Средства автоматизации построения трансляторов

LEX – генератор лексических анализаторов

YACC – генератор синтаксических анализаторов

Генерация кода

1) Распределение памяти для данных и команд программы
2) Формирование эквивалентных внутреннему представлению конструкций на выходном языке

```
case LEX_ADD:  Print ("\t ADD\t");
               Print(Pop());
               Print(Pop());
               break;
```

(вместо интерпретации ПОЛИЗа — генерация кода)

Оптимизация

критерии оптимизации:

- время выполнения программы
- объем программы
- равномерная загрузка оборудования многопроцессорного комплекса

…

Ручная оптимизация (например, с помощью ассемблерных вставок)

Оптимизации при трансляции:

- *Машинно-независимая оптимизация* — проведение преобразований исходной программы (или ее промежуточного внутреннего представления), не зависящих от выходного языка компилятора. (Происходит на фазе подготовки к генерации)

- *Машинно-зависимая оптимизация* — преобразование программы на выходном языке компилятора. (Происходит одновременно с генерацией объектной программы или после генерации)

Машинно-независимая оптимизация

Линейные участки

- вычисление константных выражений на стадии компиляции

1) A := sin (2 * 3.1415 * B + C); где B, C – константы

2) #define Pi 3.1415926
A := sin (2 * Pi * B+C);

3) int a [100][100][100] ;

a[((2 * 100) +4) * 100 + i] :=55; // a[2][4][i]

- арифметические преобразования

  A=B*C+B*D → A=B*(C+D)

- устранение общих подвыражений (избыточных вычислений)
- удаление ненужных присваиваний ("распространение копий") и других операций
- перестановка независимых смежных участков программ

  A := 2 * B * 3 *C; → A := (B * C) * (2 * 3);

- оптимизация вычисления логических выражений

Передача параметров и вызовы функций

- прямая подстановка тел функций в основной текст программы
- передача параметров через глобальные переменные, которые впоследствии связываются с регистрами центральных процессоров

Циклы

- вынесение инвариантных вычислений из тела цикла

  for (i= 0; i < 10; i ++) A [i] = B * C * A [i];

  → D = B * C;

  for (i= 0; i < 10: i ++) A [i] = D * A [i];

- замена операций с переменными цикла

  S = 10; **for** (i = 0; i < N; i ++) A [i] = i * S;

  i – индуктивная переменная

  → S = 10; T = 0;

  for (i = 0; i < 10; i ++) T = T + S, A [i] = T;

S = 10;
for (i = 0; i < N; i ++) = Q + F (S), S = S + 10;

→ элиминация индуктивной переменной:

S = 10; M = S + N * 10;
while (S <= M) {Q = Q + F (S) ; S = S + 10}

- слияние и развертывание циклов.

```
for (i = 0; i < N; i ++) { S1 }
for (i = 0; i < N; i ++) { S2 }
```

⟹

```
for (i = 0; i < N; i ++) { S1; S2 }
```

```
for (i = 0; i < n; i ++)
{if (x < y) { S1; }
 else       { S2; }
}
```

⟹

```
if (x < y)
        for (i = 0; i < n; i ++) { S1; }
else    for (i = 0; i < n; i ++) { S2; }
```

- вложенные циклы по обработке массивов могут заменяться на одиночный цикл с соответствующим пересчетом индексов.

Машинно-зависимая оптимизация

- учет регистровой структуры вычислительной аппаратуры (оптимальное распределение регистров, передача параметров в процедуры и функции через регистры)
- удаление лишних команд
- оптимизация потока управления и удаление недостижимых участков программ
- снижение "стоимости" программы (есть «дорогие» и «дешевые» команды с точки зрения времени их выполнения процессором)
- использование «машинных идиом» (например: xor ax,ax для обнуления регистра)
- слияние, дробление и развертывание циклов, иногда требующееся из-за технических особенностей аппаратуры
- учет векторных и конвейерных свойств архитектуры

Методы динамического распределения памяти

Явное выделение памяти

блоками фиксированного размера. Блоки памяти связываются в список, над которым достаточно легко выполнять операции их выделения и освобождения. Достоинства: если программа полностью использует блок памяти, то нет никаких дополнительных расходов. С каждым блоком связан указатель на его начало, требуется хранить только информацию о том, занят блок или нет.

блоками переменного размера. Возможна проблема из-за фрагментации памяти: . нельзя выделить блок, больший по размеру, чем имеющиеся свободные, хотя суммарного объема свободной памяти хватает. Тогда необходимо приостанавливать выполнение программы и начинать реорганизацию списка.

Один из методов выделения блоков переменного размера — **метод первого подходящего**.

Неявное выделение/освобождение памяти

Неявно выделяемые блоки памяти также могут быть фиксированного или переменного размера.

При неявном динамическом выделении и освобождении блоков памяти, выделяемые блоки обычно имеют следующую структуру:

- размер блока (если блок фиксированного размера, то эта информация не хранится, т.к. априори известна);
- пометка либо счетчик ссылок:
- указатели на блоки
- то, что досталось пользователю, заказавшему этот блок.

Счетчик ссылок — количество указателей в программе, которые ссылаются на этот блок (при присваивании указателей, например, $p = q$, счётчик для блока, на который указывал указатель p уменьшается на единицу, а для q увеличивается). Если счётчик становится равным нулю, то блок освобождается. (Существует проблема циклических ссылок, когда счётчики всегда >0.)

Пометка фиксирует, есть хотя бы один указатель, ссылающийся на этот блок, или нет. В некоторый момент начинает работать «сборщик мусора». Он помечает все блоки как недостижимые, а потом начинает анализ текущих указателей программы, и находит действительно используемые блоки, достижимые по указателям. Это является весьма трудоемким процессом. Блоки, на которые ничего не указывает, считаются свободными.