Машинная команда. Регистры процессора

26 января 2015 г.

21:03

**Машинная команда** — это описание операции, которую должна выполнять ЭВМ. Содержит выполняемые операции, указание по определению операндов и их адресов, типы размещения результатов выполненной операции.

Команды часто распределяются на арифметические, логические, команды пересылки данных, команды ввода-вывода в порты и команды вызова прерывания.

Совокупность выполняемых машиной операций называется **системой команд**.

**Регистр** — это совокупность элементов, принимающих значение 1 или 0, используемых для хранения информации.

Основные регистры:

* AX — аккумулятор;
* BX — базовый;
* CX — счётчик;
* DX — данные.

Сегментные регистры:

* CS — указатель на кодовый сегмент. Пара CS:IP однозначно определяет реальный физический адрес следующей выполняемой инструкции.
* DS — данные;
* SS — стек;
* ES — дополнительный.

EAX (аккумулятор), EBX (базовый), ECX (счётчик), EDX (данных), EBP (указателя базы), ESP (указатель на вершину стека), ESI (индексный), EDI (индексный) — регистры общего назначения, которые можно использовать. При выполнении текущей инструкции процессор автоматически изменяет значение в регистре IP, в результате чего регистровая память CS:IP всегда указывает на следующую, подлежащую исполнению память.

**Регистр флага** — это бит, принимающий значение единицы, если выполнено некоторое условие, и значение ноль в противном случае. Процессор имеет регистр флагов, содержащий набор флагов, отражающий текущее состояние процессора.

Сегменты (кодов, данных, стека)

26 января 2015 г.

21:12

**Сегментом** называется область памяти, которая начинается с адреса кратного 16.

При выполнении программы определяется три главных сегмента:

1. **Сегмент кода** — содержит машинные команды, из которых обычно первая исполняемая команда находится в начале этого сегмента и операционная система передаёт управление по адресу данного сегмента для выполнения программы.
2. **Сегмент данных** — содержит данные, переменные, массивы, константы, необходимые для работы программы. Начальный адрес сегмента расположен в регистре DS.
3. **Сегмент стека** — содержит адреса возврата, как для программы при возврате в ОС, так и для вызовов подпрограмм при возврате в главную программу. Адрес данного сегмента в регистре SS.
4. **Дополнительный сегмент** — используется в специальных случаях.

Стек так же можно использовать для временного хранения данных. Регистр указателя стека (SP) постоянно указывает на вершину стека. Данные помещаются в стек и извлекаются из него с помощью PUSH и POP. Увеличение стека происходит по направлению оставшихся ячеек памяти, к младшим.

Таким образом при помещении данных в стек, содержимое регистра SP уменьшается, а при извлечении данных увеличивается.

Тактовый генератор. Цикл выполнения команды

26 января 2015 г.

21:15

**Тактовый генератор**, или генератор тактовых импульсов, задаёт рабочую частоту процессора. С помощью тактовых импульсов выполняется синхронизация для внутренних команд процессора и остальных устройств. Тактовый генератор вырабатывает (генерирует) импульсы, которые следуют с определённой частотой (для разных процессоров частота разная).

**Машинный такт** соответствует одному периоду импульсов тактового генератора и является основной единицей измерения времени выполнения команд процессором.

**Цикл выполнения команды** — это время выполнения для одной машинной команды. Цикл выполнения команды реализуется за несколько машинных тактов, точное число которых зависит от сложности команды и, как правило, равно числу обращений процессора к памяти и устройствам ввода/вывода.

Выбор и выполнение программы:

1. Выбрать команду из оперативной памяти и поместить её в регистр.
2. Выбрать операнды из памяти; если в команде используется операнд, расположенный в оперативной памяти, то блок управления начинает её выборку из памяти.
3. Увеличение IP.
4. Выполнение команды.

Основные команды ассемблера

26 января 2015 г.

21:24

Арифметические:

1. ADD
2. SUB
3. INC
4. DEC
5. NEG
6. MUL (IMUL)
7. DIV (IDIV)

Логические:

1. AND
2. OR
3. EOR (XOR?)
4. COM (обратный код)
5. NEG (дополнительный код)
6. CLR (X XOR X)
7. SER (0FFH)
8. TST (X AND X)

Пересылка данных:

1. MOV
2. LDI (константа в регистр)
3. PUSH
4. POP

Управление системой:

1. NOP

Безусловный переход:

1. JMP
2. RET
3. CALL

Условный переход и сравнение:

1. JO / JNO
2. JB / JNB
3. JE / JNE
4. JBE / JNBE
5. JS / JNS
6. CMP

Команды управления циклом:

1. LOOP
2. LOOPE / LOOPZ

Структура программы на ассемблере

26 января 2015 г.

21:47

Программа на ассемблере представляет собой совокупность блоков памяти, называемых сегментами памяти.

Программа может состоять из одного или нескольких таких блоков-сегментов. Каждый сегмент содержит совокупность предложений языка, каждое из которых занимает отдельную строку кода программы.

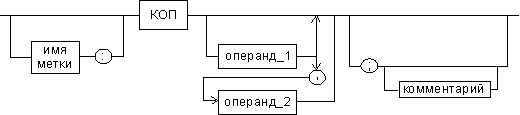
Предложения ассемблера бывают четырёх типов:

* команды или инструкции, представляющие собой символические аналоги машинных команд.

В процессе трансляции инструкции ассемблера преобразуются в соответствующие команды системы команд микропроцессора;

* макрокоманды — оформляемые определённым образом предложения текста программы, замещаемые во время трансляции другими предложениями;
* директивы, являющиеся указанием транслятору ассемблера на выполнение некоторых действий. У директив нет аналогов в машинном представлении;
* строки комментариев, содержащие любые символы, в том числе и буквы русского алфавита. Комментарии игнорируются транслятором.

Формат команд:



Директивы объявления данных

26 января 2015 г.

21:53

|  |  |  |
| --- | --- | --- |
| **Размер**  **(в байтах)** | **Объявление** | **Резервирование** |
| 1 | db | rb |
| 2 | dw  du | rw |
| 4 | dd | rd |
| 6 | dp  df | rp  rf |
| 8 | dq | rq |
| 10 | dt | rt |
| N | file |  |

x db 5

array1 dw 1,2,3,4,5

array2 db 5 dup(1)

array3 dd 4 dup(3,7,0)

str1 db 'Hello'

str2 db 'Hello',0

str3 db 'Hello$'

x1 db ?

x2 dw ?,?,?

x3 dd 10 dup(?)

data1 file 'data.bin'

data2 file 'data.bin':20

data3 file 'data.bin':20,5

Механизм работы команды call. Передача аргументов в подпрограммы

26 января 2015 г.

21:59

Подпрограммы вызываются командой call. При этом выполняются следующие действия:

1. В стеке сохраняется адрес следующей после call команды (для мал. моделей только регистр IP, большие – CS:IP).
2. В регистр CS:IP или IP загружается адрес точки входа в подпрограмму.
3. Последней командой подпрограммы должна быть команда ret. Она загружает из стека адрес следующей после call команды.

my\_proc PROC

…

my\_proc ENDP.

Для передачи параметров используется 2 способа:

* через регистр;
* через стек.

Параллельные вычислительные системы

26 января 2015 г.

22:10

Только *структурными* методами можно добиться необходимой производительности вычислительных средств. Таким основным *структурным методом* является *распараллеливание вычислений*.

Вместо одной ЭВМ в систему управления необходимо встраивать несколько, решающих единый набор функционально связанных задач. Такой коллектив ЭВМ получил название вычислительного комплекса (ВК). Информационная взаимосвязанность задач потребовала применения средств оперативного обмена данными внутри ВК и *средств синхронизации*. Однако временные *затраты* на организацию взаимодействия ЭВМ в ВК (накладные *расходы*) оказались весьма высоки. Возникла идея использования взамен *разделённой* оперативной памяти (ОП), реализованной в ВК, *общей (разделяемой)* ОП, которая связывает несколько центральных процессоров. Проблема затрат времени на обмен оказалась решённой. Так зародилось понятие *параллельной вычислительной системы* (*ВС*).

Универсальным критерием, используемым при решении задач распараллеливания, является *минимум времени выполнения* совокупности *работ*, распределяемых между процессорами.

Тесты для оценки производительности

26 января 2015 г.

22:16

Для того, чтобы оценить эффективность работы вычислительной системы на реальных задачах был разработан фиксированный набор тестов. Наиболее известным из них является LINPACK. Тест состоит в решении системы линейных уравнений с помощью LU-факторизации. Основное время затрачивается на решение векторных операций (сложение и умножение). Производительность определяется количеством «полезных» вычислительных операций и выражается в gigaFLOPS, автор данного теста Jack Dongarra. Результаты теста используются при составлении списка ТОП-500 самых мощных компьютеров мира. В настоящее время большое распространение получили тестовые программы, взятые из разных программных областей и представляющие собой реальные промышленные приложения. Такие тесты позволяют оценить производительность

действительно в реальных задачах. И получить наиболее полное представление об эффективности работы компьютера с конкретными приложениями.

Наборы тестов:

1. NBD состоит из 8 программ для определения производительности параллельных компьютеров.
2. PERFECT представляет собой комплект из 13 прикладных Fortran программ, представляющих 4 типа вычислительных задач — аэро- и гидродинамики, моделирование химических и физических процессов, инженерного проектирования, а также обработки сигналов.
3. Matrix Multiply (MM) этот тест содержит 9 различных программ умножения матриц размером 500х500. Им оценивается работа КЭШ-памяти и уровень оптимизации компилятора.
4. SLALOM оценивает объем вычислений, которые может произвести компьютер за одну минуту.
5. Stanford состоит из 8 целочисленных тестов (умножение матриц, сортировка тремя методами, перестановки, ханойские башни, расстановка восьми ферзей, головоломка)
6. STREAM — тест для работы с памятью. Основан на выполнении больших векторных операций: копирование в памяти, умножение на константу, умножение и сложение.

Адресация данных и команд

26 января 2015 г.

22:23

Любая машинная команда — это машинный код, который определяет первую операцию, указывает на данные. В адресной части команды хранится адресный код. В большинстве случаев фактическое обращение к данным происходит по физическому адресу. Обычно физической адрес не совпадает с адресным полем команды, но зависит от него. В общем случае происходит преобразование из адресного кода в физический. Способы адресации являются одним из основных архитектурных признаков. В настоящее время известно более двух десятков различных способов адресации и их модификации. Все способы делятся на две большие группы:

1. Прямые — либо накопительный адрес операнда, либо сам операнд находится непосредственно по адресному коду без всякого преобразования.
2. Непрямые — требуют выполнения процедур формирования физического адреса по адресному коду для этого ЭВМ.

Различают адресацию операндов:

1. MOV AX, 5 (непосредственная адресация)

2. MOV DS, AX (прямая регистровая адресация)

3. MOV AL, message[SI] (прямая с индексированием)

4. MOV AL, message[BX] (прямая с базированием)

5. MOV AL, message[BX + SI](прямая с индексированием и базированием)

6. (косвенная адресация обычная)

7. (косвенная адресация с индексированием и базированием)

Адресация с базированием и индексированием полезна при работе с двумерными массивами и таблицами, в ней исполнительный адрес вычисляется, как значение базового регистра, индексного регистра и сдвига. В случае двумерного массива базовый адрес может содержать начальный адрес массива, а значение вида и индексного регистра могут содержать смещение по строке и столбцу.

Сплошная (flat) модель памяти

26 января 2015 г.

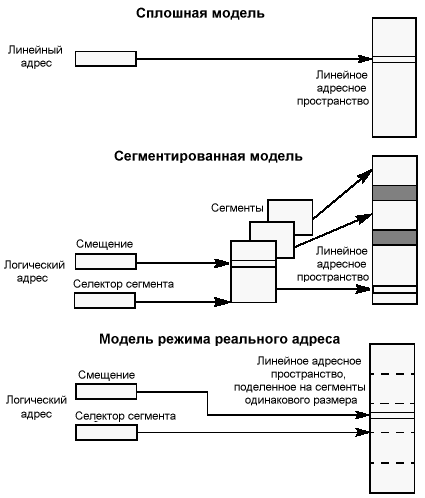
22:25

**Плоская модель памяти** — метод организации адресного пространства оперативной памяти вычислительных устройств. В плоской модели код и данные используют одно и то же адресное пространство. Для 16-битных процессоров плоская модель памяти позволяет адресовать 64 кБ оперативной памяти; для 32-битных процессоров 4 ГБ, для 64-битных — гипотетически до 16 эксабайт, фактически до 256 ТБ.

Управление памятью в основном реализуется на основе плоской модели, в целях содействия функциональности операционной системы, защиты ресурсов, многозадачности или увеличения объёма памяти за пределы ограничений, налагаемых физическим адресным пространством процессора.

Преимущества управления памятью с плоской моделью:

* В одном из многозадачных встроенных приложений, где управление памятью не нужно и не желательно, модель обеспечивает простейший интерфейс для программирования, с прямым доступом ко всем местам в памяти и минимальной сложностью конструкции программы.
* При многозадачности и распределении ресурсов плоская модель по-прежнему обеспечивает максимальную гибкость для реализации этого типа управления памятью.



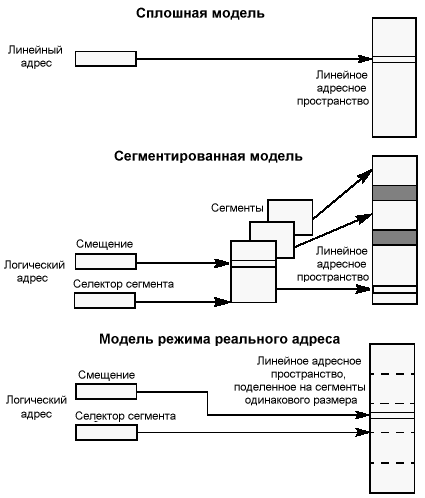
Сегментированная модель памяти

26 января 2015 г.

22:29

Сегментация — это механизм адресации обеспечивающий существование нескольких адресных пространств, как в пределах одной задачи, так и системы в целом для защиты задач от взаимного влияния. В основе механизма сегментации лежит понятие сегмента, который представляет собой блок памяти. Каждая программа имеет доступ только к трём основным сегментам: стека, данных и кода, а также к дополнительным сегментам данных, число которых от 1 до 3.

При использовании сегментированной модели (Segmented Model) для программы память представляется группой независимых адресных блоков, называемых сегментами. Для адресации байта памяти программа должна использовать логический адрес, состоящий из селектора сегмента и смещения. Селектор сегмента выбирает определённый сегмент, а смещение указывает на конкретный байт в адресном пространстве выбранного сегмента. Каждая задача в защищённом режиме может иметь до 16383 сегментов, размером до 4 Гбайт каждый, таким образом, общий объем памяти, адресуемой программой составляет 64 Тбайт. Микропроцессор при помощи блока сегментации отображает логический адрес в линейное адресное пространство. Сегментация позволяет эффективно управлять пространством логических адресов. Сегменты используются для объединения областей памяти с общими атрибутами. Каждый сегмент имеет несколько связанных с ним атрибутов: размер, расположение, тип (стек, программа или данные) и характеристики защиты.



Понятие шины

26 января 2015 г.

22:31

**Компьютерные шины (computer bus)**— служат для передачи данных между отдельными функциональными блоками компьютера и представляют собой совокупность сигнальных линий, которые имеют определённые электрические характеристики и протоколы передачи информации.

Шины могут различаться: разрядностью, способом передачи сигнала (последовательный или параллельный, синхронный или асинхронный), пропускной способностью, количеством и типами поддерживаемых устройств, протоколами работы и назначением.

Операции по шине называются транзакциями. Транзакции включают в себя: посылку адреса, посылку данных или приём данных. При обмене данными по шине, одно из устройств (ведущее) должно инициировать обмен и управлять пересылкой данных. Остальные устройства (ведомые) не обладают способностью инициировать транзакции. К шине может быть подключено несколько ведущих устройств, но в любой момент времени активным может быть только одно. Это делается, чтобы не допускать искажения передаваемой информации. Предотвращение одновременной активности нескольких ведущих обеспечивается специальной процедурой допуска одного из них к управлению шиной называемой арбитражем.

Физические шины реализованы в виде параллельных медных проводников материнской платы. Поперёк основной шины материнской платы установлены разъёмы адаптеров расширений внешних устройств ввода/вывода. Механические спецификации шин включают в себя такие характеристики такие как: размеры и размещения направляющих для их установки, разрешённое место для размещения кабельного разъёма и т.д.

Устройства, использующие шину электрически, подсоединены к её сигнальным линиям, меняя уровень напряжения на этих линиях, ведущее устройство формирует информационные или управляющие сигналы. Когда ведущее устройство выставляет на сигнальной шине какой-то уровень напряжения, этот уровень может быть воспринят приёмниками (схемами, принимающими информацию с шины). Любая транзакция по шине начинается с выставления ведущим устройством адресной информации на сигнальные шины адреса. Адрес позволяет выбрать ведомое устройство и установить соединение между ним и ведущим. По шине адреса могут передаваться адреса ячеек оперативной памяти, номера регистров процессора и адреса портов ввода/вывода.

От момента появления данных на шине, до момента появления их на адресуемом устройстве проходит некоторое время. Один из способов уменьшения этого интервала — это сокращение длины шины данных. При поступлении данных на устройство требуется выдержать некоторое время стабилизации сигнала, т.е. время, в течении которого устройство должно распознать приход новых данных. Чтобы устройство могло распознать пришедшие данные, они должны быть на входе устройства в течении некоторого времени стабилизации. Непременным атрибутом любой шины является группа линий, предназначенных для передачи управляющей информации и данных о состоянии участвующих в транзакции устройств. Совокупность таких сигнальных линий называют шиной управления.

Основные типы шин

26 января 2015 г.

22:38

Выделяют следующие типы шин:

1. Шина «процессор—память» (или шина переднего плана (Front-Side bus, FSB)). Обеспечивает связь между центральным процессором и оперативной памятью, иногда роль этой шины выполняет системная шина.
2. Шина заднего плана (Back-Side bus, BSB). Связывает системный процессор и кэш-память
3. Шина ввода/вывода обеспечивает соединение с устройствами ввода/вывода (примеры шин: PCI, SATA, SCSI, USB)
4. Системная шина используется для обмена данными по единому каналу как с оперативной памятью, так и с устройствами ввода/вывода, и позволяет логически и физически объединить все устройства на шине. Системная шина содержит несколько сотен линий, разделённых на 3 функциональные группы: данных, адреса и управления, включая линии для подачи питания. Если одному из устройств на шине необходимо передать данные в другое устройство, то оно должно выполнить следующее: получить в распоряжение шину и передать по ней данные. Если же какое-то устройство хочет получить от другого необходимые данные, оно должно получить доступ к шине, а затем передать в другое устройство запрос, затем оно переходит в состояние ожидания до тех пор, пока устройство получившее запрос перешлёт данные. Подключение к шине большого количества устройств ведёт к падению её пропускной способности. Поэтому используется иерархия шин вместо одной системной.
5. Системная шина центрального процессора (host bus) используется для связи центрального процессора с материнской платой
6. Шина оперативной памяти (memory bus) нужна для связи оперативной памяти с материнской платой и центральным процессором
7. Шина кэша (cache bus) используется для связи центрального процессора с кэш-памятью.
8. Локальная шина ввода/вывода (PCI-Express) — это шина соединяющая высокопроизводительное оборудование типа видеоадаптеров, дисковых накопителей и сетевых адаптеров с материнской платой, центральным процессором и оперативной памятью.

Характеристики шины

26 января 2015 г.

22:38

Число сигнальных линий шины адреса определяют максимально возможный объем адресного пространства, это одна из базовых характеристик шины. Совокупность линий служащих для пересылки данных называют шиной данных.

Важнейшие характеристики шины данных — это её ширина и пропускная способность. Ширина шины данных определяется количеством битов информации, которое может быть передано по шине за одну транзакцию (цикл шины). Пропускная способность шины данных, это количество единиц информации передаваемых по шине данных в единицу времени.

Разновидности памяти

26 января 2015 г.

22:56

**Таблица 5. Модели памяти**

|  |  |  |  |
| --- | --- | --- | --- |
| Модель | Тип кода | Тип данных | Назначение модели |
| TINY | near | near | Код и данные объединены в одну группу с именем DGROUP.  Используется для создания программ формата .com. |
| SMALL | near | near | Код занимает один сегмент, данные объединены в одну группу с именем DGROUP.  Эту модель обычно используют для большинства программ на ассемблере |
| MEDIUM | far | near | Код занимает несколько сегментов, по одному на каждый объединяемый программный модуль.  Все ссылки на передачу управления — типа far.  Данные объединены в одной группе; все ссылки на них — типа near |
| COMPACT | near | far | Код в одном сегменте;  ссылка на данные — типа far |
| LARGE | far | far | Код в нескольких сегментах, по одному на каждый объединяемый программный модуль |

Оперативная память

26 января 2015 г.

22:59

ОЗУ — предназначено для оперативной записи, хранения и считывания информации (программ и данных), непосредственно участвующей в процессе работы ПК. Главное достоинство оперативной памяти — её высокое быстродействие и возможность обращения к каждой ячейке памяти отдельно (прямой адресный доступ к ячейке). Память называется оперативной потому, что работает так быстро, что процессору не приходится ждать при чтении данных из памяти и записи в неё. При выключении питания ПК вся информация ОЗУ стирается. Объем установленной в компьютере оперативной памяти определяет, с каким программным обеспечением можно на нем работать. При недостаточном объёме оперативной памяти многие программы либо не работают, либо работают медленно.

Иерархия памяти компьютера

26 января 2015 г.

23:02

В большинстве ПК используется иерархическая память. В качестве уровней используется регистры и регистровая память, основная оперативная память, КЭШ память, жёсткие диски, твердотельный накопитель.

Удерживается следующий принцип, при повышении уровня памяти скорость обработки должна увеличиваться, а объем уровня уменьшаться. Эффективность использования такой иерархии достигается за счёт хранения для часто используемых данных памяти верхнего уровня, время доступа к которой минимально.

Принцип иерархической организации памяти был сформулирован в связи с тем, что с самого первого компьютера с сохраняемой программой существовало несоответствие между быстродействием арифметического устройства и оперативной памяти. Противоречия бы не существовало, если выполнить память на тех же элементах, что и арифметическое устройство, но такая память получалась слишком дорогой, кроме того, непомерно увеличивалось количество радиоламп, что заметно снижало надёжность компьютера. Иерархическое построение оперативного запоминающего устройства позволяет иметь быстродействующую память небольшого объёма только для данных и команд, подготовленных к выполнению. Все остальное хранится в запоминающем устройстве более низкого уровня, для этого стали использоваться появившиеся вскоре магнитные носители информации.

Кэш-память

26 января 2015 г.

23:05

Кэш-память — это высокоскоростная память произвольного доступа, используемая процессором компьютера для временного хранения информации. Она увеличивает производительность, поскольку хранит наиболее часто используемые данные и команды «ближе» к процессору, откуда их можно быстрее получить

Кэш-память напрямую влияет на скорость вычислений и помогает процессору работать с более равномерной загрузкой.

Когда приложение начинает работать, данные и команды переносятся с медленного жёсткого диска в оперативную память произвольного доступа (Dynamic Random Access Memory — DRAM), откуда процессор может быстро их получить. Оперативная память выполняет роль кэша для жёсткого диска.

Принципы взаимодействия ассемблерных программ с ОС

26 января 2015 г.

23:07

ОС предоставляет набор услуг по выполнению различных операций, таких как ввод-вывод, работа с файлами и каталогами, работа с памятью и т. д. Обращение из прикладной программы к операционной системе для выполнения какой-либо операции называется запросом к ОС или системным вызовом. После выполнения запроса операционная система возвращает в прикладную программу сведения об успешности выполнения запроса и, если это предусмотрено, результаты его выполнения.

**Общие принципы взаимодействия.** Способ передачи запросов в ОС и получения от неё результатов (интерфейс системных вызовов) зависит от конкретной ОС и является частью системных соглашений. Различные ОС используют разные интерфейсы системных вызовов, но в общем случае последовательность действий программы состоит из следующих операций: подготовка данных, описывающих запрос; передача управления в операционную систему; после возврата управления из операционной системы – анализ успешности выполнения запроса и его результатов.

На сегодняшний день наиболее распространёнными являются две разновидности системных вызовов: программные прерывания и API-функции.

Программные прерывания

26 января 2015 г.

23:15

Данные, составляющие описание запроса, размещаются в регистрах процессора. Для передачи операционной системе данных большого объёма используются указатели, которые размещаются опять-таки в регистрах процессора. Передача управления в операционную систему осуществляется с помощью программных прерываний с определённым вектором, команды которых присутствуют в системах команд подавляющего большинства процессоров. В DOS для этих целей используется команда int с вектором номер 21h.

После возврата управления из ОС с помощью команды возврата из прерывания (или аналогичной ей последовательности команд) признаком неудачного завершения выполнения системного вызова является единичное состояние флага переноса в регистре состояния процессора (в DOS – флаг CF регистра флагов процессора).

Результаты выполнения запроса возвращаются в прикладную программу через регистры процессора (для DOS – чаще всего регистры AX, BX, ES. В случае установленного флага переноса в регистре AX размещается код ошибки). Данный вариант системных вызовов появился в системах с защитой памяти, в которых прикладные программы исполняются в пользовательском режиме и им запрещено обращаться в не принадлежащие им области памяти.

При передаче запроса из программы в ОС требуется передать управление операционной системе, и, следовательно, процессор должен начать выбирать команды из «чужой» области памяти – памяти, принадлежащей ОС. Но аппаратура защиты памяти не разрешает этого. Таким образом, возникает конфликт, хотя, по существу, это обычная ситуация. Для устранения этого конфликта необходимо одновременно с передачей управления менять состояние процессора с пользовательского на системное, в котором снимаются все ограничения. Для этих целей наилучшим решением является прерывание, которое и передаёт управление, и меняет состояние процессора на системное.

API-функции

26 января 2015 г.

23:15

Этот вариант организации системных вызовов на сегодняшний день стал стандартом. Это так называемые API-функции (Application Programming Interface – интерфейс программирования приложений). Наиболее известной в этой сфере является группа стандартов, имеющих общее наименование POSIX (Portable Operation System Interface – переносимый интерфейс операционной системы). Впервые этот интерфейс полностью был реализован в ОС Linux. Данный вариант передачи запросов к операционной системе предполагает оформление системных вызовов в виде функций. Описанием запроса в этом случае выступает имя (идентификатор) функции и передаваемые в неё аргументы. Результаты выполнения запроса возвращаются в прикладную программу в виде результата функции и в случае необходимости через переданные в функцию аргументы (в тех случаях, когда результаты запроса представляют собой некоторую структуру данных достаточно большого объёма).

Фактически API-функции используют стандартные средства языка программирования для вызова процедур и функций, в которых передача аргументов в вызываемые подпрограммы производится через стек, а результат функции возвращается через регистры процессора (чаще всего – регистры AL, AX, EAX, DX, EDX). Если совместить стандартные механизмы работы с процедурами и функциями, используемые в современных языках программирования, с вышеприведённой последовательностью действий прикладной программы для использования услуг ОС, то получается следующий интерфейс системных вызовов.

1. Данные, составляющие описание запроса, размещаются в стеке. Состав этих данных определяется составом аргументов API-функции.
2. Передача управления в операционную систему осуществляется с помощью команды вызова подпрограммы Call, которая присутствуют в системах команд подавляющего большинства процессоров.
3. Возврат управления из ОС производится с помощью команды возврата из подпрограммы (или аналогичной ей последовательности команд).
4. Результаты выполнения запроса возвращаются в прикладную программу через регистры процессора и в случае необходимости через переданные в функцию аргументы, содержащие указатели на переменные с результирующими данными.
5. Признаком неудачного завершения выполнения системного вызова обычно является отрицательное или нулевое значение регистра EAX.

Общие вопросы взаимодействия программ с ОС

26 января 2015 г.

23:18

При разработке больших программ часто используется принцип модульного программирования, который предполагает использование готовых модулей, написанных на разных языках программирования. В такой ситуации возникает ряд вопросов, связанных с согласованием особенностей генерации кодов модулей при трансляции с различных языков

**Согласования имён и атрибутов**

сегмент кода— .CODE, \_TEXT

.DATA, \_DATA

.STACK, \_STACK

Таким образом транслятор сам даёт типовые имена сегментам модуля и присваивает им все необходимые атрибуты. Поскольку разные языки используют различные реализации вызова подпрограмм, то важнейшим вопросом обеспечения корректного взаимодействия разно-языковых модулей является согласование интерфейсов вызовов.

Способы вызова функций:

* CALL NEAR — близкий или внутрисегментный
* CALL FAR — далёкий или межсегментный

Способы возврата из подпрограмм:

* RET NEAR
* RET FAR

Виды вызовов подпрограмм и способы их указания

26 января 2015 г.

23:37

Способы вызова функций:

* CALL NEAR — близкий или внутрисегментный.
* CALL FAR — далёкий или межсегментный.

Отличия близких от далёких вызовов и возвратов заключается в следующем:

1. Близкие вызовы и возвраты изменяют только смещение в сегменте и позволяют передавать управление в пределах сегмента, в этом случае изменяется только содержимое регистра EIP.
2. Далёкие вызовы и возвраты изменяют не только смещение в сегменте, но и сегментную часть адреса и позволяют передать управление между сегментами. В этом случае изменяется не только содержимое EIP, но и содержимое сегментного регистра CS.

Расстояние (NEAR или FAR), используемое в процедуре по умолчанию, определяется текущей выбранной моделью. Для моделей TINY, SMALL и COMPACT по умолчанию процедура будет ближней (NEAR). Для всех других моделей по умолчанию выбирается расстояние FAR. Если не используются упрощенные директивы определения сегментов, то по умолчанию процедура всегда будет ближней (NEAR).

Можно переопределить используемое по умолчанию расстояние, задав нужное расстояние в определении процедуры. Для этого используются ключевые слова NEAR или FAR. Эти ключевые слова переопределяют расстояние, используемое в процедуре по умолчанию, но только для текущей процедуры. Например:

MODEL TINY ; по умолчанию расстояния NEAR  
 ; test1 - это дальняя процедура  
 test1 PROC FAR  
 ; тело процедуры  
 RET ; это будет дальним возвратом:  
 ENDP  
 ; test2 по умолчанию является  
 ; ближней процедурой  
 test2 PROC  
 ; тело процедуры

RET ; это будет ближним возвратом  
 ENDP

Стили вызовов подпрограмм

26 января 2015 г.

23:56

На сегодняшний день стандартом стала передача аргументов через стек. В этом случае рассмотрению подлежат следующие вопросы:

1. Порядок размещения аргументов в стеке
2. Очистка стека

Решение этих двух вопросов образует понятие «Стиль вызова».

|  |  |  |
| --- | --- | --- |
| Стиль | Передача аргументов | Очистка стека |
| Basic | слева направо | вызываемая процедура |
| Fortran | слева направо | вызываемая процедура |
| C/C++ | справа налево | вызывающая |
| Pascal | слева направо | вызываемая процедура |
| stdcall | справа налево | вызываемая процедура |
| Nolanguage | слева направо | вызываемая процедура |

Понятие макросредств

27 января 2015 г.

0:04

Макросредства — это совокупность конструкций языка, заставляющих транслятор генерировать или модифицировать исходный текст программы.

С учётом возможности использования макросредств ассемблерный транслятор можно рассматривать как совокупность двух взаимосвязанных модулей: макропроцессора и транслятора. Макропроцессор — это модуль, который выискивает в тексте программы макросредства и соответствующим образом модифицирует исходный текст программы. Модифицированный текст далее обрабатывается транслятором и в итоге получается объектный код программы.

Примеры макросредств: директивы proc и local.

Все макросредства ассемблера можно поделить на четыре группы:

1. Макрокоманды
2. Блоки повторений
3. Блоки условной трансляции
4. Вспомогательные директивы

Макрокоманды

27 января 2015 г.

0:06

Механизм макрокоманд является наиболее используемым из макросредств. Он позволяет при разработке программы использовать заранее написанные фрагменты текста, имеющие собственные имена. Макрокоманды можно рассматривать как аналогию процедур, но это не совсем корректно. Отличие заключается в том, что выполнение макрокоманд не в процессе выполнения программы, а во время трансляции и сводится в подстановке поименованного фрагмента текста в исходный текст программы. Определения макрокоманды (макроопределение) -- это описание фрагмента текста и присваивание ему некоторого имени.

Используются директивы macro, endm.

Синтаксис macro:

→ имя → macro → [список аргументов] →

Структура macro:

name macro a, b, c

... тело макроопределения ...

endm

name — имя,

a, b, c — аргументы

Тело макроопределения — это любой синтаксически правильный с точки зрения ассемблера текст, в котором используются формальные аргументы.

Определение макрокоманды в тексте должно быть размещено до первого вызова этой макрокоманды.

Примерчик для DOS:

exit macro returncode

MOV AH, 4Ch

MOV AL, returncode

INT 21h

endm

Вызов: exit 0

Вызов макрокоманды — это указание имени макрокоманды с последующим перечислением фактических аргументов.

Расширение макрокоманды (или макрорасширение) — это текст, подставляемый макропроцессором вместо макрокоманды. По сути макрорасширение — это тело макроопределения, в котором формальные аргументы заменены на фактические, указанные в макрокоманде.

Запись в аргументах во a = bx позволяет задать значение формального аргумента по умолчанию, если соответствующий фактический аргумент в вызове опущен.

При использовании макрокоманд часто возникает вопрос дублирования меток, имеющихся в теле макроопределения и появляющихся в расширениях макрокоманд. Для решения этого вопроса нужно выносить имена меток в список директив.

Аппаратная поддержка прерываний

27 января 2015 г.

0:08

Прерывание — это реакция вычислительной системы на некоторое асинхронное событие, которое заключается в том, что выполнение текущей программы временно прекращается и выполняется некоторая другая подпрограмма(обработчик прерываний), после чего чаще всего продолжается выполнение программы.

Прерывания:

* Внутренние
  + Программные
  + От схем контроля
* Внешние
  + От клавиатуры
  + От таймера
  + От HDD
  + …

Программные прерывания вызываются командой INT.

Прерывание схем контроля возникает в тех случаях, когда невозможна корректная работа процессора. Например, прерывание при нарушении защиты памяти. Внешние прерывания происходят по требованию внешних устройств, в тех случаях, когда им требуется обслуживание. Исключением из правил является системный таймер, он используется для отслеживания системного времени и интервалов времени при работе с внешними

устройствами. Вся ОС основана на прерываниях.

Для обеспечения возможности работы с прерываниями аппаратная часть компьютера обязательно содержит соответствующие средства для поддержки прерываний. В реальном режиме работы такими средствами являются:

1. Опрос процессора очередной линии прерывания перед исполнением очередной команды
2. Вектора прерываний
3. INT, INTO, IRET
4. Флаг разрешения прерывания IF
5. Контроллер прерываний
6. Схемы выработки сигналов прерываний во внешних устройствах

Цикл работы процессора:

1. Опрос входного сигнала на прерывания
2. Выбор команды
   1. Чтение кода команды
   2. Получение адреса следующей команды
3. Выполнение команды. Выборка операндов, выполнение операции, запись результата

Векторы прерываний в реальном режиме работы в самом начале оперативной памяти и занимают первые 2^10 байт. Всего векторов 256, от 0 до 255. За каждым вектором закреплены свои прерывания. По своему содержанию, вектор прерываний, это два слова, содержащие адрес обработчика прерывания. В старшем слове располагается сегментная часть адреса, в младшем смещение.

Команда INT вызывает прерывание с вектором, указанным в качестве аргумента команды.

Например, INT 21h, передаёт управление в DOS или INT 17h, INT 13h в BIOS

IRET используется для возврата управления из обработчика прерываний назад в прерванную программу.

Флаг IF в регистре флагов процессора определяет, если IF = 0, то процессор не воспринимает запросы внешних прерываний.

Алгоритм работы драйвера

27 января 2015 г.

0:11

Прерывание от внешних устройств обрабатываются процедурами, физически расположенными в драйверах внешних устройств. Упрощённый алгоритм функционирования драйвера устройства ввода-вывода, работающего по прерываниям, выглядит следующим образом:

1. Вход в драйвер.  
   Когда прикладная программа передаёт запрос на ввод/вывод операционной системе, та, в свою очередь формирует запрос к соответствующему драйверу, заполняет структуру данных описывающую запрос и передаёт управление в драйвер.
2. Инициализация выполнения запроса на внешние устройства.  
   Получив управление, драйвер анализирует запрос (полученную структуру) и подаёт команду внешнему устройству (помещая соответствующие коды в регистры этого устройства). С этого момента внешнее устройство начинает выполнять операцию в соответствии с полученной командой.
3. Разрешение прерывания от внешнего устройства.

В общем случае требует разрешения прерывания на этом внешнем устройстве (записи соответствующего кода в регистр управления) и разрешения прохождения запросов.

1. Выход из драйвера.

После разрешения прерывания драйвер передаёт управление в ОС, чтобы та могла загрузить процессор полезной работой. Например, выполнением одной из прикладных программ, пока внешние устройства выполняют запущенную операцию. Далее, когда внешние устройства заканчивают выполнение операции, выполняющаяся программа прерывается и управление попадает в обработчик прерывания драйвера. Он проверяет успешность выполнения операции к внешним устройствам и, если операция завершилась не успешно передаёт управление в ОС с признаком ошибки, в случае успешного завершения операции выполняется передача данных между внешним устройством и областью памяти прикладной программы, выдавшей запрос. После этого драйвер проверяет объем переданных данных, если он не совпадает с запрошенным объёмом, то подаёт следующую команду внешнему устройству, разрешает прерывание и возвращает управление назад в прерванную программу. Когда очередная команда будет выполнена, тогда произойдёт прерывания и описанный процесс повторится. Однажды обработчик событий обнаружит, что запрос выполнен (переданы все данные предписанные запросом). После чего он запретит прерывания внешнего устройства и передаст управления ОС с признаком успешного завершения выполнения запроса.

Понятие архитектуры вычислительных систем. Архитектура Джона фон Неймана

27 января 2015 г.

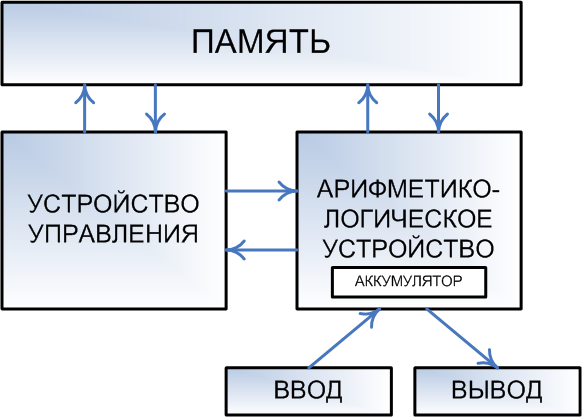
0:16

Архитектура — это описание вычислительной системы на некотором общем уровне, включающую систему команд, средств пользовательского интерфейса, организацию памяти, операции ввода/вывод и управление.

Реализация архитектуры может отличаться на уровне организации подсистем. Те или иные организации архитектуры могут отличаться, как по сложности, так и по стоимости. Детали реализации, невидимые для пользователя, например, кэш-память не оказывают влияние на архитектуру.

Общность архитектур разных вычислительных систем обеспечивает их совместимость с точки зрения пользователя. В контексте разработки вычислительной системы и проектирования её аппаратных средств, термин «Архитектура» используют для описания действия, организации конфигурации основных функциональных блоков. Описание архитектуры должно включать разъяснения принципа действия и диапазон возможностей любого узла вычислительной системы.

Сначала предполагалось, что все устройства работают последовательно, т.е. в ней допускалась одновременная работа двух устройств.



Структурный подход предполагает, что вычислительная система состоит из вышеперечисленных устройств. А программа и данные в памяти располагаются вместе. В ЦП устройство управления определяет, какие действия необходимо выполнить АЛУ, такое определение действий происходит путём определения адреса и считывания команд из выделенной ячейки оперативной памяти. ЦП имеет память в виде накопителей и регистров.

Программа фон Неймоновской модели состоит из набора команд, которые проверяются одна за другой и выполняются. В архитектуре современных ПК сохранены и детализированы основные принципы фон Неймоновской модели вычисления.

При этом современные ПК имеют следующие системные особенности:

1. Процессор совместимый с x86 архитектурой.
2. Единую систему распределения пространства памяти.
3. Унифицированное распределение адресов пространства ввода/вывода с фиксированным положением портов, систему аппаратных прерываний, позволяющей периферийным устройствам формировать сигнал, по которому процессор определённым образом реагирует на события, связанные с этим устройством.
4. Набор системных устройств и интерфейсов ввода/вывода, унифицированным по конструктивно-механическим и электрическим параметрам, типам расширений (шины ISA, PCI, PCI-Express, AGP, USB).

**Принцип однородности памяти**

Команды и данные хранятся в одной и той же памяти и внешне в памяти неразличимы. Распознать их можно только по способу использования; то есть одно и то же значение в ячейке памяти может использоваться и как данные, и как команда, и как адрес в зависимости лишь от способа обращения к нему. Это позволяет производить над командами те же операции, что и над числами, и, соответственно, открывает ряд возможностей. Так, циклически изменяя адресную часть команды, можно обеспечить обращение к последовательным элементам массива данных. Такой прием носит название модификации команд и с позиций современного программирования не приветствуется. Более полезным является другое следствие принципа однородности, когда команды одной программы могут быть получены как результат исполнения другой программы. Эта возможность лежит в основе трансляции — перевода текста программы с языка высокого уровня на язык конкретной вычислительной машины.

**Принцип адресности**

Структурно основная память состоит из пронумерованных ячеек, причем процессору в произвольный момент доступна любая ячейка. Двоичные коды команд и данных разделяются на единицы информации, называемые словами, и хранятся в ячейках памяти, а для доступа к ним используются номера соответствующих ячеек — адреса.

**Принцип программного управления**

Все вычисления, предусмотренные алгоритмом решения задачи, должны быть представлены в виде программы, состоящей из последовательности управляющих слов — команд. Каждая команда предписывает некоторую операцию из набора операций, реализуемых вычислительной машиной. Команды программы хранятся в последовательных ячейках памяти вычислительной машины и выполняются в естественной последовательности, то есть в порядке их положения в программе. При необходимости, с помощью специальных команд, эта последовательность может быть изменена. Решение об изменении порядка выполнения команд программы принимается либо на основании анализа результатов предшествующих вычислений, либо безусловно.

**Принцип двоичного кодирования**

Согласно этому принципу, вся информация, как данные, так и команды, кодируются двоичными цифрами 0 и 1. Каждый тип информации представляется двоичной последовательностью и имеет свой формат. Последовательность битов в формате, имеющая определенный смысл, называется полем. В числовой информации обычно выделяют поле знака и поле значащих разрядов. В формате команды можно выделить два поля: поле кода операции и поле адресов.

Классификация параллельных вычислительных систем по Флинну

27 января 2015 г.

0:22

Различные способы реализации параллельных вычислений можно представить, как способы организации одновременного воздействия одного или нескольких потоков команд на один или несколько потоков данных.

Все ВС могут быть разбиты на 4 класса:

1. Система с одиночным потоком команд и данных (SISD — Single Instruction Single Data)
2. Система с множественным потоком команд и одиночным потоком данных (MISD — Multiple Instruction Single Data)
3. Одиночный поток команд и множественный поток данных (SIMD). В этой системе один поток команд, но эти команды являются векторными. Каждый элемент вектора рассматривается как компонента отдельного потока данных. Таким образом одновременно обрабатывается множество потоков данных. Одна команда может выполняться над массивом данных.
4. Множественный поток команд и множественный поток данных (MIMD — Multiple Instruction Multiple Data). Подразумевает наличие нескольких процессорных устройств и, следовательно, несколько поток данных. Примерами таких устройств являются мультипроцессорные матрицы.

MISD широкого распространения не получила.