Лекции по дисциплине «Архитектура Вычислительных процессов»

Преподаватель: Осипов Олег Васильевич

(7 лаб, Экзамен)

6 сентября

Понятия ВС. Архитектура Джона Фон Неймана. Архитектура.

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 и проектирования ее аппаратных средств термин «Архитектура» используют для описания действия, организациии 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.
(1)
Структурный предполагает, что вычислительная система состоит из выше перечисленных устройств. А программа и данные в памяти располагаются вместе. В **ЦП (центральный процессор)** устройство управления определяет, какие действия необходимо выполнить **АЛУ (арифметико-логическое устройство),** такое определение действий происходит путем 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 (персональных компьютеров) сохранены и деитализированны 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 переферийным устройствам формировать сигнал по которому процессор определенным 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)

Современное состояние и тенденции развития вычислительной техники.

1) Создание унифицированных модульных архитектур. Позволяющих поэтапно расширять вычислительные мощности, при этом акцент делается на **магистрально-модульное**

построение, когда к общей магистрали наряду с общими системами подключаются подсистемы.
2) Ускорение развития систем параллельной обработки путем создания новых многопроцессорных систем и машинных комплексов.
3) Постепенный переход на вычислительные системы распределенной обработки и децентрализованное управление ими.
4) Широкое внедрение архитектуры вычислительных систем типа **RISC** (архитектура с сокращенным набором команд), данная архитектура обеспечивает более простые машинные команды выполняемые за один такт.
5) Создание **одно-кристальных компьютеров,** ориентированных на один язык программирования.
6) Создание технологий программирования, отличающихся производительностью и надежностью.

Виды ВС и способы их реализации.

Любая вычислительная система включает в себя технические средства и программное обеспечение ориентированные на решение определенных задач. Такая ориентация может осуществляться, как за счет программных средств и возможностей ОС (операционная система), так и за счет специализированных ВС:

1) Вычислительные системы рассчитанные на числовую обработку
    1) Персональные ЭВМ (производительность порядка 50-300 Гфлопс (FLops) (количество операций с плавающей запятой в секунду))
    2) Специализированные ЭВМ для числовой обработки (ВС управления радиолокационными станциями, суперкомпьютеры, матричные процессоры)
2) Ориентированные на обработку сигналов
    1) Процессоры быстрого преобразования Фурье
    2) Универсальные (сигнальные)
    3) Спеиализированные БИС для цифровой обработки
3) ВС ориентированные на обработку символов (RISC и Prolog машины)
4) Ориентированные на обработку больших объемов данных (машины БД и Баз знаний)
5) Ориентированные на обработку в сетях связей

Внедрение специализированных процессоров в различных предметных областях привело к появлению распределенных ВС.

Классификация ВС по Флинту.

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 (системы):**

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.

Принципы функционирования ЦП.

Машинная команда — это описание операции, которую должна выполнить ЭВМ. Содержит выполняемые операции: указания по определению операндов и их адресов, типы размещения результатов выполненной работы. Команды часто разделяются на арифметические, логические, команды пересылки данных, команды ввода/вывода в порты и команды вызова прерываний.
Совокупность выполняемых машиной операций называется **системой команд.**
Машинное слово — это вектор битов, используемых аппаратной частью ЭВМ как единое целое.
Операнды — это части машинной команды, которые определяют объекты над которыми выполняются операции.
Регистр — это совокупность элементов принимающих значения 1 или 0 используемых для хранения информации. Часто регистры имеют тот же размер, что и машинные слова.
Счетчик команд (IP-регистр) — это регистр процессора содержащий адрес текущей выполняемой команды.
Регистр **ax** имеет размерность 16 бит.
(2)
Регистры:
1) AX - называют аккумулятор
2) BX - базовый регистр
3) CX -
4) DX -

Сегментные регистры: CS (указатель на сегмент кода), DS (указатель на сегмент данных), SS (указатель на сегмент стека), ES (Дополнительный регистр). Эти регистры трогать не нужно.

1) CS - Указатель на кодовый сегмент. Пара CS:IP - однозначно определяет реальный физический адрес следующей выполняемой инструкции (CS + IP = физический адрес)

2) **EAX**(аккумулятор), **BAX**(базовый), **ECX**(счетчик), **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 пара CS:IP всегда указывает на следующую подлежащую исполнению инструкцию. Регистр флагов, флаг - это бит, принимающий значение единица, если выполнена некоторое условие, и значение 0 в противном случае. Процессор имеет регистр флагов содержащий набор флагов отражающие текущее состояние процессора.

Указатель на сегмент стека **SS:SP**
Указатель на сегмент данных **DS**
Указатель на сегмент кода **CS:IP**
Указатель на дополнительный сегмент **ES**

Сегментом называется область, которая начинается с адреса кратного 16. При выполнении программы определяется 3 главных сегмента.

1) Сегмент кода (содержит машинные команды, обычно первая исполняемая команда находится в начале этого сегмента и операционная система передает 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 PUSH, 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.

Основные команды:

1) Команды пересылки данных
    1) MOV <операнд назначения>, <операнд источник>
    2) XCHG - меняет регистры местами
    3) MUL - команда умножения
    4) DIV - команда деления
    5) ADD - команда сложения
    6) SUB - команда разности
    7) JMP, LOOP, RET и другие
    8) INT - команда вызова прерываний
    9) IN, OUT - команды ввода/вывода в порты
    10) CMP (сравнение), JE, JZ - команды условного перехода

Пример программы:

```
A1:   ADD AX, 01
      ADD BX, AX
      SUB CX, 1
      JMP A1
```

```
Команда цикла:
      MOV AX, 1
      MOV CX, 12
C1:   ADD AX, 10
      LOOP C1
```

Цикл выполнения программы

Это время выполнения для одной машинной команды. Цикл выполнения команды реализуется за несколько машинных циклов, 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 ее в регистр
2) Выбрать операнды из памяти, если в команде используются операнд расположенный в оперативной памяти, то блок управления начинает операцию по выборке его из памяти.
3) Увеличение IP
4) Выполнение команды

Различают следующие основные **частоты системной шины:**

1) Системная (Host Bus Clock) — внешняя частота шины процессора, это частота является опорной для всех остальных компонентов.
2) Внутренняя частота процессора (CPU Clock) — на котором работает его вычислительное ядро

Отношение внутренней частоты к внешней называют **коэффициентом умножения.** Частота шины PCI обеспечивает соединение внешней частоты процессора на различные коэффициенты 2 или 3.

Частота шины ISA Bus Clock должна быть близка к 8MHz, на ней работают все приемо-передающие элементы.
Главной отличительной способностью многопроцессорной системы является ее производительность. Т.е. количество операций выполняемых в единицу времени. Различают пиковую и реальную производительности. Под пиковой понимают величину равную произведению пиковой производительности одного процессора на число процессоров в данной машине. Это характеристика является базовой. По которой производят сравнение вычислительных систем. Чем больше пиковая производительность, тем быстрее пользователь может решить свою задачу. Пиковая производительность — величина теоретическая и не достижимая при запуске конкретного приложения.

Существует два способа ее оценки:
1) Опирается на число команд выполняемых за единицу времени. Единицей измерения является MIPS (Millions instructions per second). Но производительность выраженная в MIPS. Данная характеристика дает самое общее представление о производительности.
2) Заключается в определении числа вещественных операций (FLOPS). Такой способ является более приемлемым для пользователя. Пользуясь этой характеристикой пользователь может получить оценку времени. Однако пиковая производительность получается только в идеальных условиях. Т.е. при отсутствии конфликтов и при равномерной загрузке всех устройств, в реальных условиях на выполнение реальной программы влияют такие аппаратно программные особенности, как:
   a. Особенности структуры процессора
   b. Системы команд
   c. Состав устройств
   d. Реализация ввода/вывода
   e. Эффективность работы компилятора

Время взаимодействия с памятью

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

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

Тесты используемые при оценке производительности

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 500x500. 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 операци: копирование в памяти, умножение на константу, умножение и сложение

Адресация данных и команд в ассемблере

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 физический адрес не совпадает с адресным полем команды, но зависит от него. В общем случае происходит преобразование из адресного кода в физический код. Способы адресации являются одним из основных архитектурных признаков. В настоящее время известно более двух десятков различных способов адресации и их модификации. Все известные способы.

1) Прямые (
   Либо накопительный адрес операнда, либо сам операнд находится непосредственно по адресному коду без всякого преобразования
)
2) Не прямые (
   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)
8. (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 здвига. В случае двумерного массива базовый адрес может содержать начальный адрес массива, а значение вида и индексного регистра могут содержать смещение по строке и столбцу)

Архитектура многопроцессорных ВС

SMP (Symmetric MultiProcessing).

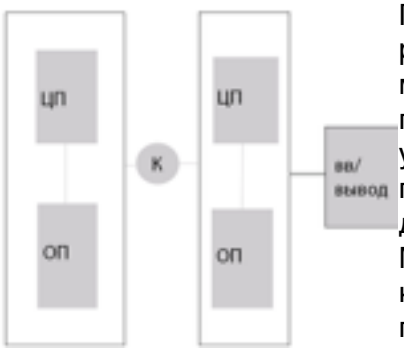
Главной особенностью систем с данной архитектурой является наличие общей физической памяти разделяеймой всеми процессами.

Память служит для передачи сообщений между процессами. При этом все вычислительные устройства при обращении к ней имеют равные права, поэтому SMP архитектуру называют симметричной. SMP система строится на основе высоко скоростной системной шины к слотам которой.

Основные преимущества:
1) Простота и универсальность для программирования. Архитектура SMP не накладывает ограничений и обычно используется модель, когда все процессоры работают независимо друг от друга. Однако, можно реализовать и межпроцессорный обмен с использованием общей памяти. Пользователь также имеет доступ ко всему объему памяти.
2) Простота эксплуатации. Как правило SMP системы используют систему кондиционирования основанную на воздушной охлаждении, что облегчает их техническое обслуживание.
3) Относительно не высокая цена.

Недостатки:
1) Системы с общей памятью плохо масштабируются, этот существенный недостаток SMP систем не позволяет считать их по настоящему перспективными. Причиной плохой масштабируемости является то, что в данный момент шина способна обрабатывать только одну транзакцию. В следствии чего возникают проблемы разрешения конфликтов при одновременном обращении к памяти. Когда произойдет такой конфликт зависит от скорости связи и количества процессоров. Кроме того, системная шина имеет ограниченную пропускную способность и ограниченное число слотов. Все это препятствует увеличению производительности при увеличении числа процессоров. В реальных системах можно задействовать не более 32 процессоров. При работе с SMP системами используют так называемую парадигму программирования с разделяемой памятью.

MPP (Massive Parallel Processing)



Главнаая особенность, в том, что память физически разделена. В этом случае система строится из отдельных модулей содержащих процессор, локальный банк оперативной памяти, сетевые адаптеры, жесткие диски и другие устройства ввода/вывода. Такие модули представляют собой полнофункциональные компьютеры. Доступ к банку данных из данного модуля имеют только процессоры из этого же модуля. Модули соединяются специальными комуникационными каналами. Пользователь может определить логический номер процессора, к которому он подключен.

Используется два варианта работы ОС на машинах MPP архитектур:
1) Полноценная ОС работает только на управляющей машине, на каждом отдельном модуле функционирует урезанный вариант ОС
2) На каждом модуле работает полноценная операционная система.

Главным преимуществом систем с разделяемой памятью является хорошая масштабируемость. В отличии от SMP систем в машинах с раздельной памятью каждый

процессор имеет доступ только к своей локальной памяти в связи с чем не возникает необходимости по тактовой синхронизации процессора. Практически все рекорды производительности на сегодня устанавливаются на машинах именно такой архитектуры состоящих из нескольких тысяч процессоров.

Недостатки:

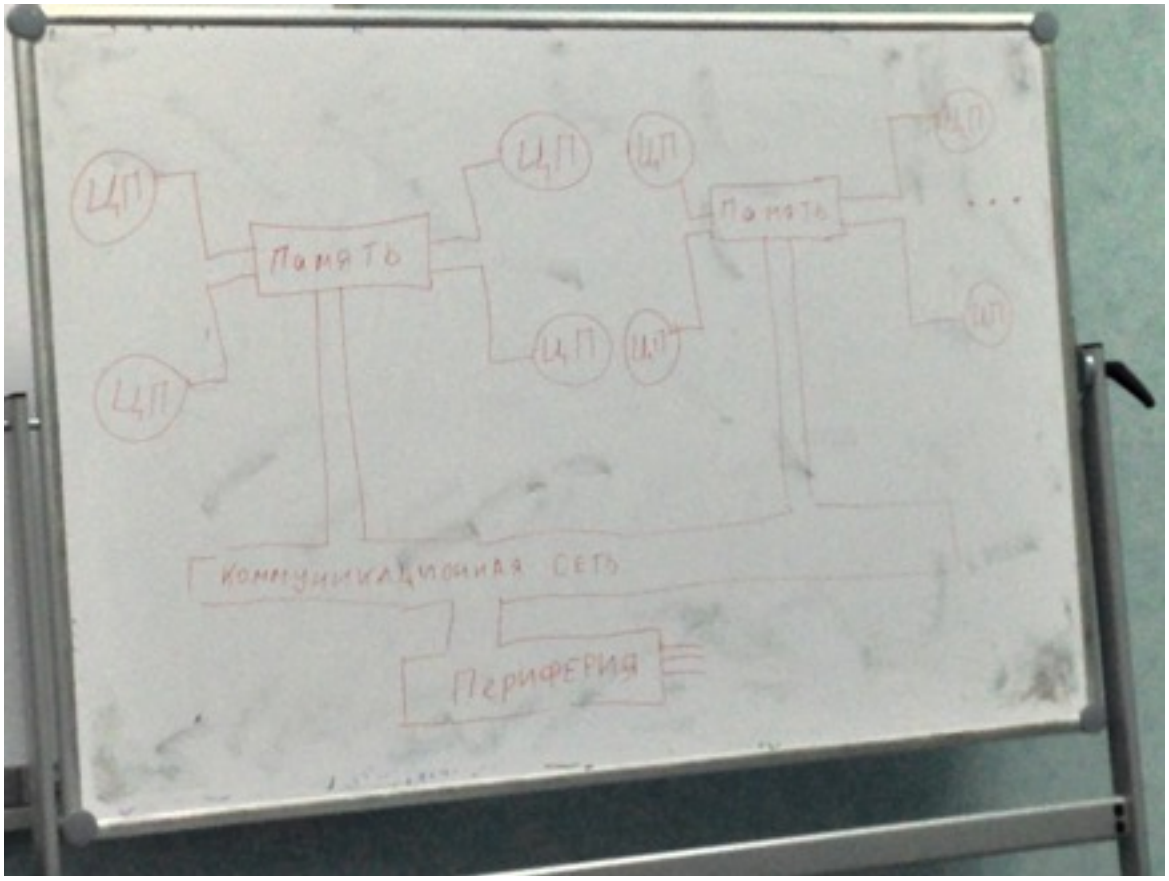
1) Отсутствие общей памяти заметно снижает скорость межпроцессорного обмена, поскольку нет общей среды для хранения данных.
2) Каждый процессор может использовать только ограниченный объем локального банка памяти
3) В следствии указанных архитектурных недостатков требуются значительные усилия для того, чтобы максимально использовать системные ресурсы. Этим объясняется высокая цена программного обеспечения для систем с раздельной памятью.

При работе с MPP системами используют так называемую парадигму программирования с передачей данных.

Гибридная архитектура

NUMA (Nuniform Memory Access) — неоднородный доступ к памяти.

Гибридная архитектура совмещает достоинства систем с общей памятью и относительную дешевизну систем с раздельной памятью. Суть этой архитектуры в особой организации памяти, а именно память физически распределена по различным частям системы, но логически она является общей так, что пользователь видит единое адресное пространство. Система построена из однородных базовых модулей состоящих из небольшого числа процессоров и модуля памяти. Модули объединены с помощью высокоскоростного коммутатора. Поддерживается единая адресное пространство, так же аппаратно поддерживается доступ к удаленной памяти, т.е. к памяти других модулей.



PVP - Parallel Vector Process (Параллельная архитектура с векторными процессорами)

Основным признаком PVP систем является наличие специальных векторно-конвейерных процессоров, в которых предусмотрены команды однотипной обработки векторов. Как правило несколько таких процессоров работают одновременно с общей памятью поскольку передача данных в векторном формате осуществляется намного быстрее, чем скалярное, то проблема взаимодействия между потоками становится несущественными.

Примеры машин с PVP архитектурой:

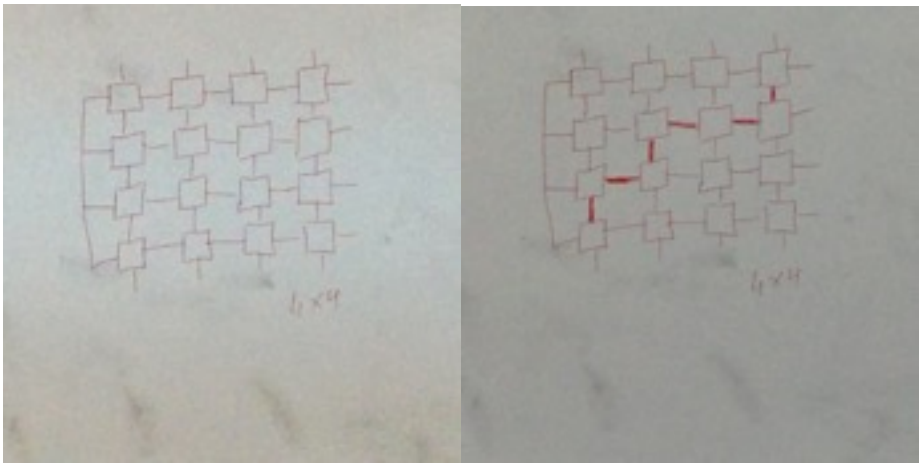
CRAY-X
NEC SX-6
Fujitsu VPP50

Кластерная архитектура

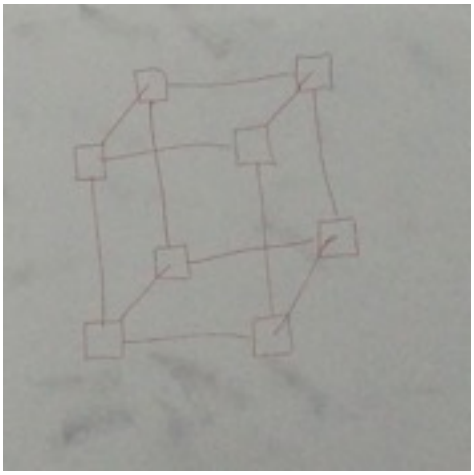
Кластер представляет собой два или более компьютеров часто называемых узлами объединенные при помощи сетевых технологий на базе шинной архитектуры или коммутатора и предстающая перед пользователями в качестве единого информационно-вычислительного ресурса. В качестве узлов кластера могут быть выбраны серверы, рабочие станции или даже обычные ПК.
Преимущества кластеризации для повышения работоспособности становится очевидным в случае сбоя какого-либо узла, при этом другой узел кластера может взять на себя нагрузку неисправного узла и пользователь не заметит прерывания. Такие системы являются очень дешевые.
Архитектура кластерный системы, т.е. способ соединения компьютеров в большей степени определяет ее производительность, чем тип используемых в ней процессоров.

Критическим параметром влияющим на величину производительности такой системы является расстояние между процессорами. Так, соединив вместе 10 ПК мы получим систему для проведения высокопроизводительных вычислений. Проблема будет состоять в поиске наиболее эффективного способа соединения стандартных устройств друг с другом, посколько при увеличении производительности каждого из процессоров в 10 раз, производительность системы в 10 раз не увеличится.

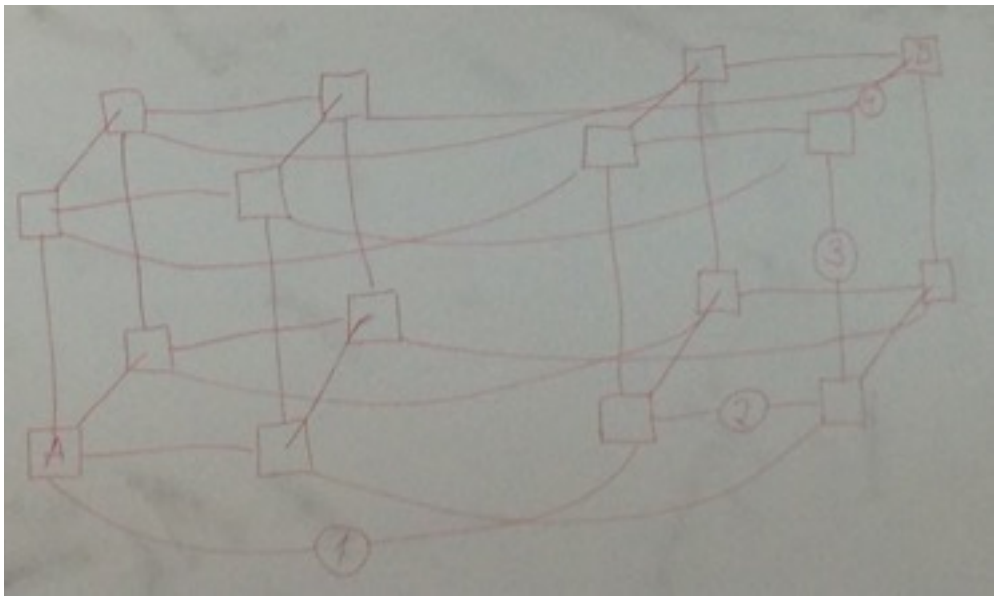
Для примера рассмотрим задачу построения 16 процессорной системы, в которой все процессоры были бы равноправны. Наиболее естественным представляется соединение в виде плоской решетки, где внешние концы используются для подсоединения внешних устройств. При таком типе соединения максимальное расстояние между процессорами равняется **шести**.



Теория же показывает, что если в системе максимальное расстояние между процессорами больше четырех, то такая система не может работать эффективно, поэтому при соединении 16 процессоров друг с другом плоская схема является не целесообразной. В таких случаях используется система типа **куб**, если число процессоров равно 8 или гиперкуб, если число процессоров больше 8.

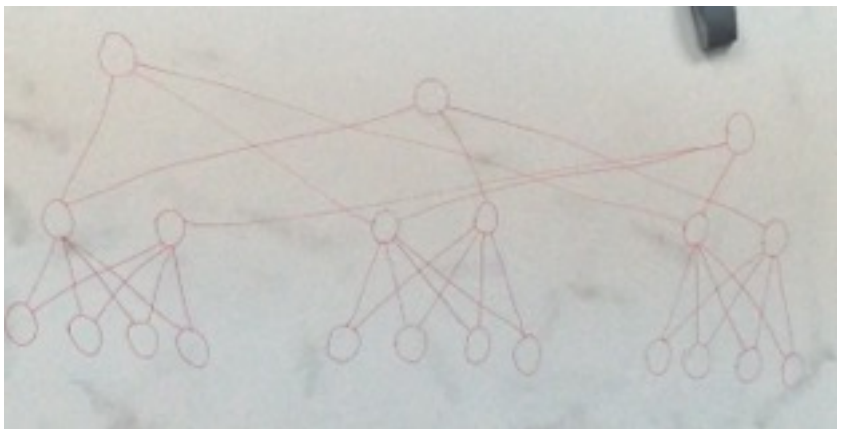


При соединении 16 процессоров потребуется четырех мерный гиперкуб. Для его построения следует взять обычный трехмерный куб, сдвинуть в нужном направлении и соединив вершины получить гиперкуб размерности четыре.



Наиболее эффективным в кластерный системах является архитектура с технологией **tack track**. Процессоры локализованы в листьях дерева, внутренние узлы дерева скомпонованы во внутреннюю сеть. Поддеревья могут общаться между собой не затрагивая более высоких уровней.

Поскольку способ соединения процессоров больше влияет на производительность кластера, чем тип используемых процессоров, то может оказаться более целесообразным создать систему из большего числа дешевых компьютеров, чем из меньшего числа дорогих.



Основные принципы RISC архитектуры

Отличительные особенности **RISC** (Reduced Instruction Set Computer) и **CISC** (Complex …). Двумя основными архитектурами набора команд используемыми компьютерной промышленностью на современном этапе развития вычислительной техники являются архитектуры RISC и CISC.
Основоположником является фирма IBM. IBM с её базовой архитектурой IBM 360 ядро которой используется с 1964 года.

RISC использует сравнительно не большой наиболее употребим команд определенные в результате статистического анализа. В компьютерной индустрии наблюдается настоящий бум с RISC архитектурой. Рабочие станции и серверы созданы на основе концепции RISC. Они завоевали лидирующие позиции благодаря своим исключительным характеристикам.

Четыре основных принципа RISC архитектур

1) Каждая команда независимо от ее типа выполняется за один машинный цикл, длительность которого должна быть максимально короткой.
2) Все команды должны иметь одинаковую длину и использовать минимум адресных форматов, что резко упрощает логику центрального управления процессором.
3) Обращение к памяти происходит только при выполнении операции записи и чтения. Вся обработка данных происходит исключительно в регистровой структуре процессора.
4) Система команд должна обеспечивать поддержку к языкам высокого уровня. Имеется ввиду подбор системы команд наиболее эффективный для различных языков программирования

Простота архитектуры RISC процессора обеспечивает его компактность и практическое отсутствие проблем с охлаждением кристалла, чего нет в процессорах Intel. На текущий день RISC и CISC процессоры практически монопольно занимают на компьютерном рынке сектор ПК. Однако, RISC процессорам нет равных в секторе высокопроизводительных серверов.

CISC	RISC
Многобайтовый команды	Однобайтовый команды
Малое количество регистров	Большое количество регистров
Сложные команды	Простые команды
Одна и менее команд за один цикл процессора	Несколько команд за один цикл процессора

Вместе с тем опора на регистры является ахиллесовой петой RISC архитектур. Проблема в том, что во время выполнения задачи RISC система вынуждена неоднократно обновлять содержимое регистров процессора, причем за минимальное время, чтобы не вызвать простоев арифметического устройства.
Для CISC систем подобной проблемы не существует, поскольку модификация регистров может происходить на фоне обработки команд память в память.

25 октября

Эталонная модель сети

В 1979 году была создана эталонная модель взаимодействия открытых систем OSI (Open System Interconnection). Обычно называемая 7-ми уровневой моделью. Функции взаимодействия делятся на ряд слоев именуемых уровнями. Все уровни модели взаимодействуют по иерархической системе, т.е. каждый уровень обслуживает уровни находящиеся выше него и пользуется услугами уровней ниже него.

Правила взаимодействия объектов одноименных уровней различных систем называют **протоколами**. А правила взаимодействия смежных уровней одной и той же системы между уровнем и интерфейсом называют междууровневым интерфейсом.

Уровни:

	Взаимодействующая машина	**Физическая среда**	**Взаимодействующая машина**
Информационные уровни	Прикладной	- - - - - - - - - - - - - - - - -	
	Представительский	- - - - - - - - - - - - - - - - -	
	Сеансовый	- - - - - - - - - - - - - - - - -	
Транспортный уровень	Транспортный	Протоколы	
Коммуницирующие уровни	Сетевой	- - - - - - - - - - - - - - - - -	
	Канальный	- - - - - - - - - - - - - - - - -	
	Физический	- - - - - - - - - - - - - - - - -	

Физический уровень — обеспечивает интерфейс между машиной участвующей во взаимодействии и средой передачи сигналов.

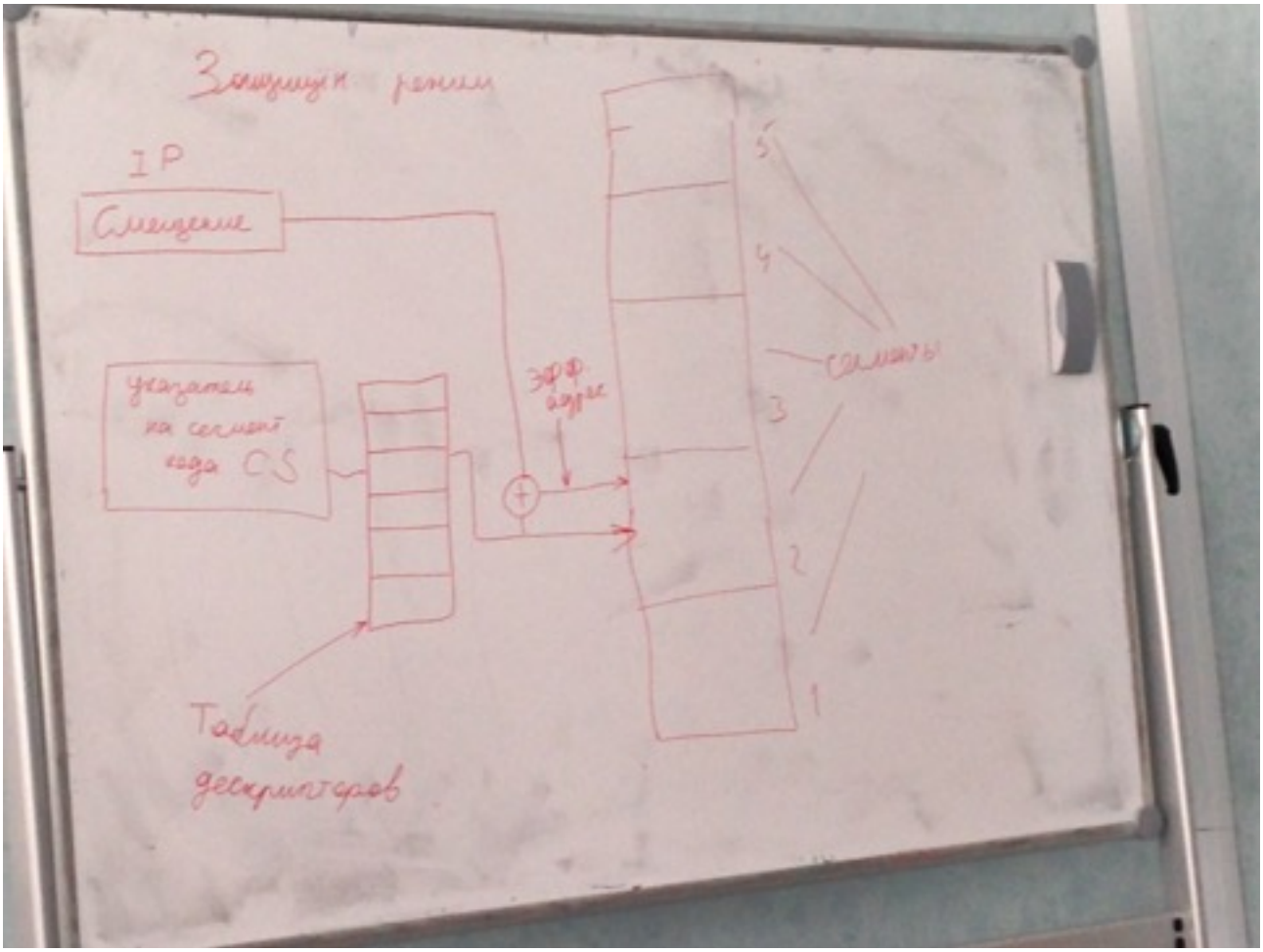
Канальный уровень — обеспечивает средства установления, поддержания и освобождение линии передачи данных связывающих системы друг с другом. Он формирует из данных формируемых первым уровнем так называемые кадры или последовательности кадров. Осуществляя доступ к среде используемой несколькими машинами обнаруживает и исправляет ошибки.

Сетевой уровень — реализует функции маршрутизации, чтобы кадры второго уровня называемые пакетами могли передаваться через несколько каналов по одной или нескольким сетям. Обычно, это требует включения в пакет сетевого адреса.

Транспортный уровень — обеспечивает взаимодействие процессов подключенных к машинам и сквозное управление движением пакетов между этими процессами.

Сеансовый уровень — организует и поддерживает сеанс взаимодействия, т.е. диалог между прикладными процессами определенного типа, обрабатываемыми на разных компьютерах.

Представительский уровень — уровень представления данных. Осуществляет интерпретацию данных (форматов, кодов, структур, команд) передаваемых во время



диалога между прикладными процессами.

Прикладной уровень — реализует все функции, которые необходимы для обслуживания прикладных процессов и которые не были реализованы всеми предыдущими уровнями.

Прикладным процессом называют основной компонент системы осуществляющий обработку информации для пользователей. Прикладной процесс является либо источником, либо потребителем информации.

Сегментированная модель памяти

Физическая память, к которой ЦП имеет доступ по шине адреса называется ОЗУ (Оперативно-Запоминающее Устройство). Каждому байту ОЗУ соответствует свой уникальный адрес называемый «**физический**». Диапазон значений физических адресов зависит от размерности шины адреса процессора. Для x86 от 0 до 4 Гб, современные ПК адресуют 64-128 Гб. Механизм управления памятью полностью аппаратный, что означает, что программа не может сама сформировать физический адрес памяти на адресной шине.

Сегментация — это механизм адресации обеспечивающий существование нескольких адресов пространств, как в приделах одной задачи так и системы в целом для защиты задач от взаимного влияния. В основе механизма сегментации лежит понятие **сегмент**а, который представляет собой блок памяти. Каждая программа в общем случае может состоять из любого состояния сегментов. Непосредственно доступ она имеет только к

трем основным сегментам стека, данных и кода, а так же к дополнительным сегментам данных, число которых от 1 до 3.
Программа заранее не знает по каким физическим адресам будут размещены ее сегменты, этим занимается ОС. ОС размещает сегменты программы в памяти по определенным физическим адресам, после чего помещает значения этих адресов в определенные места куда именно зависит от работы ЦП. В реальном режиме эти адреса заносятся непосредственно в сегментные регистры, а в защищенном они попадают в элементы специальной системной таблицы дескрипторов.

Режимы работы процессора

Существует 3 режима:
- Реальный
- Защищенный
- Виртуальный

Реальный режим — это режим работы первых 16 бит регистров процессоров. Наличие его обусловленно тем, что необходимо обеспечить в новых моделях процессоров функционирования программ разработанных для старых моделей.

Защищенный режим — это режим, в котором вычисления могут быть защищены программно аппаратным путем позволяя использовать все возможности. Все современные многозначные ОС работают в данном режиме. Режим создан для защиты задач от взаимного влияния.

Виртуальный режим — переход в этот режим возможен, если процессор уже находится в защищенном режиме. Физический адрес формируется по правилам реального режима внутри сегмента программа обращается к адресам относительно начала сегмента (начиная с 0, заканчивая адресом равным размеру сегмента). Этот относительный адрес или смещение, который ЦП использует для доступа к данным внутри сегмента называется эффективным.

Реальный режим отличается от защищенного, что в между CS и сегментами нет дескрипторов

stdcall — означает, что стек подчищает за собой аргументы вызываемых функций

Для crt_sscanf надо подключить msvcrt.inc и msvcrt.lib

Сплошная модель памяти (Flat Model)

При использовании сплошной модели памяти программа оперирует единым, непрерывным адресным пространством. В нем содержатся и код, и стек и данные программы, адрессуемые смещения от 0 до 2^32 - 1 — для 32 разрядных систем, такое 32 битное смещение называют линейным адресом. Для 16 битных процессоров плоская модель позволяет адресовать 64 Кб

2^16 байт = 65536 байт = 64 Кб
2^32 байт = 4Гб
2^64 байт = 256 Тб

Управление памятью реализуется на основе плоской модели в целях содействия функциональности ОС, защиты ресурсов, многозадачности или увеличения объема памяти за пределы ограничений налагаемым физическим адресным пространством процессора.

Преимущества управления памятью с плоской моделью

1) В многозадачных приложениях, 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 обеспечивает максимальную гибкость для реализации управления этого типа памятью.

   Фотка

Использование сегментированной модели

При использовании сегментированной модели, 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 определенный сегмент, а смещение указывает на конкретный байт в адресном пространстве сегмента. 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.

При использовании плоской или сегментированной модели, линейное адресное пространство может быть отражено либо непосредственно (линейный адрес — физический адрес), либо с помощью механизма страничной трансляции. Во втором случае линейное адресное пространство делится на страницы (обычно 4 кб), которые составляют виртуальную память.

Страничная трансляция обеспечивает отображение требуемых страниц виртуальной памяти в физическое адресное пространство, следует отметить, что сплошная модель реализуется как частный случай сегментированной модели, когда программа обращается к сегменту под который отведено все линейное пространство.

Шины

Компьютерные шины (computer bus) — служат для передачи данных между отдельными функциональными блоками компьютера и представляют собой совокупность сигнальных линий, которые имеют определенные 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 прием 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**.

Выделяют следующие типы шин:

1) Шина процессор - память (или шина переднего плана (Front-Side bus, FSB)). Шина 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 разделенных 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 перешлет данные. Подключение к шине большого количества устройств ведет к падению ее 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 высоко производительное оборудование типа видеоадаптеров, дисковых накопителей и сетевых адаптеров с материнской платой, центральным процессором и оперативной памятью.
9) Технические шины PCI представляют собой шину расширения, а не настоящую локальную шину

Физические шины реализованы в виде параллельных медных проводников материнской платы, поперек основной шины материнской платы установлены разъемы адаптеров расширений внешних устройств ввода/вывода. Механические спецификации шин включают в себя такие характеристики такие как, размеры и размещения направляющих для их установки, разрешенное место для размещения кабельного разъема и т.д. Устройства использующие шину электрически подсоединены к ее 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 приемниками (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.
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 ее 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.

Последовательность операций происходящих на шине данных в процессе одной транзакции включают в себя:

1) Получение управления шиной данных передающим устройствам
2) Выставление данных на шину
3) Появление данных на адресуемом устройстве
4) Считывание данных адресуемых устройством

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**.

Понятие прерываний

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:

1) Внутренние
    1) Программные
    2) От схем контроля
2) Внешние
    1) От клавиатуры
    2) От таймера
    3) От HDD
    4) …

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 средствми 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
   a. Чтение кода команды
   b. 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, передает 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.

Состав и размещение обработчика прерываний

ПО системы прерываний образуют совокупность всех обработчиков прерываний, точнее та ее часть, которая занимается выполнением запросов прикладных программ.

Прерывание от схем контроля обрабатываются процедурами ОС типичным их действием является принудительное аварийное завершение программы, во время исполнения которой возникло прерывание сопровождающееся сообщением об ошибке

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

1) **Вход в драйвер**
   Когда прикладная программа передает запрос на ввод/вывод операционной системе та в свою очередь формирует запрос к соответствующему драйверу, заполняет структуру данных описывающую запрос и передает управление в драйвер.
2) **Инициализация выполнения запроса на внешние устройства**
   Получив управление, драйвер анализирует запрос (полученную структуру) и подает 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.
4) **Выход из драйвера**
   После разрешения прерывания драйвер передает 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 передает 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

объемом, то подает следующую команду внешнему устройству, разрешает прерывание и возвращает управление назад в прерванную программу. Когда очередная команда будет выполнена, тогда произойдет 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.

Принципы взаимодействия ассемблерных программ с ОС

Большинство прикладных программ в своей работе не обходятся без услуг ОС. ОС предоставляют набор услуг по выполнению различных операций таких как: ввод, вывод, работа с файлами, памятью и т.д. Чем обширнее набор таких сервисов и чем они крупнее, тем проще разрабатывать программы. Обращение из прикладной программы к ОС для выполнения какой либо операции называется **запросом к ОС** или **системным вызовом**. После выполнения запроса ОС возвращает в прикладную программу сведения об успешности выполнения запроса и результат его выполнения. Способ передачи запросов в ОС и получение от нее 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:

1) Подготовка данных описывающих запрос
2) Передача управления в ОС
3) После возврата управления из ОС — анализ успешности выполненного запроса и его результата

Наиболее распространенными являются **2 разновидности системных вызовов:**

1) **Программные прерывания (используются в DOS)**
   Основные особенности:
   a) Данные, составляющие описание запроса размещаются в регистрах процессора. Для передачи ОС данных большого объема используются указатели, которые размещаются так же в регистрах процессора
   b) После возврата управления из ОС с помощью команды возврата из прерывания (IRET), признаком неудачного завершения выполнения системного вызова является единичное состояние флага переноса. Результаты выполнения запроса возвращается в прикладную программу через регистры процессора. Данный вариант системных вызовов появился в системах с защитой памяти, в котором программы выполняются в пользовательском режиме и им запрещено

обращаться в не принадлежащей им области памяти. При передачи запроса из программы в ОС требуется передать последней управление и следовательно процессор должен начать выбирать команды из «чужой» области памяти, т.е. памяти ОС, но аппаратура защиты памяти не разрешает этого, возникает конфликт для устранения которого 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 передает управление и меняет состояние процессора на системное

2) **API — интерфейс программирования приложений**
   Данный вариант передачи запросов ОС предполагает осуществление системных вызовов в виде функций. Описанием запроса выступает имя функции и передаваемые в нее аргументы. Результаты выполнения запроса возвращаются в прикладную программу в виде результата функции и в случае необходимости через переданную в функцию аргументы.
   Основные особенности:
   a) Данные составляющие описание запроса размещаются в стеке. Состав этих данных определяется составом аргументов АПИ функции
   b) Передача управления в ОС осуществляется с помощью команды **call**, которая присутствует в системах команд подавляющего большинства процессоров.
   c) Возврат управления из ОС производится с помощью команды **RET**, результаты выполнения запроса возвращаются в прикладную программу через регистры процессора EAX или EAX:EDX.

Программы написанные на разных языках программирования даже под управлением одной ОС используют различные реализации механизмов работы с подпрограммами и в следствии этого без специальных мер нельзя из процедуры написанной на одном языке вызвать процедуру написанную на другом. ОС тоже имеет свой интерфейс вызова несовместимый со стандартными механизмами вызова подпрограмм используемыми по умолчанию в различных языках программирования. Различие заключается в способе вызова, порядке размещения аргументов в стеке и способе удаления их из стека. Для решения этого вопроса в состав языков программирования включаются средства согласования и интерфейсов вызова разноязыковых программ.

Модульное программирование

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 имен и атрибутов
сегмент кода — .CODE, _TEXT
.DATA, _DATA
.STACK, _STACK
Таким образом транслятор сам дает 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 — далекий или межсегментный

Способы возврата из подпрограмм
RET NEAR
RET FAR

Отличия близких от далеких 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) Далекие 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.

ФОТО

Таким образом, при согласовании разно-языковых модулей необходимо иметь возможность определять виды вызовов и возвратов из подпрограмм. В ассемблерных программах для этого существуют следующие средства: явное задание вида вызова и возврата в командах CALL и RET, указание вида вызова в директиве PROC, определение вида вызова от модели памяти директивы MODEL.
В команде CALL указывается один операнд по смысле определяющий адрес подпрограммы, к которой происходит переход. По этому операнду и определяется вид вызова. Если адрес перехода описан в том же сегменте, что и сама команда транслятор генерирует код близкого вызова, если же адрес перехода и адрес вызова расположены в разных сегментах, то генерируется код дальнего вызова.

DOS
1) A label near:
   …
   B label far:
   …
   CALL A — ближний
   CALL B — дальний
2) A proc near
   …
   ret

   B proc far
   …
   ret

   CALL A
   CALL B
3) BX — указатель на функцию
   CALL word ptr[BX] — ближний
   CALL dword ptr[BX] — дальний

Так же имеется возможность задавать вид вызова путем указания модели памяти с помощью директивы MODEL, одним из аргументов которой является укзание типа модели памяти.

Виды моделей:
1) tiny
   Содержит 1 сегмент кода и 1 сегмент данных, физически наложенные друг на друга (16 разрядный DOS)

2) small
   Содержит 1 сегмент кода и несколько сегментов данных (16 разрядный DOS)
3) medium
   Имеют несколько сегментов кода и несколько сегментов данных (16 разрядный DOS)
4) large
   Имеют несколько сегментов кода и несколько сегментов данных (16 разрядный DOS)
5) flat
   Содержит 1 сегмент кода и 1 сегмент данных, физически наложенные друг на друга (32 разрядный Windows)

Различные модели памяти предполагают разное количество сегментов кода, что приводит у неявному указанию вида вызовов подпрограмм:
1) tiny, small, flat
   Характерен 1 сегмент кода и вызовы будут близкими
2) medium, large
   Имеющих множество сегментов кода вызовы будут дальними.

В языках высокого уровня вид вызова и модель памяти обычно определяются опциями компилятора, либо в опция среды программирования, либо в тексте программы. Помимо видов вызовов и возвратов из подпрограмм в организации межмодульных связей обязательным вопросом является согласование способов передачи аргументов подпрограммы. 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

13 декабря
В языках высокого уровня стиль вызова задается в описании процедур и функций. В ассемблерных программах стиль вызова задается директивой proc

Пример 1.
A proc pascal, near
…
A endp
Пример 2.
B proc c, near
…
B endp

Так же стиль вызова можно задавать директивой **.model small, c** или **.model small, pascal**

Вопрос согласования типов аргументов в ассемблерных программах решается путем указания их размерности при перечислении в директиве **proc**

A proc c, far, b:byte, e:word, d:word

Для описания аргументов процедур можно использовать директиву **arg**, которая следует за заголовком процедуры
arg g:dword, f:word, =h

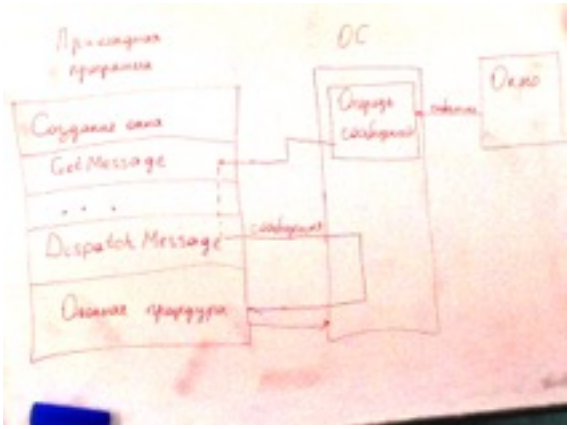
Директива **arg** позволяет не только описывать аргументы, но и присваивать значения их длины некоторому имени (h получит значение 6), при передачи параметров по ссылке в стеке размещаются их адреса, размерность которых зависит от модели памяти, если модель памяти предполагает один сегмент данных, то в стек помещается только смещение до аргумента, если имеется множество сегментов данных — в стеке, для каждого аргумента передаваемого по ссылке, будет размещен адрес состоящий из двух частей смещение и сегментная части. Поскольку модульное программирование предполагает раздельную компиляцию модулей, неизбежно появляются определенные имена приводящие к ошибкам компиляции, для исключения этих ошибок используют директивы **extrn, public.** В директиве **extrn** перечисляются все имена описанные в других модулях, а в директиве **public** — имена, которые должны быть доступны из других модулей. Для хранения внутренних переменных процедур и функций рекомендуется использовать стек. В языках высокого уровня это реализуется описанием локальных переменных в процедурах и функциях. В ассемблерных программах для этих целей используется директива **local**, которая должна следовать за строками с описанием аргументов, например, local i:word, j:dword, k:byte.
В случаях, когда необходимо сохранить регистры неизменными используется директива uses AX, BX, CX, которая сохраняет регистры в стеке при входе в процедуру и восстанавливает их при выходе.

Взаимодействие программ с ОС Windows

На сегодняшний день наиболее распространенным является консольные и оконные windows приложения. Консольные windows приложения построены также, как и 16 разрядные приложения MS DOS. При их запуске внешне происходит тоже самое, открывается окно, в котором видны текстовые сообщения, и в котором пользователь может набирать строки. Отличительной особенностью консольных Windows приложений является то, что в них используется 32 разрядная архитектура: 32 разрядные регистры и адреса. Следующим отличием является то, что консольные приложения используют сплошную (flat) модель памяти, в которой отсутствует понятие сегментов. Третьим отличием являются запросы к ОС, которые выполняются не с помощью программных прерываний, как в MS DOS, а в виде вызова подпрограмм (API функций). Оконные приложения существенно отличаются от консольных, эти отличия связаны с использованием средств ОС для работы с окнами. В оконных приложениях, окно является самостоятельным объектом ОС и отделено от самой программы.

Схема оконного приложения

Воздействие пользователя на окно является событиями, которые обрабатываются ОС и для каждого события формируется сообщение описывающее это событие. Все сообщения выстраиваются в очередь сообщений, находящуюся в ведении ОС. Задачей прикладной программы является получение сообщений из системной очереди с помощью функции GetMessage и передача их через ОС оконной процедуре прикладной программы (DispatchMessage). Оконная процедура выполняет все действия необходимые для обработки действий пользователя (событий). Для того, чтобы приложение получило свое окно, необходимо последовательно вызвать CreateWindow и ShowWindow.



Макросредства ассемблера

Макросредства — это совокупность конструкций языка заставляющих транслятор генерировать или модифицировать исходный текст программы. В C/C++ существуют понятия препроцессора. Например, #define max(a, b) (((a) > (b)) ? (a) : (b))

С учетом возможности использования макросредств ассемблерный транслятор можно использовать как совокупность двух несвязанных модулей: макропроцессора и транслятора. Макропроцессор это модуль, который выискивает в программе 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, которые вызывают подстановку в текст программы последовательностей команд, образующих коды пролога (начало процедуры) и эпилога (конец процедуры). Директива uses вызывает последовательность команд push и pop. Все макросредства ассемблера можно поделить на 4 группы:

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