1 20. Трансляторы, интерпретаторы и компиляторы. Стадии работы компилятора. Генерация и оптимизация кода.

Начало то же, что и в 19 билете.

1.1 Генерация и оптимизация кода

Перед генератором кода стоят три основные задачи: выбор команд, распределение и назначение регистров и упорядочение команд. Выбор команд означает выбор машинных команд целевой машины для реализации инструкций промежуточного представления. Распределение и назначение регистров означает принятие решения о том, какие значения в каких регистрах будут храниться. Упорядочение команд предусматривает принятие решения о том, в каком порядке должны выполняться сгенерированные команды.

1.2 Выбор команд

Генератор кода должен отобразить программу в промежуточном представлении на последовательность машинных команд, которые могут быть выполнены целевой машиной. Сложность этого отображения определяется такими факторами, как

- уровень промежуточного представления;
- природа архитектуры набора команд;
- требуемое качество генерируемого кода.

Выбор команд может быть смоделирован с использованием представления машинных команд и инструкций промежуточного представления в виде деревьев. Затем требуется "покрыть"дерево промежуточного представления множеством поддеревьев, соотвествующих машинным командам. Если каждому поддереву машинной команды назначить стоимость, то для генерации оптимальной последовательсти команд можно использовать динамическое программирование.

1.3 Распределение регистров

Регистры представляют собой наиболее быстрые вычислительные модули целевой машины, но обычно их слишком мало, чтобы хранить все

значения. Использование регистров часто разделяется на две подзадачи.

1. В процессе *распределения регистров* мы выбираем множество переменных, которые будут находиться в регистрах в каждой точке программы.

2. В фазе *назначения регистров* мы выбираем конкретные регистры для размещения в них переменных.

Поиск оптимального назначения регистров переменным представляет собой NP-полную задачу даже на машине с единственным регистром. Аппаратное обеспечение и/или операционная система целевой машины может накладывать дополнительные ограничения по использованию регистров. Например, умножение или деление требуют использования пар регистров.

1.4 Порядок вычислений

Измененеие порядка вычислений может привести к уменьшению числа требуемых регистров, а также лучшую производительность на конвеерных машинах. В общем случае поиск оптимального порядка вычислений – NP-полная задача.

1.5 Базовые блоки и графы потоков

Граф потоков – еще одно промежуточное представление. Оно строиться следующим образом.

1. Промежуточный код разделяется на *базовые блоки*, представляющие максимальные последовательности следующих друг за другом трехадресных команд, обладающие следующими свойствами:

   (a) поток управления может входить в базовый блок только через первую команду блока, переходы в середину блока отсутствуют;

   (b) управление покидает блок без останова или ветвления, за исключением, возможно, в последней команде блока.

2. Базовые блоки становятся узлами *графа потока*, ребра которого указывают порядок следования блоков.

Зачастую можно существенно снизить время работы кода, просто выполнив *локальную* оптимизацию внутри каждого блока. Большее можно получить при помощи *глобальной* оптимизации, которая рассматривает потоки информации между базовыми блоками программы.

Многие важные методы локальной оптимизации начинаются с преобразования базового блока в ориентированный ациклический граф.Такое представление базового блока позволяет выполнить ряд улучшающих преобразований.

1. Можно устранить локальные общие подвыражения, т.е. команды, которые вычисляют уже вычисленные значения.

2. Можно устранить неиспользуемый код, т.е. команды, вычисляющие никогда не используемые значения.

3. Можно переупорядочить инструкции, не зависящие друг от друга; такое переупорядочивание может снизить время хранения временного значения в регистре.

4. Можно применить алгебраические законы для переупорядочения операндов трехадресных команд.

Большинство глобальных оптимизаций основано на *анализах потоков данных*, которые представляют собой алгоритмы для сбора информации о программе. Все результаты анализов потоков данных имеют один и тот же вид: для каждой команды программы они указывают некоторое свойство, которое должно выполняться всякий раз при выполнении этой команды. Разные анализы отличаются вычисляемыми ими свойствами. Например, анализ распространения констант вычисляет для каждой точки программы и для каждой переменной, используемой в программе, имеет ли эта переменная в данной точке программы единственное константное значение или нет. Эта информация может, например, использоваться для замены обращений к переменным непосредственными константными значениями. В качестве другого примера анализ живучести переменных для каждой точки программы определяет, будет ли перезаписано значение, хранящееся в переменной, перед его прочтением. Если будет, то мы не должны заботиться о сохранении этого значения в регистре или ячейке памяти. Еще примеры: удаление бесполезного кода (if (debug)), перемещение кода.