остаточное значение прекращает существование;
5) слияние с ветвью программы (при выходе из условного предложения или последовательного предложения, содержащего завершители), в которой этот же регистр используется иначе;
6) присваивание имени для значений, характеризуемых как результат разыменования;
7) выход из блока, где описан идентификатор для значений, характеризуемых через него.

Заметим, что передачи управления и перегрузка регистров могут быть скрыты в вызовах процедур, однако, вызовы процедур (за исключением вызовов, реализованных особым образом) восстанавливают содержимое всех регистров. Кроме того, передача управления на метку данного блока из вызова возможна только в том случае, когда область действия вызываемой процедуры или одного из ее фактических параметров меньше данного блока.

Таким образом, для использования остаточных значений необходимо в каждый момент времени иметь информацию обо всех узнаваемых значениях, находящихся в данный момент в регистрах. Естественно, что в описываемой системе вся эта информация будет условной, так как во-первых, выделение регистров происходит условно, а, во-вторых, для определения времени жизни в некоторых случаях используется информация о последующем тексте.

Далее мы опишем способ представления такой информации и способ выделения регистров, ориентированный на то, чтобы при повторном появлении узнаваемого значения ему присваивался по возможности тот же регистр, что и прежде.

2. СИСТЕМА РАСПРЕДЕЛЕНИЯ РЕГИСТРОВ

Обычным решением задачи отведения регистров и использования остаточной информации (см., например, [2]) является следующее. Определяется таблица, содержащая по одной записи на каждый регистр (сюда включаются регистры общие и с плавающей запятой, а также разряды кода условия в слове состояния программы). При генерации команды загрузки в регистр узнаваемого значения в соответствующую запись таблицы заносится характеристика этого значения. По этой таблице можно выбрать свободный для отведения регистр, а также определить, имеется ли интересующее нас узнаваемое значение на регистрах. Специальные приемы (часто весьма изощренные) применяются для определения порядка использования регистров в случае невозможности удовлетворения всех запросов на регистры.

В описываемой реализации ответственность за порядок использования регистров полностью перекладывается на транслирующую систему, однако для нее не подходит описанная выше таблица использования регистров. Дело в том, что система использует специальное представление значений, а именно, двоичные деревья, узлы которых соответствуют предсказателям, а висячие вершины — конкретным значениям. Поэтому если для каждого регистра хранить дерево характеристик, то сложно найти нужное узнаваемое значение, а если хранить список узнаваемых значений, то трудно определить номер очередного свободного регистра (придется просматривать деревья всех узнаваемых значений). Удобно иметь сразу две таблицы: регистров и узнаваемых значений с перекрестными ссылками.

Очевидно, что с помощью описываемой системы можно обеспечить полный перебор всех вариантов генерации программы в целом. Для этого нужно выбирать значения всех предсказателей в самом конце программы и для всех спорных случаев с помощью предсказателей создавать все возможные варианты. Например, при необходимости пожертвовать одним из узнаваемых значений на регистрах можно попробовать выгрузить все значения по очереди. Однако возникающее при этом огромное число различных вариантов не поддается уже никакому учету.

В данной работе предлагается некоторое компромиссное решение, которое при незначительных (по сравнению с полным перебором) потерях в качестве рабочей программы позволяет резко уменьшить число сравниваемых вариантов. Предлагается выбирать значения предсказателей в конце того макроопределения, в котором они были введены. При этом возможность повторного использования значений, содержащихся в регистрах, в подконструкциях той конструкции, которой соответствует макроопределение, явным образом учитывается системой, а использование остаточных значений управляется значительно более простыми алгоритмами (по таблице узнаваемых значений).

Перейдем к более детальному описанию предлагаемой техники использования регистров. Для описания структуры данных и алгоритмов будем использовать средства Алгола-68. Один из основных приемов повышения эффективности описываемых процедур состоит в возможно более частом использовании безусловных значений, в частности используется специальная операция **tree** для проверки тождественности деревьев константам (в данной реализации конкретные значения могут быть только неотрицательными целыми

числами, а отрицательные числа представляют собой адреса условных значений; если формула **tree** *a* выдает неотрицательный результат, то *a* является безусловным значением). Заметим, что в любой процедуре все действия над локальными значениями до первого условия или цикла, управляемого предсказателем, можно выполнять безусловно.

Определяем следующие два массива:

[N] **struct** (**int** *RU*, *RB*) *R*,

[M] **struct** (**int** *NAME*, *REG*) *C*;

Здесь *N* – количество регистров, подлежащих распределению, *M* – количество одновременно существующих на регистрах узнаваемых значений, допускаемых реализацией. Очевидно, что *M* может быть больше, чем *N*, так как в одном регистре могут быть разные узнаваемые значения в зависимости от значений предсказателей. *N* и *M* — безусловные значения.

Поле *RU* может иметь следующие значения:

- 1, если соответствующий регистр свободен;
- 2, если регистр свободен, но содержит остаточное узнаваемое значение;
- значение, большее, чем 2, если регистр занят (в этом случае поле *RU* отражает также кратность использования регистра, то есть количество значений на абстрактном стеке, доступных через данный регистр).

Значением поля *RB* является ссылка на массив *C*, если соответствующий регистр содержит узнаваемое значение, и 0 – в противном случае.

Поле *NAME* – характеристика значения (безусловная). Мы считаем, что характеристики каким-либо способом закодированы целыми числами.

Поле *REG* – номер регистра, в котором хранится данное узнаваемое значение. Если значение поля *REG* тождественно равно 0 (то есть является безусловным значением), то весь элемент массива *C* считается неиспользуемым.

1. **proc** *SEARCHREG* = (**int** j) **int**:
   begin int $k := 1$; **while** $k \le M$ **and**
   (*NAME* **of** *C* [*k*] $\neq j$ **or tree** *REG* **of** *C* [*k*] = 0) **do** $k +:= 1$ **od**;

Процедура, состоящая только из безусловных действий, выдающая индекс элемента массива *C* с характеристикой *j* (*M* + 1, если такого элемента нет). Предполагается, что в *C* для каждой характеристики всегда будет не более чем один элемент с полем *REG*, не равным тождественно 0.

2. **proc** *REPREG* = (**int** *k*) **void**:
   RU **of** *R* [*REG* **of** *C* [*k*]]+:=1;

Обеспечивает повторное использование регистра, содержащего узнаваемое значение с характеристикой *NAME* **of** *C* [*k*].

3. **proc** *OCPREG1* = (**int** *i*) **void**:
   begin ref int *r* = *RB* **of** *R*[*i*]; **if** $r \neq 0$ **then**
   REG **of** *C*[*r*] := 0; r := 0 **fi**; *RU* **of** *R*[*i*] := 3 **end**;

Занимает регистр *i* неузнаваемым значением.

4. **proc** *OCPREG2* = (**int** *i*, *j*) **void**:
   begin int *k* := 1; **while tree** *REG* **of** *C* [*k*] ≠ 0 **do** *k* +:= 1 **od**;
   C [*k*] := (*j*, *i*);
   co Все предыдущие действия можно выполнять безусловно **co**
   R [*i*] := (3, *k*)
   end;

Занимает регистр *i* узнаваемым значением с характеристикой *j*.

5. **proc** *FREEREG* = (**int** *i*) **void**:
   RU **of** *R* [*i*] := **if** *RB* **of** *R* [*i*] = 0 **then** 1 **else** *R* [*i*] - 1 **fi**;

Эта процедура освобождает регистр *i*.

6. **proc** *GETREG* = **int**:
   begin int *i* := 1;
   while (*i* ≤ *N* | *RU* **of** *R* [*i*] ≠ 1 | **false**) **do** i +:= 1 **od**;
   if *i* ≤ *N* **then** *i* **else** *i* := 1;
   while (*i* ≤ *N* | *RU* **of** *R* [*i*] ≠ 2 | **false**) **do** i +:= 1 **od**;
   if *i* ≤ *N* **then** *i* **else fail**; **skip fi**
   fi end;

Эта процедура выдает номер регистра, но оставляет его свободным; не сбрасывается даже информация об остаточном значении.

Если свободных регистров нет, то текущий вариант (то есть данный набор значений предсказателя) бракуется. На самом деле это не означает, что дальнейшее программирование невозможно вообще, так как предполагается, что параллельно с каждым вариантом, использующим регистры, был определен вариант программирования без использования регистров.

Поступила в редакцию 9.III.1976

ЛИТЕРАТУРА

1. *А. Н. Терехов, Г. С. Цейтин.* Средства эффективного синтеза объектной программы. — Программирование, 1975, №6.
2. *Д. Грис.* Конструирование компиляторов для цифровых вычислительных машин. М., "Мир", 1975.