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 расщепление 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.
- развертывание циклов – замена тела цикла линейной последовательностью операторов

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

Распределение памяти

Распределение памяти - это процесс, в результате которого отдельным элементам исходной программы ставятся в соответствие адрес, размер и атрибуты области памяти, необходимой для размещения лексических единиц.
Область памяти - это блок ячеек памяти, выделяемых для данных и каким-то образом объединенных логически.
Распределение памяти выполняется после фазы анализа текста исходной программы **на этапе подготовки к генерации объектного модуля** (перед генерацией кода объектного модуля).

Исходными данными для процесса распределения памяти служат сведения о семантике конструкций ЯП, таблица идентификаторов, построенная лексическим анализатором и информация, полученная синтаксическим анализатором при анализе декларативной чисти программы.
Современные компиляторы, в основном, работают с относительными, а не с абсолютными адресами ячеек памяти.

Семантика программ подразумевает, что при их выполнении области памяти будут необходимы для хранения:

- кодов пользовательских программ;
- данных, необходимых для работы этих программ;
- кодов системных программ, обеспечивающих поддержку пользовательских программ в период их выполнения;
- записей о текущем состоянии процесса выполнения программ (например, записей об активации процедур).

По **способу использования** области памяти делятся на ***глобальные*** и ***локальные***, а по **способу распределения** – на ***статические*** и ***динамические***.

Классы памяти

Выделяемую память можно разделить на локальную / глобальную и статическую / динамическую.

Локальная память - это область памяти, которая выделяется в начале выполнения некоторого фрагмента результирующей программы (блока, функции, оператора…) и может быть освобождена по завершении выполнения данного фрагмента. Доступ к локальной области памяти всегда запрещен за пределами того фрагмента программы, в котором она выделяется.

Глобальная память - это область памяти, которая выделяется один раз при инициализации результирующей программы и действует все время выполнения программы. Как правило, глобальная область памяти доступна из любой части исходной программы.

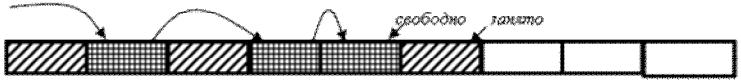
Статическая память - это область памяти, размер которой известен на этапе компиляции. Для статической памяти компилятор порождает некоторый адрес (как правило, относительный), и дальнейшая работа с ней происходит относительно этого адреса.

Динамическая память - это область памяти, размер которой становится известным только на этапе выполнения результирующей программы. Для динамической памяти компилятор порождает фрагмент кода, который отвечает за распределение памяти (ее выделение и освобождение). Как правило, с динамическими областям памяти связаны многие операции с указателями и с экземплярами объектов (классов) в ООЯП.

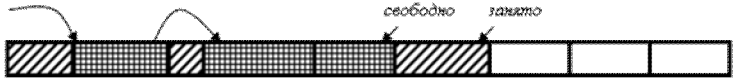
Динамические области памяти можно разделить на динамические области памяти, **выделяемые пользователем** и **непосредственно компилятором**.

Методы динамического распределения памяти

Явное выделение блоков фиксированного размера:



Явное выделение блоков переменного размера:



Неявное выделение динамической памяти

Неявно выделяемые блоки памяти 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 в нём не хранится.);
- счётчик ссылок, пометка (обычно есть либо одно, либо другое):

Счетчик ссылок подсчитывает количество указателей в программе, которые ссылаются на этот блок (например, при $p = q$, один счётчик (для блока, на который указывал указатель p) уменьшается на единицу, а другой увеличивается), если счётчик равен нулю, то блок не используется и его можно освободить. Существует проблема циклических ссылок, когда счётчики всегда > 0.

Пометка фиксирует, задействован данный блок или не задействован, т.е. у программы есть хотя бы один указатель, ссылающийся на этот блок. В некоторый момент начинает работать «сборщик мусора». Он помечает все блоки как недостижимые, а затем начинает анализ текущих указателей программы, что является очень трудоемким процессом. Блоки, на которые ничего не указывает, считаются свободными.