15. Простой двухпроходной ассемблер

Главной целью первого прохода является построение важнейшей внутренней таблицы АП – таблицы символических имен (ТСИ). Как следует из названия, эта таблица содержит все символические имена, используемые в исходной программе, т.е. встречающиеся в поле метки как при объявлении данных, так и при описании команд. Таблица должна содержать следующие поля:

- само символическое имя
- связанный с этим именем адрес области памяти
- дополнительная информация об имени, например – тип данных

В простейшем случае ТСИ можно реализовать как массив записей, но для ускорения поиска можно использовать структуру типа “дерево поиска” с ключами, в качестве которых выступают символические имена. В начале работы АП таблица является пустой.

Для отслеживания адресов, назначаемых данным и командам, используется специальная внутренняя переменная АП, называемая счетчиком адреса (СА). Начальное значение СА равно нулю при создании перемещаемых программ и равно заданному адресу загрузки при создании программ в абсолютном формате. При обработке очередной строки значение СА увеличивается на размер объявленных в этой строке данных или на длину описанной в строке команды. При этом алгоритмы обработки сегментов данных и кода немного отличаются.

Обработка строк сегмента данных включает следующие шаги:

- выделение имени из поля метки
- поиск имени в построенной к данному моменту ТСИ со следующими возможными ситуациями:
    - если имя найдено в ТСИ, то генерируется сообщение об ошибке типа “Дважды определенное имя”
    - если имя отсутствует в ТСИ, то оно в нее включается вместе с текущим значением переменной СА

- из поля команды выделяется директива описания данных, которая проверяется на правильность
- обрабатывается поле операндов и подсчитывается их количество
- выполняется проверка правильности описания операндов
- если необходимо, операнды переводятся во внутреннее представление
- переменная CA увеличивается на размер выделенной для операндов памяти (произведение числа операндов на их байтовый размер), тем самым становится известным адрес для размещения следующей порции данных

Например, пусть имеется следующий фрагмент объявления данных:

```
N1    DW   1234
N2    DB   1, 2, 3, 4
N3    DB   'Hello from assembler!'
N4    DD   DUP  10  (?)
```

Тогда имя N1 получит адрес CA=0, а так как в первой строке только один операнд-элемент данных размером в слово, то значение CA увеличится на 2. Имя N2 получит адрес 2, будет выделено 4 байтовых элемента данных и значение CA увеличится на 4. Имя N3 получит адрес 6, текстовая строка будет заменена кодами символов и значение CA увеличится на 21. Имя N4 получит адрес 27 и после обработки всех директив в четвертой строке АП увеличит значение CA на 40 (10 двойных слов), при этом никакие данные в эту области памяти занесено не будет, т.е. выполняется просто резервирование памяти.

Фрагмент ТСИ после обработки этих четырех строк будет содержать следующую информацию:

символическое имя	назначенный адрес (16-ричный)	дополнительная информация

N1	00 00 00 00	данные типа WORD
N2	00 00 00 02	данные типа BYTE
N3	00 00 00 06	символьные данные
N4	00 00 00 1B	нет

Значение CA после обработки этого фрагмента будет равно $67_{10} = 43_{16}$ и именно это значение будет использоваться для назначения адресов последующим элементам данных или командам в кодовом сегменте.

Сами данные будут представлены в выходном объектном модуле просто в виде последовательности байтов:

04 D2 01 02 03 04 kH ke kl kl ko . . . k! 00 00 00 . . . 00

1234 1,2,3,4 Коды символов 40 байтов

Теперь рассмотрим алгоритм обработки строк кодового сегмента на первом проходе. Реализация алгоритма предполагает использование специальной таблицы – так называемой таблицы кодов операций (ТКО). Эта таблица представляет собой массив записей, каждая из которых содержит следующие поля:

- мнемоническое обозначение команды, используемое при написании ассемблерных программ
- двоичный код команды
- байтовая длина команды
- дополнительная информация о команде

В отличие от ТСИ эта таблица является постоянной для текущей версии ассемблера, т.е. в процессе ассемблирования она только используется и не изменяется. Это позволяет реализовать ТКО наиболее эффективным для поиска способом, например – в виде хеш-таблицы с ключами-мнемониками команд. Пример таблицы кодов операций:

мнемоника	двоичный	длина	дополнительная

команды	код команды	команды	информация о команде
MOV	2A	4	
ADD	16	4	
JMP	0D	5	
CALL	CC	5	
. . . . .			

Алгоритм обработки строк с командами включает следующие шаги:

- анализ поля метки на пустоту и выделение имени-метки (если она есть)
- поиск имени в ТСИ и либо формирование сообщения об ошибке, либо добавление имени в ТСИ вместе с текущим значением переменной СА
- обработка поля кода операции с распознаванием в нем либо мнемонического кода операции, либо имени управляющей директивы
- поиск выделенного имени команды в ТКО с формированием соответствующего результата:
    - если МКК в ТКО не найден, то генерируется ошибка типа “Неопределенная команда”
    - если МКК в ТКО найден, то из ТКО извлекается двоичный код команды, который заменяет мнемонику, а также – дополнительная информация о команде, такая как возможный тип команды и ее возможная длина
- анализируется поле операндов для определения количества и типа операндов и каждый операнд обрабатывается следующим образом:
    - если операнд является непосредственной константой, то она переводится во внутреннее представление и включается в состав формируемой команды
    - если операнд является регистровым, то имя регистра заменяется его внутренним номером и включается в состав формируемой команды

- если операнд является символическим именем, то организуется его поиск в построенной к данному моменту ТСИ со следующими возможными результатами:
  - если имя найдено в ТСИ, то назначенный ему адрес подставляется в машинную команду (ссылки вперед не было) и тем самым формируется полная машинная команда
  - если имя в ТСИ не найдено (ситуация ссылки вперед), то окончательное формирование команды откладывается до второго прохода, вместо имени в команду записывается нулевой адрес и во вспомогательной таблице запоминается адрес этого нулевого поля и использованное в этом поле символическое имя
- после анализа и обработки операндов текущее значение СА увеличивается на длину команды
- частично обработанная команда сохраняется как промежуточный результат для второго прохода

Как видно, при отсутствии в исходном тексте ссылок вперед, уже на первом проходе будет выполнена практически вся необходимая работа. Тем не менее, возможна ситуация, когда некоторые команды потребуют обработки на втором проходе. Информация о всех таких командах сохраняется ассемблером во внутренней вспомогательной таблице.

После успешного завершения первого прохода построенная ТСИ будет содержать все используемые в программе имена, относящиеся как к данным, так и к командам.

На втором проходе сегмент данных можно уже не обрабатывать, т.к. он полностью обработан на первом проходе. Обрабатываются только те команды, которые не были до конца сформированы из-за появления ссылок вперед. Более подробно, АП должна выполнить следующие действия:

- просмотреть вспомогательную таблицу с несформированными командами
- выполнить поиск каждого символического имени из этой таблицы в ТСИ
- если имя в ТСИ не найдено, сгенерировать сообщение об ошибке типа “Неопределенное имя”
- если имя найдено, извлечь из ТСИ назначенный имени адрес и заменить этим адресом нулевое операндное поле
- сформировать окончательный вид машинной команды с указанием используемого способа адресации
- сохранить созданный код в виде объектного модуля некоторого формата

В простейшем случае выходной объектный модуль должен содержать следующую минимально необходимую информацию:

- имя модуля
- байтовая длина модуля, которая легко определяется как конечное значение переменной СА
- точка входа в программу, т.е. адрес первой выполняемой команды
- набор данных во внутреннем представлении
- набор машинных команд в двоичном коде

Такая структура объектного модуля предполагает его автономность, т.е. отсутствие компоновки с другими модулями. Более сложную структуру должны иметь модули, предназначенные для последующего объединения (компоновки) в единый исполняемый модуль.

Практические задания к теме №15.

Задание 1. Создать Windows-приложение для визуальной иллюстрации логики работы двухпросмотрового ассемблера

Исходные данные:

- набор основных команд процессора (8-10 команд), собранных в Таблицу Кодов Операций (ТКО)
- основные псевдокоманды-директивы (объявление данных, описание внешних имен и ссылок)
- небольшой исходный текст на языке ассемблера (5-6 команд, 2-3 псевдокоманды, 3-4 метки)

Результаты работы:

- промежуточные (после первого прохода): Таблица Символических Имен (ТСИ), вспомогательная таблица с частично сгенерированными командами
- окончательные: основные разделы объектного модуля (заголовок, таблица перемещений, таблицы внешних и общедоступных имен, тело), сообщения об ошибках

Порядок работы.

Работа выполняется в несколько этапов с постепенным наращиванием возможностей ассемблера

Этап 1. Простейший ассемблер для создания модулей в абсолютном формате

- создать основную форму, разместив на ней 3 панели и меню из двух команд
- в левой панели разместить 2 строковые сетки (компонент TStringGrid) для исходного текста и ТКО, и строку ввода для задания адреса загрузки
- в средней панели разместить 3 строковые сетки для вспомогательной таблицы, ТСИ и списка ошибок первого прохода

- в правой панели разместить 2 строковые сетки для заголовка объектного модуля и сообщений об ошибках, а также - список для созданного машинного кода
- команды меню должны запускать поочередно первый и второй проходы
- реализовать логику первого прохода в обработчике соответствующей команды
- ввести в сетки левой панели исходные данные, задать адрес загрузки и проверить работоспособность кода для нескольких примеров (в том числе – при наличии ошибок)
- реализовать логику второго прохода в соответствующем обработчике и проверить его работоспособность

Этап 2. Создание программ в перемещаемом формате

- добавить в правую панель список для запоминания адресов команд, операнды которых должны настраиваться при компоновке (таблица перемещений)
- изменить логику второго прохода для возможности формирования таблицы перемещений
- проверить работоспособность кода для нескольких вариантов исходного текста

Этап 3. Раздельное ассемблирование модулей

- расширить исходный текст, добавив псевдокоманду EXTRN для задания двух имен как внешних и псевдокоманду PUBLIC для задания 1-2 общедоступных имени
- изменить логику первого прохода: добавить обработку псевдокоманд описания общедоступных и внешних имен, добавить в сетку для ТСИ столбец для запоминания признака общего имени
- добавить в правую панель сетку для отображения таблицы внешних имен и сетку для отображения таблицы общедоступных имен

- изменить логику второго прохода: ввести проверку символических операндов как внешних имен, добавить формирование таблицы внешних имен и таблицы общедоступных имен
- проверить работоспособность созданного кода на нескольких примерах
- добавить функцию сохранения объектного кода в текстовом (для наглядности) файле в полном перемещаемом формате

Рекомендации по использованию компонента TStringGrid

При **визуальном** проектировании формы надо с помощью **Object Inspector** установить для сетки-таблицы следующие **опубликованные** свойства:

- **ColCount** и **RowCount** – в значения, равные числу столбцов и строк
- **DefaultRowHeight** (высота строки) – в 15 пикселов
- **Options / goEditing** (возможность редактирования сетки пользователем) – в **true (только для двух входных таблиц!)**
- **ScrollBars** – в значение **ssNone** (отключение полос прокрутки)
- **FixedCols** и **FixedRows** – в значение 0 (нет фиксированных строк и столбцов)

Кроме того, **программно** доступны следующие свойства:

- **Col, Row** – координаты текущей активной клетки
- **Cells** – индексируемый двумерный массив клеток
- **Cols** – одномерный массив столбцов в виде строковых объектов (типа **TStrings**)
- **Rows** – одномерный массив строк в виде строковых объектов (типа **TStrings**)

Рекомендации по обработке строк

- Для сравнения двух строк можно использовать стандартную функцию **CompareText(строка, строка**), которая возвращает 0 при совпадении строк. При этом регистр **НЕ** учитывается. Параметр **строка** устанавливается как значение соответствующей клетки необходимой таблицы

- В клетках сетки-таблицы находится **текстовая** информация, поэтому для выполнения арифметических операций с 16-ричными адресами надо строку преобразовать в целое число с помощью функции **StrToInt ('$'+текст).** Здесь префикс $ является признаком 16-ричного целого числа. Обратное преобразование целого числа в текстовую строку в 16-ричном представлении выполняется функцией **IntToHex (число, 8),** где 8 задает количество 16-ричных цифр в числе.