Московский ордена Трудового Красного Знамени
физико-технический институт
Ордена Ленина и ордена Октябрьской Революции
институт атомной энергии им. И. В. Курчатова
Ордена Ленина институт прикладной математики
Академии наук СССР

Вакуленко С. В.

Разработка переносимой Си-ориентированной инструментальной системы программирования для ЭВМ Эльбрус-Б.

Дипломная работа.

Научные руководители:

Платонов Александр Константинович,
доктор физико-математических наук,
ИПМ АН СССР

Пасынков Игорь Григорьевич,
кандидат физико-математических наук,
ИАЭ им. И. В. Курчатова

Москва
1989

Содержание

1

Цель и результаты работы.

Цель работы.

- Изучение возможности и способов переноса портабильного Си-компилятора на машины словной архитектуры, в частности, Эльбрус-Б.
- Разработка структуры объектных файлов, а также системы ассемблирования и загрузки, с учетом использования их при переносе ОС UNIX на Эльбрус-Б.
- Разработка представления данных, распределения регистров, соглашения о связях.
- Перенос портабильного Си-компилятора на ЭВМ Эльбрус-Б.

Результаты работы.

- Как для первого, так и для третьего режима Эльбрус-Б разработаны представление данных, распределение регистров, соглашения о связях.
- Для первого и третьего режимов Эльбрус-Б разработаны структуры объектных файлов, а также созданы загрузчик, ассемблер, библиотекарь и набор программ для работы с объектными файлами (*nm*, *lorder*, *size*, *strip*).
- Для первого и третьего режимов Эльбрус-Б созданы кросс-Си-компиляторы на базе портабильного компилятора Джонсона, а также библиотека-на-счете.
- Для обоих компиляторов собраны библиотеки стандартных функций языка Си.
- В целях отладки системы разработан эмулятор первого режима команд Эльбрус-Б.
- Создан набор стандартных файлов-заголовков, предназначенных для использования при переносе ОС UNIX.
- Для облегчения отладки системы создан деассемблер выполняемого (объектного) файла.
- Разработана документация по полученным программным продуктам.

2

Введение.

В истории отечественной вычислительной техники особое место занимают ЭВМ ряда БЭСМ. Почти пятнадцать лет БЭСМ-6 была лучшей из советских вычислительных машин. Накопленное на ней программное обеспечение исчисляется миллионами команд и тысячами человеко-лет. В последние десять лет в вычислительных центрах страны успешно эксплуатируется аналог БЭСМ-6 на интегральных микросхемах - СВС, имеющий существенно большую память и высокую производительность.

В настоящее время начинается серийный выпуск новой машины, продолжающей линию БЭСМ-6, - ЭВМ Эльбрус-Б. Эта машина имеет три режима команд. Первый режим совместим с БЭСМ-6. Второй режим имеет ту же систему команд, но адресное пространство увеличено с 32 Кслов до 128 Мслов. В третьем режиме длина слова увеличена с 48 до 64 бит, адресное пространство - 128 Мслов, система команд расширена. Объем физической памяти Эльбрус-Б - 64 Мбайта, производительность - около 5 млн. операций в секунду (MIPS).

Институтом точной механики и вычислительной техники (ИТМ и ВТ, Москва) проводятся работы по адаптации ОС ДИСПАК (БЭСМ-6) для Эльбрус-Б. Система обеспечивает пакетный счет задач и элементарные средства диалога и предназначена для облегчения использования накопленного на БЭСМ-6 программного обеспечения в новой среде. Однако по своей идеологии она находится на уровне конца шестидесятых годов и значительно отстает от используемых в мире операционных систем.

В связи с этим в Новосибирском филиале ИТМ и ВТ (НФ ИТМ и ВТ, Новосибирск) и Институте атомной энергии (ИАЭ, Москва) были начаты работы по переносу операционной системы ДЕМОС (аналог UNIX) на Эльбрус-Б. Данная работа является одним из этапов этой деятельности.

3

Метод раскрутки и его применение к переносу Си-компилятора.

Классическая схема переноса ОС UNIX выглядит следующим образом. Имеется целевая машина с терминалом (консолью), дисковым и ленточным накопителями и инструментальная ЭВМ (желательно под ОС UNIX - это облегчает задачу).

1. На инструментальной машине создается кросс-компилятор, ассемблер, загрузчик и необходимые библиотеки для целевой машины.
2. Создаются драйверы консоли, ленты, диска для целевой машины.
3. Собирается программа - начальный загрузчик (*boot*) для целевой машины. Ее функция - считывать с терминала команды оператора и производить проверку и разметку дисков и лент, копирование файлов а также загрузку программ с ленты или диска в память для выполнения.
4. Собираются утилиты для создания и проверки файловой системы на диске.
5. Начальный загрузчик и утилиты записываются на ленту и переносятся на целевую машину, где с их помощью производится форматирование диска и создание файловой системы.
6. На инструментальной машине переписывается машинно-зависимая (ассемблерная) часть ядра (монитор памяти, обработка прерываний, таймер) и собирается версия ядра для целевой машины.
7. Собираются необходимые программы для работы ядра (*init*, *getty*, *sh* и пр.).
8. Ядро и утилиты переносятся в файловую систему целевой машины.
9. Производится загрузка ядра с диска и проверка на работоспособность. При обнаружении ошибок вносятся правки в тексты ядра и программ на инструментальной машине, и процесс повторяется.
10. После отладки ядра на целевую машину переносятся скомпилированные на инструментальной машине компилятор, ассемблер, загрузчик, библиотеки.
11. Тексты компилятора переносятся на целевую машину и транслируются. Полученная программа проверяется на работоспособность.
12. Таким же образом производится перенос текстов остальных компонент с инструментальной машины. После этого необходимость в инструментальной машине отпадает.

Первый пункт изложенной схемы занимает от 35% до 40% всей работы по переносу ядра операционной системы.

Таким образом, метод раскрутки для переноса Си-компилятора выглядит так:

1. Создание кросс-компилятора, генерирующего код для целевой машины.
2. Создание кросс-ассемблера, кросс-загрузчика и необходимых библиотек.
3. Сборка компилятора «на самом себе». Трансляция текстов кросс-компилятора с получением кода для целевой машины.
4. Перенос компилятора в кодах на целевую машину и тестирование. Исправление ошибок в текстах и повторение пунктов 3, 4 до получения работающей версии компилятора на целевой ЭВМ.
5. Перенос текстов, трансляция и отладка на целевой машине.

4

Система загрузки.

4.1. Сегментация исполняемой программы.

На стадии выполнения адресное пространство программы делится на шесть сегментов (в режиме БЭСМ - пять сегментов).

1. Сегмент констант. Содержит литеральные константы, используемые программой. Доступен только на чтение.
2. Сегмент команд. Содержит выполняемые коды программы. Доступен только на выполнение.
3. Сегмент инициализированных данных. Содержит глобальные переменные программы, имеющие начальные значения. Доступен на чтение и запись.
4. Сегмент неинициализированных данных. Содержит глобальные переменные программы, не имеющие начального значения (инициализируются нулем) и память, выделяемую динамически. В режиме Эльбрус-Б располагается в привилегированной области памяти (начальные 512 Кслов).
5. Сегмент массивов (только для режима Эльбрус-Б). Аналогичен сегменту неинициализированных данных, но, в отличие от него, может располагаться в непривилегированной области памяти. Сюда помещаются все глобальные неинициализированные массивы.
6. Сегмент стека. Расширяется автоматически. Доступен на чтение и запись.

Сегменты констант и команд могут разделяться всеми процессами, выполняющими данную программу, что сильно экономит память и пространство подкачки.

4.2. Структура объектного файла для режима БЭСМ.

Объектный (а также исполняемый) файл для первого режима (БЭСМ) состоит из шести частей:

- Заголовок файла. Содержит следующую информацию:
  - признак типа объектного файла - обычный или разделяемый

  - длина сегмента констант
  - длина сегмента команд
  - длина сегмента инициализированных данных
  - длина сегмента неинициализированных данных
  - длина таблицы символов
  - адрес входа (словный)
  - признак неперемещаемости. Установлен в 1, если файл содержит программу в абсолютных адресах и информация о настройке удалена.
- Сегмент констант.
- Сегмент команд.
- Сегмент инициализированных данных.
- Информация о настройке, по полслова на каждые полслова из сегментов констант, команд, инициализированных данных. Содержит:
  - тип настройки M (абсолютное значение, относительно внешнего имени, относительно одного из сегментов)
  - формат настройки X (словный адрес или короткоадресная команда)
  - ссылка на таблицу символов (при настройке относительно внешнего имени)
- Таблица символов. Для каждого символа указаны:
  - тип (неопределенный, абсолютный, принадлежащий одному из сегментов или имя файла) и признак внешнего символа
  - значение (словный адрес)
  - длина имени (от 1 до 255) или 0 - признак конца таблицы символов
  - имя символа

Размеры всех сегментов и таблицы символов должны быть кратны длине слова (6 байт).

4.3. Структура объектного файла для режима Эльбрус-Б.

Объектный файл для третьего режима команд (Эльбрус-Б) состоит из шести частей:

- Заголовок файла:
  - признак типа объектного файла
  - длина сегмента констант
  - длина сегмента команд
  - длина сегмента инициализированных данных
  - длина сегмента неинициализированных данных
  - длина сегмента массивов
  - длина таблицы символов

  - адрес входа
  - признак неперемещаемости
- Сегмент констант.
- Сегмент команд.
- Сегмент инициализированных данных.
- Информация о настройке:
  - тип настройки M (абсолютное значение, относительно внешнего имени, относительно одного из сегментов, общий блок)
  - формат настройки X (словный адрес, короткоадресная команда, команда префиксации, адрес с префиксацией)
  - ссылка на таблицу символов
- Таблица символов.

Размеры всех сегментов и таблицы символов должны быть кратны длине слова (8 байт).

Объектный файл третьего режима отличается от объектного файла первого режима наличием сегмента массивов и дополнительных форматов настройки.

Данная структура объектного файла фактически сохраняет совместимость со структурой объектного файла ОС UNIX версии 7 для PDP-11 и версии 4.2BSD для VAX-11. Основные ее достоинства - простота и универсальность, наличие всех необходимых средств для построения различных компиляторов и символьных отладчиков.

Длину объектного файла можно уменьшить примерно на 30% за счет более рациональной структуры таблицы настройки.

4.4. Загрузчик.

Загрузчик, или редактор связей, - базовая системная компонента, предназначенная для компоновки нескольких объектных модулей в один и для подключения к нему необходимых подпрограмм из общих библиотек.

Загрузчик устроен по классической двухпроходной схеме.

Первый проход по объектным файлам и библиотекам, указанным в списке параметров:

1. Считывание таблицы имен, заполнение внутренней таблицы глобальных имен.
2. Вычисление длин сегментов и таблицы символов, а также адреса входа.
3. Составление таблицы ссылок на необходимые библиотеки.
4. Считывание и формирование в памяти сегмента констант.

Промежуточные действия:

1. Проход по внутренней таблице символов и выделение места под общие блоки.
2. Создание промежуточных файлов для хранения сегментов констант, команд, инициализированных данных и их таблиц настройки, а также таблицы локальных символов.
3. Запись заголовка в выходной файл.

Второй проход по объектным файлам и библиотекам с использованием таблицы ссылок на библиотеки, созданной на первом проходе:

1. Перепись локальных символов в выходной файл, настройка адресов глобальных символов.
2. Настройка сегмента констант и запись его в промежуточный файл.
3. Настройка и перепись сегментов команд и инициализированных данных в промежуточные файлы.

Заключительные операции:

1. Перепись сегментов констант, команд и инициализированных данных (с выравниванием, если необходимо, на границу листа) в выходной файл
2. Перепись таблиц настройки.
3. Перепись таблицы локальных символов.
4. Запись таблицы глобальных символов.
5. Если не было ошибок и неразрешенных внешних ссылок, выходной файл получает признак выполняемого.

Данный загрузчик имеет несколько особенностей:

1. Оптимизация сегмента констант. Одинаковые константы, используемые в различных объектных файлах, располагаются в одной ячейке памяти. В результате за счет часто используемых констант экономится память. При включении оптимизации констант длина сегмента констант уменьшается на больших программах в 5-6 раз.
2. Выравнивание начала сегментов команд и данных на границу листа. Это позволяет закрывать их на запись и использовать одну копию сегментов для всех процессов, выполняющих данный файл.
3. Задание базового адреса загрузки. По умолчанию программа настраивается для загрузки с адреса 0. В случае некоторых системных компонент, например, начального загрузчика, возникает необходимость располагать программу в высших адресах памяти. Загрузчику можно задать параметр - базовый адрес загрузки.

4.5. Структура библиотечного файла.

Библиотечные файлы являются универсальным средством хранения файлов в виде библиотек, хотя используются в основном для создания общих библиотек объектных модулей. Заметим, однако, что библиотечные файлы не могут служить в целях переноса информации с машины на машину, так как их формат связан с длиной слова и порядком байт в слове.

Библиотечный файл устроен следующим образом:

- одно слово - признак библиотечного файла
- последовательность (возможно, пустая) записей, каждая из которых содержит :
  - заголовок файла (имя, время последней модификации, идентификаторы владельца и группы, режим доступа, длина в байтах)
  - собственно файл длины, указанной в заголовке; дополняется пустым байтом до четной длины

Структура библиотечного файла аналогична (с точностью до представления данных) используемой в ОС UNIX 7 версии.

4.6. Библиотекарь.

Библиотекарь - системная компонента, предназначенная для работы с библиотечными файлами.

Основные функции:

1. Запись файлов в архив.
2. Считывание файлов из архива.
3. Удаление файлов из архива.
4. Выдача каталога архива.
5. Заведение нового архива.
6. Перемещение файлов внутри архива.

4.7. Другие программы для работы с объектными файлами.

Для работы с объектными файлами и библиотеками создано несколько утилит:

1. NM - выдача таблиц имен объектных или библиотечных файлов. Можно выдавать все, или только глобальные, или только неопределенные символы. выдачу можно сортировать по именам или по адресам, в прямом или обратном порядке. Для каждого символа выдается адрес и тип.
2. LORDER - построение графа зависимости объектных файлов. Выходом является список пар имен файлов, где первый файл зависит от второго (по определению один объектный файл зависит от второго, если он ссылается на внешний символ, определенный во втором файле). Выход команды *lorder* может быть обработан командой топологической сортировки *tsort* для получения оптимального порядка компоновки модулей загрузки.
3. SIZE - выдача размера объектного файла. Команда печатает в наглядном виде размеры сегментов и общий размер объектного файла. Данные выдаются в байтах или словах.
4. STRIP - удаление таблицы символов и таблицы настройки. Загрузчик оставляет в выполняемом файле таблицу символов на случай использования отладчика. Если программа уже отлажена, таблицу символов можно удалить командой *strip*.
5. SHOW - деассемблер объектного файла. Преобразует объектный файл в наглядное текстовое представление, близкое к языку ассемблера. Используется для отладки ассемблера и загрузчика.

5

Ассемблер.

Ассемблер - одна из важнейших составных частей системы программирования. Во-первых, поскольку трансляция языка Си в данной версии компилятора происходит в язык ассемблера, от удачного выбора ассемблера существенно зависит сложность переноса компилятора. Во-вторых, программы на Си после преобразования в ассемблер увеличиваются в 3-3.5 раза, следовательно, скорость ассемблирования сильно влияет на общую скорость компиляции. В-третьих, ассемблер должен быть достаточно простым, чтобы трудоемкость разработки ассемблера не оказалась больше трудоемкости переноса компилятора. Последнее требование не противоречит первому, так как для компилятора не нужны макро-возможности, условное ассемблирование, изощренные конструкции и т. п.

При проектировании языка ассемблера были приняты следующие решения:

1. язык максимально упрощен;
2. язык включает все конструкции, необходимые для компилятора;
3. сегментация объектного файла имеет непосредственное отражение в командах ассемблера;
4. особое внимание обращено на эффективность ассемблера по памяти и по времени трансляции;
5. адресные выражения имеют регулярную структуру и включают большое число разнообразных операций над адресами;
6. мнемоника команд преемственна широко распространенному на БЭСМ-6 и СВС автокоде МАДЛЕН;
7. базирование короткоадресных команд выполняется автоматически.

С точки зрения языка ассемблера программа делится на четыре сегмента (для Эльбрус-Б - пять сегментов): команд, инициализированных данных, текстовых строк, неинициализированных данных (для Эльбрус-Б добавляется сегмент массивов). Сегмент констант для программиста «невидим». Он формируется автоматически из литеральных констант, встречающихся в программе Производится слияние одинаковых литералов.

Сегмент команд, инициализированных данных и текстовых строк не отличаются друг от друга, за исключением выравнивания на границу слова. В сегменте команд выравни-

вание производится посредством пустой команды, в остальных сегментах - посредством нулевого полуслова.

Сегмент текстовых строк - это фактически второй сегмент инициализированных данных. Он используется компилятором для инициализации массивов указателей текстовыми строками. В конце ассемблирования сегмент текстовых строк дописывается в конец сегмента данных.

В сегментах неинициализированних данных и массивов нельзя размещать команды и данные. Здесь допускаются только метки и команды резервирования места под переменные.

В режиме БЭСМ короткоадресные команды, обращающиеся к литералам, базировать не нужно, так как сегмент констант всегда размещается в привилегированной области памяти (4 Кслова). Обращения короткоадресными командами к остальным сегментам всегда базируются командой *utc*.

В режиме Эльбрус-Б все сегменты, кроме сегмента массивов, лежат в привилегированной памяти (512 Кслов), поэтому базируются только обращения к этому сегменту.

Особое внимание в ассемблере обращено на эффективность по времени трансляции. Основная часть времени уходит на лексический анализ входного текста и поиск по таблицам. Тщательное кодирование лексического анализатора, генерация кода по однопроходной схеме и применение хеширования для всех внутренних таблиц позволили добиться высокой скорости. При работе на PC AT с тактовой частотой 10 Мгц средняя скорость трансляции составляет 500 ассемблерных операторов в секунду. При работе Си-компилятора на фазу ассемблирования тратится 20-25% общего времени трансляции.

6

Эмулятор.

Эмулятор системы команд БЭСМ-6 был разработан в целях отладки кросс-системы в рамках инструментальной ЭВМ.

Основными требованиями были:

1. достаточно точная эмуляция всех команд БЭСМ-6 в режиме пользователя;
2. наличие режимов трассировки различной степени подробности, пошагового выполнения;
3. наличие достаточно развитой системы директив для использования эмулятора в качестве адресного отладчика;
4. высокая мобильность эмулятора, работа на инструментальных ЭВМ с различной длиной слова и размером адресуемой памяти;
5. удовлетворительная эффективность по скорости работы.

Хотя оптимизация эмулятора по скорости работы и не проводилась, он работает достаточно быстро. Замедление на PC AT по сравнению с БЭСМ составляет около 200 раз.

При работе на инструментальных ЭВМ с большим адресным пространством вся память эмулируемой БЭСМ-6 размещается в оперативной памяти. Для машин с малым адресным пространством (меньше 256 Кбайт) разработан алгоритм страничной подкачки. Листы памяти БЭСМ-6, к которым дольше всего не было обращений, выталкиваются во временный файл, а при необходимости снова подгружаются. С использованием подкачки эмулятор вполне эффективно работает даже на адресном пространстве в 40 Кбайт.

Основная функция эмулятора - загрузка выполняемого файла в формате БЭСМ и выполнение. Возможна трассировка выполняемых команд, состояния регистров, подкачки листов. Посредством директив пошагового выполнения, просмотра и изменения содержимого памяти и регистров можно эффективно управлять работой программы.

7

Си-компилятор.

В качестве исходной версии для переноса компилятора Си на ЭВМ Эльбрус-Б был выбран портабильный компилятор Джонсона для PDP-11. Созданный в середине 70-х годов, он был в дальнейшем перенесен на ряд других машин. Входной язык компилятора Джонсона фактически является более мобильным, чем стандарт ANSI.

Компилятор разрабатывался с ориентацией переноса его на другие типы ЭВМ. Так, в нем применен таблично-управляемый метод генерации кода, машинно-зависимая часть четко отделена от машинно-независимой и составляет около 25%, произведена полная параметризация компилятора по машинно-зависимым величинам. Благодаря этому, трудоемкость переноса оказывается существенно ниже трудоемкости разработки нового компилятора «с нуля». Исходя из имеющегося опыта, затраты на перенос лежат в диапазоне от 3-х до 6 месяцев и зависят от сложности архитектуры целевой машины.

Компилятор работает по однопроходной схеме. В случае нехватки оперативной памяти может быть разбит на два прохода. Скорость трансляции на PC AT - около 60 строк.

В данной работе не описываются входящие в состав системы программирования препроцессор языка Си и монитор «cc». Они полностью машинно-независимы и при переносе никаких правок не требуют.

Три важнейших вопроса, возникающих при переносе компилятора, - это правильный выбор представления данных, распределения регистров, соглашения о межмодульных связях. От их решения в существенной степени зависят сложность реализации компилятора, совместимость его со стандартами языка, степень мобильности программ.

7.1. Представление данных.

В связи с отсутствием аппаратных средств для работы с целыми числами различной длины типы *short*, *int*, *long* эквивалентны и равны одному машинному слову.

Тип *char* эквивалентен *unsigned char*.

В первой версии компилятора с целью упрощения тип *double* эквивалентен *float*.

В дальнейшем будем рассматривать только типы *char*, *int*, *unsigned int*, *float*, указатель на словный объект и указатель на байт.

1. Тип *char* имеет длину 8 бит (1 байт). Одиночная переменная занимает целое слово и размещается в младших разрядах. Массивы упаковываются по 6 байт в слово (БЭСМ) или 8 байт в слово (Эльбрус-Б). Последовательные элементы массива располагаются в слове справа налево. Заметим, что это отличается от традиционного представления текстовых строк на БЭСМ-6. Однако этот способ позволяет повысить совместимость с имеющимся программным обеспечением.
2. Тип *int* занимает 1 слово. Мантисса и знак представляют собой собственно число, биты порядка равны 0. Таким образом, информационная длина типа составляет 41 бит для БЭСМ и 53 бита для Эльбрус-Б. Это представление целых чисел также отличается от традиционного для БЭСМ, где порядок обычно не равен 0. Однако наличие ненулевого порядка противоречит определению логических битовых операций, а также усложняет реализацию операции сдвига.
3. Тип *unsigned* занимает 1 слово, значащими являются все биты - 48 для БЭСМ и 64 для Эльбрус-Б.
4. Тип *float* имеет длину 1 слово, представление - в естественном виде.

Тип «указатель на словный объект» представляет собой 15-битный (БЭСМ) или 27-битный (Эльбрус-Б) словный адрес, занимающий 1 слово и расположенный в младших разрядах. Доступ по такому указателю возможен посредством команды косвенной адресации.

Тип «указатель на байт» отличается от типа «указатель на слово» тем, что в 19-21 битах (БЭСМ) или в 31-33 битах (Эльбрус-Б) находится номер байта в слове в диапазоне 0-5 (БЭСМ) или 0-7 (Эльбрус-Б). Младший байт имеет номер 0.

Заметим, что словный указатель есть частный случай байтового, или, что то же самое, преобразование типа (*int **) в (*char **) вырождено до тождественного. Этот факт очень важен, так как на машинах без аппаратной байтовой адресации часто большую проблему составляют случаи передачи в качестве параметра функции словного адреса вместо байтового.

7.2. Распределение регистров.

По наборам регистров первый и третий режимы команд Эльбруса-Б идентичны, поэтому распределение регистров для них одинаково.

Все номера регистров даны в десятичном виде.

Регистр 15 - указатель стека.

Регистр 14 - передача адреса возврата в вызываемую программу, а также как временный для некоторых операций.

Регистр 13 - указатель на область локальных переменных в стеке.

Регистр 12 - указатель на область аргументов функции в стеке.

Регистры 10, 11 - временные, используются для хранения промежуточных данных в процессе вычисления выражений.

Регистры 1-9 - для размещения регистровых переменных типа «словный указатель», а также как временные.

Номера регистров параметризованы и могут быть достаточно легко изменены.

7.3. Соглашение о связях.

Соглашение о межмодульных связях определяет порядок передачи параметров и адреса возврата, а также сохранения-восстановления регистров.

При передаче параметров вызывающая функция записывает в стек значение сумматора, затем - параметры в прямом порядке. На сумматоре передается целое число, равное размеру памяти (в словах), занятой параметрами, со знаком минус. Это необходимо для того, чтобы вызываемая программа смогла вычислить адрес первого параметра. Таким способом реализуются функции с переменным числом аргументов, например, *printf*.

Адрес возврата при вызове передается в 14 регистре.

Вызываемая программа обязана сохранить содержимое регистров 1-9, 12, 13.

При возврате вызываемая программа должна уменьшить содержимое 13 регистра на длину параметров.

7.4. Входной язык.

Лексический и синтаксический анализаторы, блок семантического анализа, блок построения дерева выражения, основная часть кодогенератора машинно-независимы, поэтому входной язык компилятора практически не меняется при переносе с машины на машину. Синтаксис языка получается полностью одинаковым. В семантике большая часть изменений приходится на преобразование типов.

Заметим,что в стандарте языка некоторые элементы считаются машинно-зависимыми, что упрощает их реализацию:

- сдвиг на отрицательное число и число разрядов, превышающее длину слова;
- наличие прерывания по делению на 0 и переполнению;
- порядок вычисления аргументов функции;
- порядок вычисления подвыражений;
- распространение знака при сдвиге;
- наличие знака у типа *char*;
- длины типов;
- порядок размещения полей в слове;
- наличие и максимальное количество регистровых переменных;
- способ представления указателей.

Значимая длина идентификатора в данной реализации может иметь длину 511 или 8 символов (задается при сборке компилятора). Длина внешнего идентификатора зависит от допустимой длины имени в ассемблере и загрузчике и равна 254 символам. В реальных программах идентификаторы длиной больше 40 символов практически не встречаются, но они могут оказаться полезными для символьного отладчика.

7.5. Общая схема компиляции.

Процесс компиляции от исходного текста до выполняемого файла делится на четыре этапа. Разбор флагов, создание промежуточных файлов, управление ходом компиляции производит монитор «cc».

1. Фаза препроцессора. В исходном файле обрабатываются директивы препроцессора *#include, #if, #ifdef, #ifndef, #else, #endif, #define, #undef, #line, #class, #ident,* раскрываются макроопределения.
2. Фаза компилятора. Обработанный препроцессором файл преобразуется в текст на языке ассемблера.
3. Фаза ассемблера. Полученный после компиляции ассемблерный файл преобразуется в объектный модуль.
4. Фаза загрузки. Скомпилированные объектные модули, а также необходимые библиотеки компонуются в выполняемый файл.

Время работы отдельных частей примерно следующее:

препроцессор - 2-5%
компилятор - 60-65%
ассемблер - 20-25%
загрузчик - 5-10%

7.6. Структура кодогенератора.

Кодогенератор четко отделен от остальных частей компилятора. При сборке в двухпроходном варианте он образует второй проход. От первого прохода ко второму через промежуточный файл передается:

- информация о началах блоков программы;
- информация о концах блоков;
- деревья выражений;
- ассемблерные вставки и фрагменты готовой программы.

По внутренней структуре кодогенератор можно разбить на несколько отдельных блоков:

1. блок чтения промежуточного файла (используется при сборке двухпроходной версии);
2. блок декомпозиции дерева выражения;
3. блок поиска минимального поддерева;
4. блок генерации кода для минимального поддерева;
5. блок генерации кода для условных операций;
6. блок распределения регистров;
7. блок поиска в таблице шаблонов;
8. блок таблицы шаблонов для генерации кодов операций;
9. блок вычисления чисел Сети-Ульмана;
10. блок управления стратегией генерации кода;
11. блок генерации ассемблерного кода.

Первые семь из перечисленных блоков машинно-независимы.

Заметим, что генерация отдельных частей ассемблерного кода происходит еще на ранних этапах синтаксического анализа. Так порождаются метки, заголовки и окончания функций, инициализация переменных.

В общих чертах генерация кода происходит следующим образом:

1. Чтение дерева из промежуточного файла - в случае двухпроходной версии. Если компилятор собран в однопроходном виде, кодогенератор получает дерево выражения в виде параметра.
2. Декомпозиция дерева по операциям «запятая», постинкремент, постдекремент.
3. Канонизация дерева. Включает в себя некоторые машинно-зависимые преобразования адресных узлов и вычисление чисел Сети-Ульмана.
4. Поиск минимального поддерева и генерация кода. Замена поддерева узлом, отражающим местонахождение результата. Минимальным называется поддерево, которое, согласно оценке Сети-Ульмана, может быть вычислено на имеющемся наборе регистров, без обращения к временным ячейкам памяти.
5. Пункты 3, 4 повторяются в цикле, пока все дерево не будет вычислено и свернуто в один узел.

Основой генерации кода для минимального поддерева является поиск в таблице шаблонов кодов операций. Если операция найдена в таблице, ее левый и правый операнды удовлетворяют шаблону и имеются требуемые ресурсы, происходит генерация кода, соответствующего данной операции. Если шаблон не найден, делается попытка преобразования узла к более простому виду и снова поиск в таблице. В конце концов шаблон должен быть найден. Если кодогенератор заходит в тупик, он сообщает об ошибке компилятора. Это означает, что таблица шаблонов неполна, или в машинно-зависимых программах управления генерацией кода есть противоречия.

7.7. Методы оптимизации.

В данной реализации набор методов оптимизации ограничен следующими методами:

- вычисление константных подвыражений;
- исключение деления и умножения на 1;
- преобразование умножения на степень двойки в сдвиг;
- исключение последовательности операций «*&»;
- минимизация количества используемых регистров и временных ячеек памяти.

Поиск общих подвыражений не производится.

8

Выводы.

1. Реализация Си-компилятора на базе портабильного компилятора Джонсона возможна для ЭВМ различных архитектур. Компилятор обладает всеми необходимыми элементами для настройки на конкретный тип машины. Вместе с тем эта работа не может быть проделана автоматически, по формализованному описанию архитектуры ЭВМ, так как кроме таблиц и алгоритмов с четкой и ясной структурой присутствует также большое число неформализуемых эвристик. Хотя перенос компилятора возможен за относительно короткое время и значительная часть работы делается легко, некоторые элементы кодогенератора требуют больших усилий в изобретении нетривиальных алгоритмов и эвристик.
2. К языку ассемблера особых требований не предъявляется. Необходимо лишь наличие секций и некоторые соглашения об инициализации и внешних именах.
3. Компилятор и остальные компоненты кросс-системы достаточно мобильны и к настоящему времени работают на разнообразных типах инструментальных машин.
4. Процент изменений и примерное количество измененных строк в текстах при изготовлении системы распределились по компонентам следующим образом:

   компилятор - 20% - примерно 3000 строк
   ассемблер - 100% - 1500 строк
   эмулятор - 100% - 1500 строк
   загрузчик - 70% - 1000 строк
   препроцессор - 0%
   остальные компоненты - 40% - 1000 строк
   библиотека на счете - 100% - 200 строк
   файлы заголовков - 20% - 200 строк
   стандартная библиотека - 2% - 200 строк

9

Литература.

1. B. W. Kernighan and D. M Ritchie, The C Programming Language, Prentice-Hall, Englewood Cliffs, New Jersey (1978).
2. Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. - М.: Мир, 1978.
3. D. M. Ritchie, «A Tour through the UNIX C Compiler», UNIX System Programmer's Manual, Volume 2, AT&T Bell Laboratories. Published by Holt, Rinehart, and Winston, New York, 1983, 1979. Pages 529-543.
4. S. C. Johnson, «A Tour through the Portable C Compiler», UNIX System Programmer's Manual, Volume 2, AT&T Bell Laboratories. Published by Holt, Rinehart, and Winston, New York, 1983, 1979. Pages 544-568.
5. Волков А. И., Автокод мадлен. (Описание транслятора). - Дубна: Изд. ОИЯИ, 11-5427, 1970.
6. Лобачев Ю. В., Основич А. Л., Сердюк Г. И. Реализация ОС ДЕМОС на ЭВМ семейства БЭСМ. - В кн.: Мобильное программное обеспечение. Калинин: Изд. НПО Центрпрограммсистем, 1988.
7. Вакуленко С. В., Пасынков И. Г., Рыжков С. И. Реализация компиляторов с языка Си для ЭВМ линии БЭСМ-6. - В кн.: Мобильное программное обеспечение. Калинин: Изд. НПО Центрпрограммсистем, 1988.

Приложения

A.OUT(5)

ИМЯ

a.out - формат объектного файла.

ФОРМАТ

```
# include <a.out.h>
```

ОПИСАНИЕ

Объектный файл (по умолчанию он получает имя a.out) является результатом работы ассемблера as(1) и редактора связей ld(1). Редактор связей делает объектный файл выполняемым, если не было ошибок и неразрешенных внешних ссылок.

```
/*
 * Структура файла a.out для БЭСМ-6:
 *
 * Заголовок из 8 слов:
 *                  магическое число 407, 410, 411
 *                  длина сегмента const   )
 *                  длина сегмента text    )
 *                  длина сегмента data    ) в байтах, кратно 6
 *                  длина сегмента bss     )
 *                  длина таблицы символов )
 *                  адрес входа
 *                  признак неперемещаемости
 *
 * заголовок         0
 * const:            48
 * text:             48 + constsize
 * data:             48 + constsize + textsize
 * настройка const:  48 + constsize + textsize + datasize
 * настройка text:   48 + 2*constsize + textsize + datasize
 * настройка data:   48 + 2*constsize + 2*textsize + datasize
 * таблица символов: 48 + 2*constsize + 2*textsize + 2*datasize
 */

      /* заголовок файла a.out */
struct exec {
  int     a_magic;
  long    a_const;
  long    a_text;
  long    a_data;
  long    a_bss;
  long    a_syms;
  short   a_entry;
  short   a_flag;
};

      /* элемент таблицы символов */
struct nlist {
  short   n_len;          /* длина имени в байтах */
  short   n_type;         /* тип */
  short   n_value;        /* значение */
```

```
  char *  n_name;         /* указатель на имя */
};

      /* бит неперемещаемости */
# define RELFLG 1

      /* длина заголовка в байтах */
# define HDRSZ  48

      /* типы файлов */
# define FMAGIC 0407
# define NMAGIC 0410
# define AMAGIC 0411

      /* типы символов */
# define N_EXT  040
# define N_TYPE 037
# define N_FN   037
# define N_UNDF 00
# define N_ABS  01
# define N_CONST 02
# define N_TEXT 03
# define N_DATA 04
# define N_BSS  05
# define N_COMM 07      /* для ld */
# define N_STRNG 06     /* для as */

      /* типы настройки */
# define RABS   0
# define RCONST 02
# define RTEXT  04
# define RDATA  06
# define RBSS   010
# define REXT   016  /* одновременно является битовой маской */
# define RSHORT 01   /* одновременно является битовой маской */
# define RSTRNG 014  /* для as */

/* макросы, определяющие различные позиции в файле */

# define N_TXTOFF(x) HDRSZ
# define N_SYMOFF(x) (N_TXTOFF(x)+(x).a_const+(x).a_text+(x).a_data)
# define N_STROFF(x) (N_SYMOFF(x)+(x).a_syms)

# define BADMAG(x) ((x).a_magic != FMAGIC && (x).a_magic != NMAGIC)
# define N_BADMAG  BADMAG
# define FORMAT    "%05o" /* для печати значения символа */
# define N_FORMAT  FORMAT
```

```
/*
 * Структура файла a.out для Эльбрус-Б:
 *
 * Заголовок из 9 слов:
 *                  магическое число 407, 410, 411
 *                  длина сегмента const   )
```

```
 *                  длина сегмента text    )
 *                  длина сегмента data    ) в байтах, кратно 8
 *                  длина сегмента bss     )
 *                  длина сегмента abss    )
 *                  длина таблицы символов )
 *                  адрес входа
 *                  признак неперемещаемости
 *
 * заголовок         0
 * const:            72
 * text:             72 + constsize
 * data:             72 + constsize + textsize
 * настройка const:  72 + constsize + textsize + datasize
 * настройка text:   72 + 2*constsize + textsize + datasize
 * настройка data:   72 + 2*constsize + 2*textsize + datasize
 * таблица символов: 72 + 2*constsize + 2*textsize + 2*datasize
 */

      /* заголовок файла a.out */
struct exec {
  int     a_magic;
  long    a_const;
  long    a_text;
  long    a_data;
  long    a_bss;
  long    a_abss;
  long    a_syms;
  long    a_entry;
  short   a_flag;
};

      /* элемент таблицы символов */
struct nlist {
  short   n_len;          /* длина имени в байтах */
  short   n_type;         /* тип */
  long    n_value;        /* значение */
  char *  n_name;         /* указатель на имя */
};

      /* бит неперемещаемости */
# define RELFLG 1

      /* длина заголовка в байтах */
# define HDRSZ  72

      /* типы файлов */
# define FMAGIC 0407
# define NMAGIC 0410
# define AMAGIC 0411

      /* типы символов */
# define N_EXT  040
# define N_TYPE 037     /* маска */
# define N_FN   037
# define N_UNDF 00
```

```
# define N_ABS  01
# define N_CONST 02
# define N_TEXT 03
# define N_DATA 04
# define N_BSS  05
# define N_ABSS 06
# define N_STRNG 07     /* для as */
# define N_COMM 010
# define N_ACOMM 011

      /* типы настройки */
# define RABS   0
# define RCONST 010
# define RTEXT  020
# define RDATA  030
# define RBSS   040
# define RABSS  050
# define RSTRNG 060 /* для as */
# define REXT   070 /* одновременно является битовой маской */

# define RSHIFT 04
# define RTRUNC 05
# define RLONG  06
# define RSHORT 07 /* одновременно является битовой маской */

# define RGETIX(h) (h>>6)
# define RPUTIX(h) ((long)h<<6)

/* макросы, определяющие различные позиции в файле */

# define N_TXTOFF(x) HDRSZ
# define N_SYMOFF(x) (N_TXTOFF(x)+(x).a_const+(x).a_text+(x).a_data)
# define N_STROFF(x) (N_SYMOFF(x)+(x).a_syms)

# define BADMAG(x) ((x).a_magic != FMAGIC && (x).a_magic != NMAGIC)
# define N_BADMAG  BADMAG
# define FORMAT    "%09o"  /* для печати значения символа */
# define N_FORMAT  FORMAT
```

Файл состоит заголовка, разделов констант, программы и данных, таблицы размещений и таблицы имен. Последние два раздела могут отсутствовать, если файл был сформирован редактором связей ld с ключом `-s` или обработан командой strip(1).

В заголовке размеры каждого раздела задаются в байтах, но всегда имеют значения, кратные длине слова. Размер заголовка не входит в размер ни одного из сегментов программы.

При загрузке программы из объектного файла в память, сегмент констант располагается с логического адреса 010 для БЭСМ или 011 для Эльбрус-Б. Остальные сегменты размещаются в зависимости от типа файла:

0407 Сегмент команд не закрывается по записи и не будет разделятся между процессами, поэтому данные идут непосредственно за текстом.

0410 Сегмент данных начинается с границы листа, следующего за командным сегментом. Последний закрывается на запись и будет использоваться всеми процессами, выполняющими данную программу.

0411 Сегмент команд начинается с границы листа, следующего за сегментом констант.

Сегмент стека размещается в самом конце логического пространства процесса. Если необходимо, сегмент стека автоматически расширяется. Сегмент данных изменяется только с помощью системного вызова brk (2).

Для БЭСМ: если имя имеет тип - «неопределенное внешнее», и поле n_value ненулевое, оно воспринимается редактором связей, как имя common-области (общей области), размер которой определяется значением n_value. Для Эльбрус-Б: имя common-области имеет тип N_COMM или N_ACOMM.

Значение полуслова в образе текста или данных (если только это не ссылка на неопределенный внешний объект) соответствует значению при загрузке файла в память. Если в таблице размещений для этого полуслова указано, что оно является внешней неопределенной ссылкой, то его значение рассматривается как сдвиг от указанного внешнего имени. Если при обработке файла редактором связей неопределенный символ обнаруживается в другом модуле, его адрес добавляется к этому полуслову.

Если информация о размещениях не уничтожена, то значения n_value полностью соответствуют адресам объектов в памяти. Если таблица размещения удалена, бит RELFLG поля a_flag не равен нулю.

Биты REXT таблицы перемещений указывают сегмент, к которому относится соответствующее слово текста или данных:

RABS абсолютное значение
RCONST сегмент констант
RTEXT сегмент текста
RDATA инициализированные данные
RBSS неинициализированные данные (bss)
RABSS массивы (для Эльбрус-Б)
REXT неопределенное внешнее

Биты RSHORT определяют формат настраиваемого полуслова:

RSHIFT левая часть адреса (для Эльбрус-Б)
RTRUNC правая часть адреса (для Эльбрус-Б)
RLONG длинноадресная команда (для Эльбрус-Б)
RSHORT короткоадресная команда (для Эльбрус-Б)

Остаток слова размещения в случае внешней ссылки содержит номер символа, в противном случае не используется. Первый символ имеет номер 0, второй - 1 и т.д.

ДОПОЛНИТЕЛЬНЫЕ ССЫЛКИ

as(1), ld(1), nm(1)

AR(5)

ИМЯ

ar - формат архивного (библиотечного) файла

ФОРМАТ

```
# include <ar.h>
```

ОПИСАНИЕ

Команда ar(1) используется для объединения нескольких файлов в один архивный файл. Архивный файл используется, в основном, в качестве библиотеки для редактора связей ld.

Файл, созданный командой ar начинается со слова, содержащего «магический» признак ARMAG, за которым следуют все составляющие архив файлы. Каждый файл начинается с заголовка:

```
# define ARMAG          0177545
# define ARHDRSZ        48

struct ar_hdr {
  char  ar_name [14];
  long  ar_date;
  int ar_uid;
  int ar_gid;
  int ar_mode;
  long  ar_size;
};
```

Имя дополняется символом «пусто». Время последнего изменения файла на момент записи в архив заносится в ar_date.

Каждый файл начинается с четной границы; если необходимо, между файлами вставляется дополнительный символ. Однако размер отражает фактическую длину файла, без учета вставки.

Следует обратить внимание, что в архивном файле не предусмотрены пустые участки.

ДОПОЛНИТЕЛЬНЫЕ ССЫЛКИ

ar(1), ld(1), nm(1)

LD(1)

ИМЯ

ld - редактор связей

ФОРМАТ

```
ld [ключ] файл ...
```

ОПИСАНИЕ

Команда ld объединяет несколько объектных программ в одну, разрешает внешние ссылки и производит поиск в библиотеках. В простейшем случае задаются несколько объектных файлов, и ld объединяет их, создавая объектный модуль, который может либо выполняться, либо являться входным для последующих запусков ld (в последнем случае должен задаваться ключ «-r» для сохранения таблицы перемещения). Результат работы ld помещается в файл с именем a.out. Этот файл делается выполняемым, если в процессе загрузки не было ошибок.

Указанные параметрами программы объединяются в заданной последовательности. Точкой входа для выходного модуля является начало первой программы (если не используется ключ «-e»).

Если какой-либо из параметров представляет собой библиотеку, эта библиотека просматривается только один раз в тот момент, когда она встречается в списке параметров. Загружаются только те программы, которые определены как неразрешенные внешние ссылки. Если подпрограмма из библиотеки ссылается на другую подпрограмму из той же библиотеки, то последняя должна находится в библиотеке после подпрограммы, которая на нее ссылается.

Символы «_econst», «_etext», «_edata», «_ebss» (для Эльбрус-Б) и «_end» («econst», «etext», «edata» и «end» в языке Си) зарезервированы и, если на них имеются ссылки, устанавливаются на первую ячейку над константами, первую ячейку над программой, первую ячейку над инициализированными данными, первую ячейку над неинициализированными данными и первую ячейку над всеми данными соответственно. Попытка переопределить эти символы приводит к ошибке.

Команда ld распознает несколько ключей. За исключением ключа «-l», все они должны находиться перед именами файлов:

-lX Этот ключ является сокращением имени библиотеки /lib/libX.a, где «X» - строка. Если она не существует, команда ld пытается отыскать библиотеку /usr/lib/libX.a. Поскольку осуществляется поиск, местонахождение «-l» является существенным.

-o Параметр «имя» после ключа «-o» используется в качестве имени выходного файла ld вместо «a.out».

-r Генерирует биты перемещения в выходном файле, так что он может участвовать в последующем прогоне ld. Этот флаг предотвращает также окончательное определение общих символов и подавляет диагностические сообщения относительно неопределенных символов.

-x Не записывать в выходной файл информацию о локальных символах.

-X Не записывать в выходной файл информацию о локальных символах, начинающихся с буквы «L» (такой вид имеют внутренние локальные метки, генерируемые компилятором).

-s Удаляет из результирующего файла таблицу символов и биты перемещения с целью экономии места (ценой снижения полезности отладчиков). Эта информация может быть удалена также с помощью команды strip(1).

-S Удалить все символы, кроме локальных и глобальных меток.

-u Параметр «имя» после ключа «-u» заносится в таблицу символов в качестве неопределенного внешнего имени.

-e Параметр «имя» после ключа «-e» становится точкой входа в программу.

-D Параметр «число» после ключа «-D» определяет размер сегмента данных.

-T Параметр «число» после ключа «-T» задает начальный адрес размещения программы в памяти (базовый адрес). По умолчанию программа располагается с адреса 0. Используется при создании автономных системных программ, например, начального загрузчика.

-n Создать программу с разделяемым сегментом констант и данных. Начало сегмента данных выравнивается на границу листа.

-d Выделить место под common-блоки (при включенном флаге -r).

-k Выровнять начало сегмента команд на границу листа.

ФАЙЛЫ

/lib/lib*.a библиотеки

/usr/lib/lib*.a дополнительные библиотеки

a.out выходной файл

ДОПОЛНИТЕЛЬНЫЕ ССЫЛКИ

as(1), ar(1), cc(1), size(1), a.out(5)

AR(1)

ИМЯ

ar - обслуживание архивов и библиотек.

ФОРМАТ

```
ar ключ [позиционное-имя] архив имя ...
```

ОПИСАНИЕ

Команда ar обслуживает группы файлов, объединенных в единый архивный файл. Основным применением этой программы является создание и модификация библиотечных файлов, используемых редактором связей ld. Тем не менее, она может быть использована в любых подобных целях.

«Ключ» представляет собой один из символов «drqtpmx», который может объединяться с одним или несколькими из символов «vuaibcl» (последние могут быть опущены). Аргумент «архив» задает имя архивного файла. Аргумент «имя» представляет собой имя файла, который необходимо занести в архив или имя файла, уже входящего в состав архива. Ключи задают следующие режимы работы:

d Удаление файлов с указанными именами из архива.

r Запись перечисленных файлов в архив. Если файл уже входит в архив, то он заменяется на новый. Если вместе с ключом «r» используется необязательный ключ «u», то в архиве заменяются только те файлы, которые имеют более поздние даты модификации, чем одноименные файлы, входящие в архив. Если используется один из ключей позиционирования «abi», и присутствует параметр «позиционное-имя» новые файлы помещаются после (a) или перед (b или i) файлом «позиционное-имя». В противном случае новые файлы помещаются в конец.

q Быстрое добавление перечисленных файлов в конец архива. Ключи позиционирования не допускаются. В этом режиме не проверяется, находятся ли уже в архиве файлы с заданными именами. Этот режим полезен только при создании больших архивов по частям для устранения лишних действий.

t Получение имен файлов, входящих в архив. Если имена файлов не заданы, выдается список имен всех файлов архива, в противном случае, выводится список имен только заданных файлов. Если файл, с заданным именем, отсутствует в архиве, выдается соответствующая диагностика.

p Выдача в стандартный файл вывода перечисленных файлов из архива. Содержимое архива не изменяется.

m Перемещение перечисленных файлов. Если присутствует ключ позиционирования, и параметр «позиционное-имя», файлы перемещаются в указанное место, как и в режиме «r», иначе файлы перемещаются в конец архива.

x Перечисленные файлы переписываются из архива в отдельные файлы. Если имена файлов не заданы, распаковывается весь архив. При использовании ключа «x» архивный файл не изменяется. Файл считанный из архива получает дату модификации соответствующую времени выборки. Если одновременно с «x» задан ключ «o» дата модификации соответствует дате последней модификации файла.

v Если указан данный ключ, команда ar выдает подробный протокол своих действий. При использовании совместно с ключом «t», выдается более подробная информация о файлах, входящих в архив. Когда данный ключ используется совместно с ключом «p», перед каждым файлом указывается его имя.

c Создает новый архив, даже если архивный файл «архив» уже существует.

l Обычно команда ar помещает свои временные файлы в каталоге /tmp. Задание данного ключа приводит к тому, что они размещаются в текущем каталоге.

ФАЙЛЫ

/tmp/v* временные файлы

ДОПОЛНИТЕЛЬНЫЕ ССЫЛКИ

ld(1), lorder(1), ar(5)

ЗАМЕЧАНИЯ

Если в списке параметров один и тот же файл указан дважды, он может быть дважды помещен в архив.

Ключ «o» игнорируется, если пользователь не является владельцем файла или привилегированным пользователем.

NM(1)

ИМЯ

nm - выдать список имен.

ФОРМАТ

```
nm [-gnopru] [файл ...]
```

ОПИСАНИЕ

Команда nm распечатывает список имен (таблицу символов) каждого из указанных в списке параметров объектных файлов. Если параметр представляет собой архив, будет распечатываться таблица символов для каждого объектного файла архива. Если «файл» не указан, распечатываются символы из файла «a.out».

Каждому символическому имени предшествует его значение (пробелы, если оно не определено) и одна из букв U (неопределенное), A (абсолютное), F (имя файла), T (символ текстового сегмента), L (символ сегмента констант), D (символ сегмента данных), B (символ сегмента bss), Y (символ сегмента abss), C (общий блок сегмента bss) или M (общий блок сегмента abss). Если символ является локальным (не внешним), эта буква печатается на нижнем регистре. Вывод сортируется по алфавиту.

Можно использовать следующие ключи:

-g Выдавать только глобальные (внешние) символы.
-n Отсортировать по значениям, а не по алфавиту.
-o Указывать имя файла или элемента архива в каждой строке вывода, а не только один раз.
-p Не сортировать, печатать в порядке таблицы символов.
-r Отсортировать в обратном порядке.
-u Печатать только неопределенные символы.

ДОПОЛНИТЕЛЬНЫЕ ССЫЛКИ

ar(1), ar(5), a.out(5)

LORDER(1)

ИМЯ

lorder - определить связи объектных файлов.

ФОРМАТ

```
lorder файл ...
```

ОПИСАНИЕ

Входная информация представляет собой один или несколько объектных библиотечных или архивных (см. ar(1)) файлов. В результате работы получается список пар имен объектных файлов. Первый файл пары ссылается на внешний идентификатор, определенный во втором. Выход может быть обработан командой tsort(1) для нахождения такого порядка размещения модулей в библиотеке, который обеспечивает однопроходный поиск для ld(1).

Следующая строка предназначена для построения новой библиотеки из существующих файлов «.o»:

```
ar cr library `lorder *.o | tsort`
```

ДОПОЛНИТЕЛЬНЫЕ ССЫЛКИ

tsort(1), ld(1), ar(1)

ЗАМЕЧАНИЯ

Имена объектных файлов как внутри, так и вне библиотек, должны иметь окончание «.o» - в противном случае результат будет бессмысленным.

SIZE(1)

ИМЯ

size - сообщить размер объектного файла

ФОРМАТ

```
size [-w] [файл...]
```

ОПИСАНИЕ

Команда size печатает (десятичное) число байтов, занимаемое текстовым сегментом, сегментом данных и bss, а также их сумму в десятичном и восьмеричном коде для каждого объектного «файла», указанного в параметрах. Если имя файла не задано, используется файл a.out. Если задан флаг «-w», все данные выдаются в словах.

STRIP(1)

ИМЯ

strip - удалить таблицу символов.

ФОРМАТ

```
strip имя ...
```

ОПИСАНИЕ

Команда strip удаляет таблицу символов и биты перемещения, которые обычно содержатся в выводных файлах ассемблера и редактора связей. Используется для сокращения объема памяти для хранения выполняемого кода отлаженной программы.

Действие команды strip аналогично использованию ключа «-s» в команде ld.

ФАЙЛЫ

/tmp/s?????? временный файл.

ДОПОЛНИТЕЛЬНЫЕ ССЫЛКИ

ld(1)

SHOW(1)

ИМЯ

show - деассемблер объектного файла.

ФОРМАТ

```
show [-rR] [файл ...]
```

ОПИСАНИЕ

Команда show распечатывает объектный файл в формате, близком к ассемблерному. Если «файл» не указан, распечатывается файл «a.out».

Можно использовать следующие ключи:

-r Для каждого полуслова из сегментов констант, команд и данных выдавать также информацию о настройке.

-R Информацию о настройке печатать в восьмеричном виде.

По умолчанию информация о настройке выдается условным обозначением. Для 1-го режима команд (БЭСМ-6) оно состоит из буквы, определяющей тип настройки, и, возможно, числа, означающего номер внешнего символа в таблице имен.

c настройка относительно сегмента констант
t настройка относительно сегмента команд
d настройка относительно сегмента данных
b настройка относительно сегмента неинициализированных данных
a абсолютное значение
e настройка относительно внешнего имени

Если настраиваемое полуслово является короткоадресной командой, буква выводится на верхнем регистре.

Условное обозначение типа настройки для 3-го режима команд (Эльбрус-Б) состоит из буквы или числа, определяющих тип настройки, и буквы, означающей формат настраиваемой информации. Тип настройки:

c настройка относительно сегмента констант
t настройка относительно сегмента команд
d настройка относительно сегмента данных
b настройка относительно сегмента неинициализированных данных
y настройка относительно сегмента массивов
a абсолютное значение
число настройка относительно внешнего имени, число - индекс имени в таблице

Формат настройки:

l длинноадресная команда
s короткоадресная команда
h старшая часть адреса
t младшая часть адреса
a абсолютный адрес

ДОПОЛНИТЕЛЬНЫЕ ССЫЛКИ

ar(1), ar(5), a.out(5), nm(1)

AS(1)

ИМЯ

as - ассемблер

ФОРМАТ

```
as [-u] [-x] [-X] [-o обфайл] [файл]
```

ОПИСАНИЕ

Команда as ассемблирует файл с указанным именем или стандартный файл ввода, если имя файла не задано. Результат помещается в файл «a.out». По умолчанию все неопределенные имена считаются внешними. Можно задавать следующие флаги:

-o параметр, следующий за флагом «-o», задает имя выходного файла вместо «a.out»

-u не считать неопределенные имена внешними

-x не записывать в выходной файл информацию о локальных символах

-X не записывать в выходной файл информацию о локальных символах, начинающихся с буквы „L“.

ФАЙЛЫ

a.out объектный файл

ДОПОЛНИТЕЛЬНЫЕ ССЫЛКИ

ld(1), nm(1), adb(1), a.out(5)

EMU(1)

ИМЯ

emu - эмулятор системы команд Эльбрус-Б (1-й режим)

ФОРМАТ

```
emu [-b] [-v] [-t] [-s] [файл]
```

ОПИСАНИЕ

Команда emu загружает и выполняет объектный файл в формате Эльбрус-Б (1-й режим), эмулируя отдельные команды. Предоставляется возможность с помощью простого встроенного адресного отладчика останавливать программу во время выполнения, смотреть и изменять содержимое памяти и регистров, производить пошаговое (покомандное) выполнение программы, включать/выключать режимы трассировки. Если имя файла не задано, используется файл «a.out».

При работе эмулятора производится страничная (листовая) подкачка оперативной памяти эмулируемого процесса.

Допустимые флаги:

-b перед выполнением первой команды перейти в режим отладчика
-v включить режим выдачи подробной информации о состоянии процессора перед запросом очередной команды отладчика
-t увеличить режим трассировки (допускаются комбинации -t, -tt, -ttt)
-s выдавать подробную информацию о подкачке страниц

Система команд отладчика:

q выход из эмулятора
h выдача списка команд
f выдачи информации о текущем выполняемом файле
g продолжение выполнения с прерванного места
g <восьмеричное-число> продолжение выполнения с указанного места
s выполнить очередную команду
s <восьмеричное-число> выполнить одну команду с указанного места
w выдать информацию о текущем состоянии процессора
t выдать информацию о режиме трассировки
t <восьмеричная-цифра> установить уровень трассировки
v сменить режим выдачи подробной информации
a - выдать содержимое сумматора
y - выдать содержимое регистра младших разрядов
r - выдать содержимое регистра режимов АУ
c - выдать содержимое регистра-модификатора
j - выдать содержимое регистра адреса следующей команды
i <восьмеричное-число> - выдать содержимое указанного регистра
rtag - выдать содержимое регистра признака правой команды
<восьмеричное-число> - выдать содержимое ячейки памяти по указанному адресу
a = <слово> присвоить значение сумматору

y = <слово> присвоить значение регистру младших разрядов

r = <восьмеричное-число> присвоить значение регистру режимов АУ

c = <восьмеричное-число> присвоить значение регистру-модификатору

j = <восьмеричное-число> присвоить значение регистру адреса следующей команды

i <восьмеричное-число> = <восьмеричное-число> присвоить значение указанному регистру

rtag = <восьмеричное-число> присвоить значение регистру признака правой команды

<восьмеричное-число> = <слово> записать значение в ячейку памяти по указанному адресу

x <команда> выполнить указанную команду

ДОПОЛНИТЕЛЬНЫЕ ССЫЛКИ

as(1), ar(1), cc(1), size(1), a.out(5)

Описание входного языка мобильного компилятора Джонсона

Компилятор получает на входе текст программы, прошедшей обработку препроцессором. Кроме текста собственно программы на языке Си во входном потоке могут содержаться строки вида

```
# <число> <имя-файла>
```

причем символ «#» должен стоять в первой позиции. Посредством таких конструкций препроцессор передает компилятору информацию о том, к какому исходному файлу и какой строке относится очередной фрагмент текста.

При описании синтаксиса используются следующие обозначения:

имя мета-понятие
"строка" строка исходного текста
"\"" символ «двойная кавычка»
| разделение альтернатив
() группирование
[] возможное отсутствие элемента
? возможное отсутствие предшествующего элемента
***** повторение предшествующего элемента 0 и более раз
+ повторение предшествующего элемента 1 и более раз
... диапазон дискретных значений

Используемое понятие ASM-СИМВОЛ обозначает любой символ кроме двойной кавычки и конца строки.

```
файл =
    (внешнее-определение | ассемблерная-вставка) *
ассемблерная-вставка =
    "asm" "(" "\"" ASM-СИМВОЛ * "\"" ")" ";"
внешнее-определение =
    [атрибуты] ([список-внешних-описаний] ";" |
    описание-функции тело-функции)
атрибуты =
    класс тип | тип класс | класс | тип | класс тип класс
класс =
    "auto" | "extern" | "fortran" | "register" | "static" | "typedef"
тип =
    простой-тип | простой-тип простой-тип |
    простой-тип простой-тип простой-тип |
    определение-структуры | определение-перечисления
простой-тип =
    "char" | "double" | "float" | "int" | "long" | "short" |
    "unsigned" | "void"
определение-структуры =
    структура имя | структура [имя] "{" список-определений-типа [";"] "}"
структура =
    "struct" | "union"
список-определений-типа =
    определение-типа (";" определение-типа) *
```

```
определение-типа =
    тип [список-описаний]
список-описаний =
    описание ("," описание) *
описание =
    описание-функции |
    описание-данных [":" константное-выражение] |
    ":" константное-выражение
описание-данных =
    "*" описание-данных |
    описание-данных "(" ")" |
    описание-данных "[" "]" |
    описание-данных "[" константное-выражение "]" |
    имя |
    "(" описание-данных ")"
описание-функции =
    "*" описание-функции |
    описание-функции "(" ")" |
    описание-функции "[" "]" |
    описание-функции "[" константное-выражение "]" |
    "(" описание-функции ")" |
    имя "(" список-имен ")" |
    имя "(" ")"
список-внешних-описаний =
    внешнее-описание ("," внешнее-описание) *
внешнее-описание =
    описание-данных | описание-функции
определение-перечисления =
    "enum" [имя] "{" список-элементов-перечисления [","] "}" | "enum" имя
список-элементов-перечисления =
    элемент-перечисления ("," элемент-перечисления) *
элемент-перечисления =
    имя ["=" константное-выражение]
константное-выражение =
    выражение
тело-функции =
    описание-аргументов * составной-оператор
описание-аргументов =
    атрибуты [список-описаний] ";"
составной-оператор =
    "{" (атрибуты [список-внешних-описаний] ";") * оператор * "}"
оператор =
    выражение ";" |
    ассемблерная-вставка |
    составной-оператор |
    "if" "(" выражение ")" оператор |
    "if" "(" выражение ")" оператор "else" оператор |
    "while" "(" выражение ")" оператор |
    "do" оператор "while" "(" выражение ")" ";" |
    "for" "(" выражение ? ";" выражение ? ";" выражение ? ")" оператор |
    "switch" "(" выражение ")" оператор |
    "break" ";" |
    "continue" ";" |
    "return" выражение ? ";" |
    "goto" имя ";" |
```

```
    ";" |
    метка оператор
метка =
    имя ":" | "case" выражение ":" | "default" ":"
выражение =
    выражение "?" выражение ":" выражение |
    выражение бинарная-операция выражение |
    терм
бинарная-операция =
    "<=" | ">=" | "<" | ">" | "," | "/" | "%" | "+" | "-" | "<<" |
    ">>" | "*" | "==" | "!=" | "&" | "|" | "^" | "&&" | "||" |
    "*=" | "/=" | "%=" | "+=" | "-=" | ">>=" | "<<=" | "&=" |
    "|=" | "^=" | "=" | "=-" | "=*" | "=&" | "=+" | "=/" | "=%" |
    "=|" | "=^" | "=<" | "=>"
терм =
    терм ("++" | "--") | ("++" | "--") терм | "*" терм | "&" терм |
    "-" терм | "!" терм | "^" терм | "sizeof" терм |
    "(" тип [пустое-описание] ")" терм |
    "sizeof" "(" тип [пустое-описание] ")" |
    терм "[" выражение "]" |
    терм "(" ")" |
    терм "(" выражение ("," выражение) * ")" |
    терм "->" имя |
    терм "." имя |
    имя |
    целое |
    вещественное |
    строка |
    "(" выражение ")"
пустое-описание =
    "(" [пустое-описание] ")" "(" ")" |
    "*" пустое-описание |
    пустое-описание "[" [константное-выражение] "]" |
    "(" [пустое-описание] ")"
имя =
    буква (буква | цифра) *
буква =
    "_" | "A"..."Z" | "a"..."z"
цифра =
    "0"..."9"
целое =
    "'" символ + "'" | "0" ("x" | "X") шестнадцатеричная-цифра * |
    "0" восьмеричная-цифра * | цифра +
шестнадцатеричная-цифра =
    цифра | "a"..."f" | "A"..."F"
восьмеричная-цифра =
    "0"..."7"
строка =
    "\"" символ * "\""
вещественное =
    цифра * ["." [цифра *] [("e" | "E") ["+" | "-"] цифра *]
```

Приведенная грамматика не может служить определением языка Си. Это всего лишь иллюстрация, описывающая набор конструкций, которые синтаксический анализатор

компилятора считает «правильными». Многие из них будут отвергнуты на этапе семантической проверки. Например, конструкция

```
int short unsigned x;
```

правильна, а

```
int int int x;
```

ошибочна, хотя обе они соответствуют приведенной грамматике.

Тем не менее, приведенное выше описание может быть полезным для понимания тонких мест языка.

Рисунки.

заголовок файла
сегмент .const
сегмент .text
сегмент .data
информация о настройке
таблица символов

Структура файла a.out.

24 5	4 2	1
ссылка на таблицу символов	M	X

Структура информации о настройке для БЭСМ-6.

32 7	6 4	3 1
ссылка на таблицу символов	M	X

Структура информации о настройке для Эльбрус-Б.

признак архивного файла
заголовок файла 1
файл 1
заголовок файла 2
файл 2
...
заголовок файла N
файл N

Структура библиотечного (архивного) файла.

О Т З Ы В

на дипломную работу студента 372 группы МФТИ
Вакуленко С.В.

Разработка переносимой Си-ориентированной инструментальной системы программирования для ЭВМ Эльбрус-Б.

В данной работе подводится итог (начатой 3 года назад в Научном Студенческом Обществе ФУПМ МФТИ при поддержке ряда сотрудников Отделения Информатики и Электроники ИАЭ им.И.В.Курчатова) реализации для отечественных ЭВМ серии БЭСМ-6 СИ-компилятора на базе портабильной версии Джонсона. Эта версия транслятора языка СИ является ключевой для реализации всех версий Операционной Системы UNIX (4.3 BSD и System V) на конкретные архитектуры современных ЭВМ (от 16 разрядных микро-ЭВМ типа PC IBM/AT до супер-ЭВМ типа Cray-II, ETA-10).

С самых первых этапов этой работы основную часть ее выполнял студент Вакуленко С.В. В течение 1986-1987 учебного года был реализован вариант кросс-системы для ЭВМ БЭСМ-6 на базе ЭВМ Labtam-32.

Успешное выполнение этого промежуточного этапа позволило в феврале 1987 г., на совещании разработчиков ОС ДЕМОС в НПО «Центрпрограммсистем» (г. Калинин), гарантировать реализацию промышленной версии кросс-системы уже для «почти отечественной» и массовой ЭВМ СМ-4. В этом были заинтересованы сотрудники Новосибирского филиала ИТМиВТ им С.А.Лебедева, которые запланировали «имитационную» реализацию базовых возможностей ОС UNIX в рамках штатной ОС ФЕЛИКС для ВК Эльбрус-1К2 к середине 1988 года.

Качественное выполнение тов. Вакуленко С.В. этой работы к концу 1987 г. позволило группе сотрудников НФ ИТМиВТ приступить к полноценной реализации версии 7 ОС UNIX, успешно завершенной к осени 1988 года.

Эта реализация послужила в дальнейшем для самих разработчиков базовой инструментальной системой для второго - уже промышленного - этапа для 64-разрядного режима ЭВМ Эльбрус-Б. На этом этапе тов. Вакуленко С.В. заново перепроектировал основные компоненты кросс-системы: кросс-транслятор, ассемблер, загрузчик и необходимые библиотеки для целевой ЭВМ и передал их группе Основича А.Л. из НФ ИТМиВТ, которая намерена завершить реализацию полномасштабной версии 7 ОС UNIX (включая и транслятор для языка Fortran 77) к осени 1989 года.

Следует отметить, что выполненные тов. Вакуленко С.В. реализации переносимой версии транслятора языка СИ для отечественных ЭВМ были первыми в СССР и, как видно из вышеизложенного, предопределили успех истинной постановки ОС ДЕМОС - отечественной версии ОС UNIX - для ЭВМ класса БЭСМ-6.

Именно в этом и состоит главная практическая ценность представленной к защите дипломной работы Вакуленко С.В. Оформление дипломной записки прекрасно - и по полиграфическому качеству и по лаконичности изложения!

Проникновение во все тонкости реализации главной компоненты всех версий ОС UNIX - транслятора СИ - позволяет тов. Вакуленко С.В. легко - по ходу дела - выполнять крупные

разработки - как полностью оригинальные так и «адаптационные» во всем спектре современных аппаратных и операционных обстановок.

Дипломная работа Вакуленко С.В., несомненно, заслуживает оценки отлично и ее следует представить на конкурс лучших дипломных работ выпускников МФТИ за 1989 г.

Научный руководитель, к.ф.-м.н.
ИАЭ им.И.В.Курчатова
И. Г. Пасынков