Собрал Мигалин Сергей

Операционные системы

Домашнее задание 1

1. Расслоение памяти

Метод расслоения памяти (интерливинг) применяется для увеличения скорости доступа к основной (оперативной) памяти.

В обычном случае во время обращения к какой-то одной из ячеек модуля основной памяти никакие другие обращения к памяти производиться не могут. При расслоении памяти соседние по адресам ячейки размещаются в различных модулях памяти, так что появляется возможность производить несколько обращений одновременно.

Например, при расслоении на два направления ячейки с нечетными адресами оказываются в одном модуле памяти, а с четными — в другом. При простых последовательных обращениях к основной памяти ячейки выбираются поочередно. Таким образом, расслоение памяти позволяет обращаться сразу к нескольким ячейкам, поскольку они относятся к различным модулям памяти

2. Регистр перемещения

Регистр перемещения обеспечивает возможность динамического перемещения программ в памяти.

В регистр перемещения заносится базовый адрес программы, хранящейся в основной памяти. Содержимое регистра перемещения прибавляется к каждому указанному в выполняемой программе адресу. Благодаря этому пользователь может писать программу так, как если бы она начиналась с нулевой ячейки памяти.

Во время выполнения программы все исполнительные адреса обращений формируются с использованием регистра перемещения — и благодаря этому программа может реально размещаться в памяти совсем не в тех местах, которые она должна была бы занимать согласно адресам, указанным при трансляции.

3. Прерывания и опрос состояний

Одним из способов, позволяющих некоторому устройству проверить состояние другого, независимо работающего устройства, является опрос. Например, первое устройство может периодически проверять, находится ли второе устройство в определенном состоянии, и если нет, то продолжать свою работу. Опрос может быть связан с высокими накладными расходами.

Прерывания дают возможность одному устройству немедленно привлечь внимание другого устройства, с тем чтобы первое могло сообщить об изменении своего состояния.

Состояние устройства, работа которого прерывается, должно быть сохранено, только после этого можно производить обработку данного прерывания. После завершения обработки прерывания состояние прерванного устройства восстанавливается, с тем чтобы можно было продолжить работу.

4. Буферизация

Буфер — это область основной памяти, предназначенная для промежуточного хранения данных при выполнении операций ввода-вывода. Скорость выполнения операции ввода-вывода зависит от многих факторов, связанных с характеристиками аппаратуры ввода-вывода, однако в обычном случае ввод-вывод производится не синхронно с работой процессора.

При вводе, например, данные помещаются в буфер средствами канала ввода-вывода; после занесения данных в буфер процессор получает возможность доступа к этим данным. При вводе с простой буферизацией канал помещает данные в буфер, процессор обрабатывает эти данные, канал помещает следующие данные и т. д. В то время, когда канал заносит данные, обработка этих данных производиться не может, а во время обработки данных нельзя заносить дополнительные данные.

Метод *двойной буферизации* позволяет совмещать выполнение операции ввода-вывода с обработкой данных; когда канал заносит данные в один буфер, процессор может обрабатывать данные другого буфера. А когда процессор заканчивает обработку данных одного буфера, канал будет заносить новые данные опять в первый буфер. Такое поочередное использование буферов иногда называют буферизацией с переключением (*«триггерной» буферизацией*). Обмен данными между каналами и процессорами будет рассмотрен ниже.

5. Периферийные устройства

Периферийные (внешние) устройства – это аппаратура, которая позволяет вводить информацию в компьютер или выводить её из него.

Периферийные устройства делят на три типа:

- устройства ввода — устройства, использующиеся для ввода информации в компьютер: мышь, клавиатура, тачпад, сенсорный экран, микрофон, сканер, веб-камера, устройство захвата видео, ТВ-тюнер;
- устройства вывода — устройства, служащие для вывода информации из компьютера: видеокарта, монитор, принтер, акустическая система;
- устройства хранения(ввода/вывода) — устройства, служащие для накопления информации, обрабатываемой компьютером: накопитель на жёстких магнитных дисках (НЖМД), накопитель на гибких магнитных дисках (НГМД), ленточный накопитель, USB-флеш-накопитель.

6. Защита памяти

Защита памяти — важное условие для нормальной работы многоабонентских вычислительных систем (систем коллективного пользования). Защита памяти ограничивает диапазон адресов, к которым разрешается обращаться программе.

Защиту памяти для программы, занимающей непрерывный блок ячеек памяти, можно реализовать при помощи так называемых граничных регистров, где указываются старший и младший адреса этого блока памяти. При выполнении программы все адреса обращения к памяти контролируются, чтобы убедиться в том, что они находятся в промежутке между адресами, указанными в граничных регистрах.

Защиту памяти можно реализовать также при помощи ключей защиты памяти, относящихся к определенным областям основной памяти; программе разрешается обращение к ячейкам памяти только тех областей, ключи которых совпадают с ключом данной программы.

7. Таймер и часы

Интервальный таймер — эффективный способ предотвращения /монополизации процессора одним из пользователей в многоабонентских системах. По истечении заданного интервала времени таймер генерирует сигнал прерывания для привлечения внимания процессора; по этому сигналу процессор может переключиться на обслуживание другого пользователя.

Часы истинного времени дают возможность компьютеру следить за реальным календарным временем с точностью до миллионных долей секунды, а при необходимости даже точнее.

8. Каналы ввода – вывода

В первых компьютерах с ростом требуемых объемов вычислений, особенно в условиях обработки экономических данных, узким местом, как правило, оказывался ввод-вывод. Во время выполнения операций ввода-вывода процессоры были заняты управлением устройствами ввода-вывода. В некоторых машинах в каждый конкретный момент времени могла выполняться всего лишь одна операция ввода-вывода. Важным шагом для решения этой проблемы явилась разработка каналов ввода-вывода.

Канал ввода-вывода — это специализированный процессор, предназначенный для управления вводом-выводом независимо от основного процессора вычислительной машины.

Канал имеет возможность прямого доступа к основной памяти для записи или выборки информации.

В первых машинах взаимодействие между процессорами и каналами осуществлялось при помощи процессорных команд типа:

- условный переход (если канал занят выполнением операции);
- ожидание (пока не закончится выполнение команды канала);
- запись (содержимого управляющих регистров канала в основную память для последующего опроса процессором).

В современных машинах с управлением по прерываниям процессор выполняет команду «начать ввод-вывод» (SIO), чтобы инициировать передачу данных ввода- вывода по каналу; после окончания операции ввода-вывода канал выдает сигнал прерывания по завершению операции ввода-вывода, уведомляющий процессор об этом событии.

Истинное значение каналов состоит в том, что они позволяют значительно увеличить параллелизм работы аппаратуры компьютера и освобождают процессор от подавляющей части нагрузки, связанной с управлением вводом-выводом.

Для высокоскоростного обмена данными между внешними устройствами и основной памятью используется селекторный канал. *Селекторные каналы* имеют только по одному подканалу и могут обслуживать в каждый момент времени только одно устройство.

Мультиплексные каналы имеют много подканалов; они могут работать сразу с многими потоками данных в режиме чередования.

Байт-мультиплексный канал обеспечивает режим чередования байтов при одновременном обслуживании ряда таких медленных внешних устройств, как терминалы, перфокарточные устройства ввода-вывода, принтеры, а также низкоскоростные линии передачи данных.

Блок- мультиплексный канал при обменах в режиме чередования блоков может обслуживать несколько таких высокоскоростных устройств, как лазерные принтеры и дисковые накопители.

9. Захват цикла памяти

Узкое место, где может возникнуть конфликтная ситуация между каналами и процессором — это *доступ к основной памяти*.

Поскольку в каждый конкретный момент времени может осуществляться только одна операция обращения (к некоторому модулю основной памяти) и поскольку каналам и процессору может одновременно потребоваться обращение к основной памяти, в обычном случае приоритет здесь предоставляется каналам.

Это называется *захватом цикла памяти*; канал буквально захватывает, или «крадет» циклы обращения к памяти у процессора. Каналам требуется лишь небольшой процент общего числа циклов памяти, а предоставление им приоритета в этом смысле позволяет обеспечить лучшее использование устройств ввода-вывода. Подобный подход принят и в современных операционных системах; планировщики, входящие в состав операционной системы, как правило, отдают приоритет программам с большим объемом ввода-вывода по отношению к программам с большим объемом вычислений.

10. Относительная адресация

Когда потребность в увеличении объемов основной памяти стала очевидной, архитектуры компьютеров были модифицированы для работы с очень большим диапазоном адресов.

Машина, рассчитанная на работу с памятью емкостью 16 Мбайт (1 Мбайт — это 1 048 576 байт), должна иметь 24-разрядные адреса.

Включение столь длинных адресов в формат каждой команды даже для машины с одноадресными командами стоило бы очень дорого, не говоря уже о машинах с многоадресными командами. Поэтому для обеспечения работы с очень большими адресными пространствами в машинах начали применять адресацию типа база + смещение, или относительную адресацию, при которой все адреса программы суммируются с содержимым базового регистра.

Подобное решение имеет то дополнительное преимущество, что программы становятся перемещаемыми, или позиционно-независимыми; это свойство программ имеет особенно важное значение для многоабонентских систем, в которых одну и ту же программу может оказаться необходимым размещать в различных местах основной памяти при каждой загрузке.

11. Режимы работы компьютера

Режимы работы и их динамический выбор позволяет лучше организовать защиту программ и данных. Работающая программа может выполнять только некоторое подмножество команд, разрешенное для конкретного режима.

Режим задачи

Режим задачи, не позволяет, например, непосредственно производить операции ввода-вывода. Если программе пользователя разрешить выполнение любых операций ввода-вывода, она могла бы вывести главный список паролей системы, распечатать информацию любого другого пользователя или вообще испортить операционную систему.

Режим супервизора

Режим супервизора равносилен статусу самого полномочного пользователя. Присваивается обычно операционной системе. Дает доступ ко всем командам, предусмотренным в машине. Команды, которые нельзя выполнить в режиме задачи, называются *привилегированными*.

Для большинства современных вычислительных машин подобного разделения на два режима — задачи и супервизора — вполне достаточно. Однако в случае машин с высокими требованиями по защите от несанкционированного доступа желательно иметь более двух режимов. Благодаря этому можно обеспечить более высокую степень детализации защиты.

Доступ по принципу минимума привилегий: каждому конкретному пользователю следует предоставлять минимально возможный приоритет и право доступа только к тем ресурсам, которые ему действительно необходимы для выполнения предусмотренных задач.

В процессе развития компьютерных архитектур выявилась тенденция к увеличению количества привилегированных команд, что является свидетельством реализации большего числа функций операционных систем аппаратными средствами.

Некоторые микропроцессоры уже имеют целые операционные системы, реализованные микропрограммно, а во многих случаях значительная часть функций операционной системы реализуется в аппаратуре.

12. Виртуальная память

Системы *виртуальной памяти* дают возможность указывать в программах адреса, которым не обязательно соответствуют физические адреса основной памяти.

Виртуальные адреса, выдаваемые работающими программами, при помощи аппаратных средств во время выполнения программы преобразуются в адреса команд и данных, хранящихся в основной памяти.

Системы виртуальной памяти позволяют программам работать с адресными пространствами гораздо большего размера, чем адресное пространство основной памяти., создавать программы, независимые от ограничений основной памяти, и позволяют обеспечить работу многоабонентских систем с общими ресурсами.

В системах виртуальной памяти применяются методы:

- *страничная организация* - обмен между основной и внешней памятью блоками данных фиксированного размера
- *сегментация* - разделение программ и данных на логические компоненты, называемые сегментами, что упрощает управление доступом и коллективное использование

13. Мультипроцессорная обработка

В мультипроцессорных машинах несколько процессоров работают с общей основной памятью и одной операционной системой.

При такой работе возникает опасность конфликтных ситуаций. Чтобы их избежать, необходимо обеспечить координированный упорядоченный доступ к каждой общей ячейке памяти, чтобы два процессора не могли изменять ее содержимое одновременно — и в результате, быть может, портить его. Подобная координация необходима также и в случае, когда один процессор пытается изменить содержимое ячейки, которую хочет прочитать другой процессор.

Упорядочение доступа необходимо также и для однопроцессорных машин. Например, при использовании нескольких потоков.

14. Прямой доступ к памяти

Одним из способов достижения высокой производительности вычислительных машин является минимизация количества прерываний, происходящих в процессе выполнения программы.

Разработанный для этого способ *прямого доступа к памяти* (ПДП) требует лишь одного прерывания на каждый блок символов, передаваемых во время операции ввода-вывода. Благодаря этому обмен данными производится значительно быстрее, чем в случае, когда процессор прерывается при передаче каждого символа.

После начала операции ввода-вывода символы передаются в основную память по *принципу захвата цикла* — канал захватывает шину связи процессора с основной памятью на короткое время передачи одного символа, после чего процессор продолжает работу. Когда внешнее устройство оказывается готовым к передаче очередного символа блока, оно «прерывает» процессор.

Однако *в случае ПДП состояние процессора запоминать не требуется*, поскольку передача одного символа означает для процессора скорее задержку, или приостановку, чем обычное прерывание. Символ передается в основную память под управлением специальных аппаратных средств, а после завершения передачи процессор возобновляет работу.

ПДП—это способ повышения производительности, особенно необходимый для систем, в которых выполняются очень большие объемы операций ввода-вывода.

Аппаратные средства, обеспечивающие захват циклов памяти и управление устройствами ввода-вывода в режиме ПДП, называются каналом прямого доступа к памяти.

15. Конвейеризация

Конвейеризация — это аппаратный способ, применяемый в высокопроизводительных вычислительных машинах с целью использования определенных типов параллелизма для повышения эффективности обработки команд.

Структуру конвейерного процессора можно представить как технологическую линию производственного предприятия: на конвейере процессора на различных стадиях выполнения одновременно могут находиться несколько команд. Такое совмещение требует несколько большего объема аппаратуры, однако позволяет существенно сократить общее время выполнения последовательности команд.

16. Иерархия памяти

Современные вычислительные машины содержат несколько видов памяти, в том числе основную (первичную, оперативную), внешнюю (вторичную, массовую) и кэш-память. В основной памяти должны размещаться команды и данные, к которым будет обращаться работающая программа. Внешняя память — это магнитные ленты, диски, карты и другие носители, предназначенные для хранения информации, которая со временем будет записана в основную память. Кэш-память — это буферная память очень высокого быстродействия, предназначенная для повышения скорости выполнения работающих программ; для программ пользователя эта память, как правило, «прозрачна». В вычислительных машинах, оснащенных кэш-памятью, текущая часть программы помещается в кэшпамять, что позволяет выполнять ее гораздо быстрее, чем если бы она находилась в основной памяти. Все эти виды памяти создают единую иерархию памяти; переход по уровням этой иерархии от кэш-памяти к основной и затем к внешней памяти сопровождается уменьшением стоимости и скорости и увеличением емкости памяти. Память, как правило, разделяется на байты (символы) или слова (состоящие из постоянного количества байтов). Каждая ячейка памяти имеет свой адрес; множество адресов, доступных программе, называется адресным пространством.

Одной из основных составляющих любого компьютера является память. В идеале память должна быть максимально быстрой (работать быстрее, чем производится выполнение одной инструкции, чтобы работа центрального процессора не замедлялась обращениями к памяти), довольно большой и чрезвычайно дешевой. Никакая современная технология не в состоянии удовлетворить все эти требования, поэтому используется другой подход. Система памяти создается в виде иерархии уровней (рис. 1.9). Верхние уровни обладают более высоким быстродействием, меньшим объемом и более высокой удельной стоимостью хранения одного бита информации, чем нижние уровни, иногда в миллиарды и более раз.

Рис. 1.9. Типичная иерархия памяти. Приведенные значения весьма приблизительны

Верхний уровень состоит из внутренних регистров процессора. Они выполнены по той же технологии, что и сам процессор, и поэтому не уступают ему в быстродействии. Следовательно, к ним нет и задержек доступа. Внутренние регистры обычно предоставляют возможность для хранения 32 × 32 бит для 32-разрядного процессора или 64 × 64 бит — для 64-разрядного. В обоих случаях этот объем не превышает 1 Кбайт. Программы могут сами управлять регистрами (то есть решать, что в них хранить), без вмешательства аппаратуры.

Затем следует **кэш-память**, которая управляется главным образом аппаратурой. Оперативная память разделяется на **кэш-строки**, обычно по 64 байт, с адресами от 0 до 63 в кэш-строке 0, адресами от 64 до 127 в кэш-строке 1 и т. д. Наиболее интенсивно используемые кэш-строки оперативной памяти сохраняются в высокоскоростной кэш-памяти, находящейся внутри процессора или очень близко к нему. Когда программе нужно считать слово из памяти, аппаратура кэша проверяет, нет ли нужной строки в кэш-памяти. Если строка в ней имеется, то происходит результативное обращение к кэш-памяти (cache hit — кэш-попадание), запрос удовлетворяется за счет кэш-памяти без отправки запроса по шине к оперативной памяти. Обычно результативное обращение к кэшу занимает по времени два такта. Отсутствие слова в кэш-памяти вынуждает обращаться к оперативной памяти, что приводит к существенной потере времени. Кэш-память из-за своей высокой стоимости ограничена в объеме. Некоторые машины имеют два или даже три уровня кэша, причем каждый из последующих медленнее и объемнее предыдущего.

Кэширование играет существенную роль во многих областях информатики, это относится не только к кэшированию строк оперативной памяти. Довольно часто для повышения производительности к кэшированию прибегают везде, где есть какой-либо объемный ресурс, который можно поделить на фрагменты, часть из которых используется намного интенсивнее всех остальных. Операционные системы используют кэширование повсеместно. Например, большинство операционных систем держат интенсивно используемые файлы (или фрагменты файлов) в оперативной памяти, избегая их многократного считывания с диска. Точно так же результаты преобразования длинных имен файлов вроде /home/ast/projects/minix3/src/kernel/clock.c в дисковый адрес, по которому расположен файл, могут кэшироваться, чтобы исключить необходимость в повторных поисках. И наконец, может кэшироваться для дальнейшего использования результат преобразования адресов веб-страниц (URL) в сетевые адреса (IP-адреса). Можно привести массу других примеров использования технологии кэширования.

В любой системе кэширования довольно скоро возникает ряд вопросов.

1. Когда помещать в кэш новый элемент?
2. В какую область кэша помещать новый элемент?
3. Какую запись удалять из кэша, когда необходимо получить в нем свободное пространство?
4. Куда именно в памяти большей емкости помещать только что «выселенный» элемент?

Не каждый из этих вопросов имеет отношение к кэшированию. Например, в процессе кэширования строк оперативной памяти в кэше центрального процессора при каждом неудачном обращении к кэш-памяти в нее, как правило, будет вводиться новый элемент. При вычислении нужной кэш-строки для размещения нового элемента обычно

используются некоторые старшие биты того адреса памяти, на который осуществляется ссылка. Например, при наличии 4096 кэш-строк по 64 байта и 32-разрядных адресов биты с 6-го по 17-й могли бы использоваться для определения кэш-строки, а биты с 0-го по 5-й — для определения байта в кэш-строке. В этом случае элемент, подлежащий удалению, совпадает с тем элементом, в который попадают новые данные, но в других системах такой порядок может и не соблюдаться. Наконец, когда кэш-строка переписывается в оперативную память (если она была изменена в процессе кэширования), место в памяти, в которое ее нужно переписать, однозначно определяется адресом, фигурирующим в запросе.

Применение кэширования оказалось настолько удачным решением, что многие современные процессоры имеют сразу два уровня кэш-памяти. Первый уровень, или **кэш L1**, всегда является частью самого процессора и обычно подает декодированные команды в процессорный механизм исполнения команд. У многих процессоров есть и второй кэш L1 для тех слов данных, которые используются особенно интенсивно. Обычно каждый из кэшей L1 имеет объем 16 Кбайт. Вдобавок к этому кэшу процессоры часто оснащаются вторым уровнем кэш-памяти, который называется **кэш L2** и содержит несколько мегабайт недавно использованных слов памяти. Различия между кэш-памятью L1 и L2 заключаются во временной диаграмме. Доступ к кэшу первого уровня осуществляется без задержек, а доступ к кэшу второго уровня требует задержки в один или два такта.

При разработке многоядерных процессоров конструкторам приходится решать, куда поместить кэш-память. На рис. 1.8, *а* показан один кэш L2, совместно использующийся всеми ядрами. Такой подход применяется в многоядерных процессорах корпорации Intel. Для сравнения на рис. 1.8, *б* каждое ядро имеет собственную кэш-память L2. Именно такой подход применяет компания AMD. Каждый из подходов имеет свои аргументы «за» и «против». Например, общая кэш-память L2 корпорации Intel требует использования более сложного кэш-контроллера, а избранный AMD путь усложняет поддержание согласованного состояния кэш-памяти L2 разных ядер.

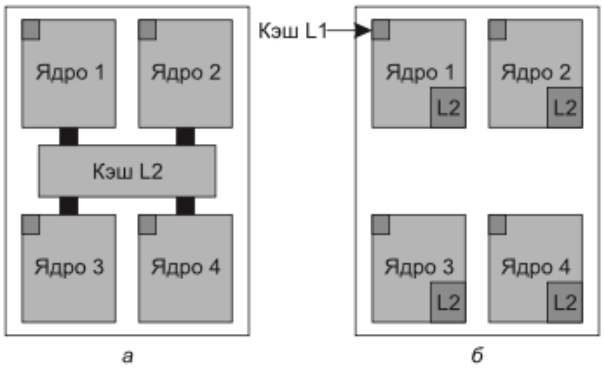

Рис. 1.8. Четырехъядерный процессор: *а* — с общей кэш-памятью второго уровня (L2); *б* — с отдельными блоками кэш-памяти L2

Следующей в иерархии идет оперативная память. Это главная рабочая область системы памяти машины. Оперативную память часто называют оперативным запоминающим устройством (**ОЗУ**), или памятью с произвольным доступом (Random Access Memory (**RAM**)). Ветераны порой называют ее core memory — памятью на магнитных сердечниках, поскольку в 1950–1960-е годы в оперативной памяти использовались крошечные намагничиваемые ферритовые сердечники. Прошли десятилетия, но название сохраняется. В настоящее время блоки памяти имеют объем от сотен мегабайт до нескольких гигабайт, и этот объем стремительно растет. Все запросы

процессора, которые не могут быть удовлетворены кэш-памятью, направляются к оперативной памяти.

Дополнительно к оперативной памяти многие компьютеры оснащены небольшой по объему неизменяемой памятью с произвольным доступом — постоянным запоминающим устройством (**ПЗУ**), оно же память, предназначенная только для чтения (Read Only Memory (**ROM**)). В отличие от ОЗУ она не утрачивает своего содержимого при отключении питания, то есть является энергонезависимой. ПЗУ программируется на предприятии-изготовителе и впоследствии не подлежит изменению. Эта разновидность памяти характеризуется высоким быстродействием и дешевизной. На некоторых компьютерах в ПЗУ размещается начальный загрузчик, используемый для их запуска. Такой же памятью, предназначенной для осуществления низкоуровневого управления устройством, оснащаются некоторые контроллеры ввода-вывода.

Существуют также другие разновидности энергонезависимой памяти, которые в отличие от ПЗУ могут стираться и перезаписываться, — электрически стираемые программируемые постоянные запоминающие устройства (**ЭСППЗУ**), они же **EEPROM** (Electrically Erasable PROM), и флеш-память. Однако запись в них занимает на несколько порядков больше времени, чем запись в ОЗУ, поэтому они используются для тех же целей, что и ПЗУ. При этом они обладают еще одним дополнительным свойством — возможностью исправлять ошибки в содержащихся в них программах путем перезаписи занимаемых ими участков памяти.

Флеш-память также обычно используется как носитель информации в портативных электронных устройствах. Если упомянуть лишь два ее применения, то она служит «пленкой» в цифровых фотоаппаратах и «диском» в переносных музыкальных плеерах. По быстродействию флеш-память занимает промежуточное положение между ОЗУ и диском. Также, в отличие от дисковой памяти, если флеш-память стирается или перезаписывается слишком часто, она приходит в негодность.

Еще одна разновидность памяти — CMOS-память, которая является энергозависимой. Во многих компьютерах CMOS-память используется для хранения текущих даты и времени. CMOS-память и схема электронных часов, отвечающая за отсчет времени, получают питание от миниатюрной батарейки (или аккумулятора), поэтому значение текущего времени исправно обновляется, даже если компьютер отсоединен от внешнего источника питания. CMOS-память также может хранить параметры конфигурации, указывающие, например, с какого диска системе следует загружаться. Потребление энергии CMOS- памятью настолько низкое, что батарейки, установленной на заводе-изготовителе, часто хватает на несколько лет работы. Однако когда батарейка начинает выходить из строя, на компьютере могут проявиться признаки «болезни Альцгеймера» и он станет «забывать» то, что помнил годами, например с какого жесткого диска следует производить загрузку.

17. Программирование на машинном языке

Машинный язык — это язык программирования, непосредственно воспринимаемый компьютером. Каждая команда машинного языка интерпретируется аппаратурой, выполняющей указанные функции. Команды машинного языка в принципе являются довольно примитивными. Только соответствующее объединение этих команд в программы на машинном языке дает возможность описывать достаточно серьезные алгоритмы. Наборы команд машинного языка современных компьютеров зачастую включают некоторые очень

эффективные возможности (см, например, описание миникомпьютеров VAX в гл. 19). Говорят, что машинный язык является машинно-зависимым: программа, написанная на машинном языке компьютера одного типа, как правило, не может выполняться на компьютере другого типа, если его машинный язык не идентичен машинному языку первого компьютера (или не является расширением по отношению к этому языку). Еще одним признаком машинной, или аппаратной, зависимости является характер самих команд: в командах машинного языка указываются наименования конкретных регистров компьютера и предусматривается обработка данных в той физической форме, в которой они существуют в этом компьютере. Большинство первых компьютеров программировались непосредственно на машинном языке, а в настоящее время на машинном языке пишется лишь очень небольшое число программ.

18. Ассемблеры и макропроцессоры

Программирование на машинном языке требует очень много времени и чревато ошибками. Поэтому были разработаны языки ассемблерного типа, позволяющие повысить скорость процесса программирования и уменьшить количество ошибок кодирования. Вместо чисел, используемых при написании программ на машинных языках, в языках ассемблерного типа применяются содержательные мнемонические сокращения и слова естественного языка. Однако компьютеры не могут непосредственно воспринять программу на языке ассемблера, поэтому ее необходимо вначале перевести на машинный язык. Такой перевод осуществляется при помощи программы-транслятора, называемой ассемблером. Языки ассемблерного типа также являются машинно-зависимыми. Их команды прямо и однозначно соответствуют командам программы на машинном языке. Чтобы ускорить процесс кодирования программы на языке ассемблера, были разработаны и включены в ассемблеры так называемые макропроцессоры. Программист пишет макрокоманду как указание необходимости выполнить действие, описываемое несколькими командами на языке ассемблера. Когда макропроцессор во время трансляции программы читает макрокоманду, он производит макрорасширение — т. е. генерирует ряд команд языка ассемблера, соответствующих данной макрокоманде, Таким образом, процесс программирования значительно ускоряется, поскольку программисту приходится писать меньшее число команд для определения того же самого алгоритма.

19. Компиляторы

Тенденция к повышению вычислительной мощности команд привела к разработке некоторых очень эффективных макропроцессоров и макроязыков, упрощающих программирование на языке ассемблера. Однако развитие ассемблеров путем введения макропроцессоров не решает проблемы машинной зависимости. В связи с этим были разработаны языки высокого уровня. Языки высокого уровня открывают возможность машинно-независимого программирования. Большинству пользователей компьютер нужен только как средство реализации прикладных систем. Языки высокого уровня позволяют пользователям заниматься преимущественно задачами, специфичными для их конкретных прикладных областей, не вникая в особенности применяемых ими машин. Благодаря этому существенно повышается скорость процесса программирования, обеспечивается обмен программами между машинами различных типов, и у людей появляется возможность разрабатывать эффективные прикладные системы, не затрачивая времени и сил на полное изучение внутренней структуры машины. Перевод с языков высокого уровня на машинный язык осуществляется при помощи программ, называемых компиляторами. Компиляторы и ассемблеры имеют общее название «трансляторы». Написанная пользователем программа,

которая в процессе трансляции поступает на вход транслятора, называется исходной программой; программа на машинном языке, генерируемая транслятором, называется объектной программой, или выходной (целевой) программой.

Когда программы разрабатываются и отлаживаются, компиляции производятся часто, а выполняются программы обычно недолго — пока не обнаружится очередная ошибка. В этих условиях вполне приемлемы быстрые компиляторы без оптимизации. Они позволяют быстро получить объектную программу, однако ее код может оказаться совершенно неэффективным с точки зрения как занимаемой памяти, так и скорости выполнения. После того как программа отлажена и готова для производственной эксплуатации, при помощи оптимизирующего компилятора для нее генерируется эффективный машинный код. Оптимизирующий компилятор работает гораздо медленнее, однако обеспечивает получение объектного кода очень высокого качества. В 70-х годах господствовало мнение о том, что хороший программист, работающий на языке ассемблера, может создать гораздо лучший код программы, чем оптимизирующий компилятор, В настоящее время оптимизирующие компиляторы настолько эффективны, что генерируемый ими код по качеству не уступает или даже превосходит код, созданный высококвалифицированным программистом на языке ассемблера. Благодаря этому программы, которые должны быть исключительно эффективными (например, операционные системы), больше не приходится писать на языке ассемблера. Сегодня крупные операционные системы в большинстве случаев пишутся на языках высокого уровня и при помощи оптимизирующих компиляторов очень высокого качества транслируются в эффективный машинный код.

20. Интерпретаторы

Существует один интересный и распространенный вид трансляторов, интерпретаторы, которые не генерируют объектную программу, а фактически обеспечивают непосредственное выполнение исходной программы. Интерпретаторы особенно распространены в системах проектирования программ, где программы идут, как правило, лишь небольшое время до момента обнаружения очередной ошибки. Интерпретаторы распространены также в сфере персональных компьютеров. Они свободны от накладных затрат, свойственных ассемблированию или компиляции. Однако выполнение программы в режиме интерпретации идет медленно по сравнению с компилированным кодом, поскольку интерпретаторам надо транслировать каждую команду при каждом ее выполнении.

21. Процедурно-ориентированные и проблемно-ориентированные языки

Языки высокого уровня бывают либо *процедурно-ориентированными*, либо *проблемно-ориентированными*.

Процедурно-ориентированные языки высокого уровня —это универсальные языки программирования, которые можно использовать для решения самых разнообразных задач. *Проблемно-ориентированные языки* предназначаются специально для решения задач конкретных типов. Такие языки, как Паскаль, Кобол, Фортран, Бейсик и ПЛ/1 обычно считаются процедурно-ориентированными, а такие языки, как GPSS (язык моделирования) и SPSS (язык для выполнения статистических вычислений),—проблемно-ориентированными.

22. Спулинг

При *спулинге* (вводе-выводе с буферизацией) посредником между работающей программой и низкоскоростным устройством, осуществляющим ввод-вывод данных для этой программы, становится высокоскоростное устройство, например дисковый накопитель. Вместо вывода строк данных непосредственно, скажем, на построчно печатающее устройство программа записывает их на диск. Благодаря этому текущая программа может быстрее завершиться, с тем чтобы другие программы могли быстрее начать работать. А записанные на диск строки данных можно, распечатать позже, когда освободится принтер. Для этой процедуры название «спулинг» весьма подходит, поскольку она напоминает намотку нитки на катушку, с тем чтобы ее можно было при необходимости размотать.

23. Абсолютные и перемещающие загрузчики

Загрузчик —это системная программа, которая размещает команды и данные программы в ячейках основной памяти. *Абсолютный загрузчик* размещает эти элементы именно в те ячейки, адреса которых указаны в программе на машинном языке. *Перемещающий загрузчик* может загружать программу в различные места основной памяти в зависимости, например, от наличия свободного участка основной памяти в момент загрузки (называемое временем загрузки).

Программы для выполнения должны размещаться в основной памяти. Решение этой задачи иногда осуществляет сам пользователь, иногда транслятор, иногда системная программа, называемая загрузчиком, а иногда—операционная система. Сопоставление команд и элементов данных с конкретными ячейками памяти называется *привязкой к памяти.* При программировании на машинном языке привязка к памяти производится в момент кодирования. Отсрочка привязки программы к памяти обеспечивает увеличение гибкости как для пользователя, так и для системы, однако это связано с существенным повышением сложности трансляторов, загрузчиков, аппаратных средств и операционных систем.

24. Связывающие загрузчики и редакторы связей

В составе системного программного обеспечения поставляются большие *библиотеки подпрограмм*, так что программист, которому необходимо выполнять определенные стандартные операции, может воспользоваться для этого готовыми подпрограммами. В частности, операции ввода-вывода обычно выполняются подпрограммами, находящимися вне программы пользователя. Поэтому программу на машинном языке, полученную в результате трансляции, приходится, как правило, комбинировать с другими программами на машинном языке, чтобы сформировать необходимый выполняемый модуль. Эту процедуру объединения программ выполняют *связывающие загрузчики* и *редакторы связей* до начала выполнения программы.

Связывающий загрузчик во время загрузки объединяет необходимые программы и загружает их непосредственно в основную память. *Редактор связей* также осуществляет подобное объединение программ, однако он создает *загрузочный модуль*, который записывается во внешнюю память для будущего использования. Редактор связей играет особенно важную роль для производственных систем; когда программу необходимо выполнять, ее загрузочный модуль, сформированный при помощи редактора связей, может быть загружен немедленно —без накладных затрат времени (часто весьма больших) на повторное объединение отдельных частей программы.

25. Микропрограммы

В конце 60-х —начале 70-х годов появилось *динамическое микропрограммирование*, предусматривающее возможность легкой загрузки новых микропрограммных модулей в управляющую память, служащую для выполнения микропрограмм. Благодаря этому стало возможным динамично и часто менять наборы команд вычислительной машины

Микропрограммирование вводит дополнительный уровень средств программирования, нижележащий по отношению к машинному языку компьютера, и тем самым оно позволяет определять конкретные команды машинного языка. Подобные возможности являются неотъемлемой частью архитектуры современных компьютеров и имеют громадное значение с точки зрения обеспечения высоких скоростных характеристик и защиты операционных систем.

Микропрограммы размещаются в специальной управляющей памяти очень высокого быстродействия. Они состоят из индивидуальных микрокоманд, которые гораздо более элементарны по своей природе и более рассредоточены по функциям, чем обычные команды машинного языка. В компьютерах, где набор команд машинного языка реализуется при помощи микропрограммирования, каждой команде машинного языка соответствует целая и ,быть может, большая микропрограмма, Тем самым сразу же становится очевидным, что микропрограммирование окажется эффективным только в том случае, если управляющая память будет обладать гораздо большим быстродействием; чем основная.

26. Горизонтальный и вертикальный микрокод

Команды микрокода можно разделить на *горизонтальные* и *вертикальные*.

Выполнение *вертикальных микрокоманд* очень похоже на выполнение обычных команд машинного языка. Типичная вертикальная микрокоманда задает пересылкуодного или нескольких элементов данных между регистрами.

Горизонтальный микрокод действует совсем по-другому. Каждая его команда содержит гораздо большее число бит, поскольку может задавать параллельную операцию пересылки данных для многих или даже всех регистров данных устройства управления.

Горизонтальные микрокоманды являются более мощными, чем вертикальные, однако может оказаться, что соответствующие микропрограммы гораздо сложнее кодировать и отлаживать.

27. Эмуляция

Эмуляция — метод, позволяющий сделать одну вычислительную машину функциональным эквивалентом другой. Набор команд машинного языка эмулируемого компьютера микропрограммируется на эмулирующем компьютере —и благодаря этому программы, представленные на машинном языке первого компьютера, могут непосредственно выполняться на втором. Фирмы-разработчики компьютеров широко применяют эмуляцию при внедрении новых машин, и пользователи, привязанные к старым компьютерам, получают, например, возможность без всяких изменений выполнять свои ранее отлаженные программы на новых машинах. Тем самым процесс перехода с машины на машину становится менее сложным и болезненным.