Лекции по ЭВМ
Пронкин Ю. Н.

2010

Конспект лекций Ю. Н. Пронкина для студентов 1-2 курсов механико-математического факультета МГУ, отделения механики. Составили студенты механико-математического факультета - Сапунов К.В. и Егорычев О.О

Предисловие

Перед Вами лежит конспект лекций по ЭВМ и программированию студентов механико – математического факультета МГУ им.Ломоносова Сапунова К.В. и Егорычева О.О. Курс лекций был прочитан Пронкиным Юрием Николаевичем со 2 семестра 2009 года по 3 семестр 2010 года. Авторы надеются, что данный конспект подготовит студентов к экзамену, а также познакомит с этим увлекательнейшим и интереснейшим предметом.

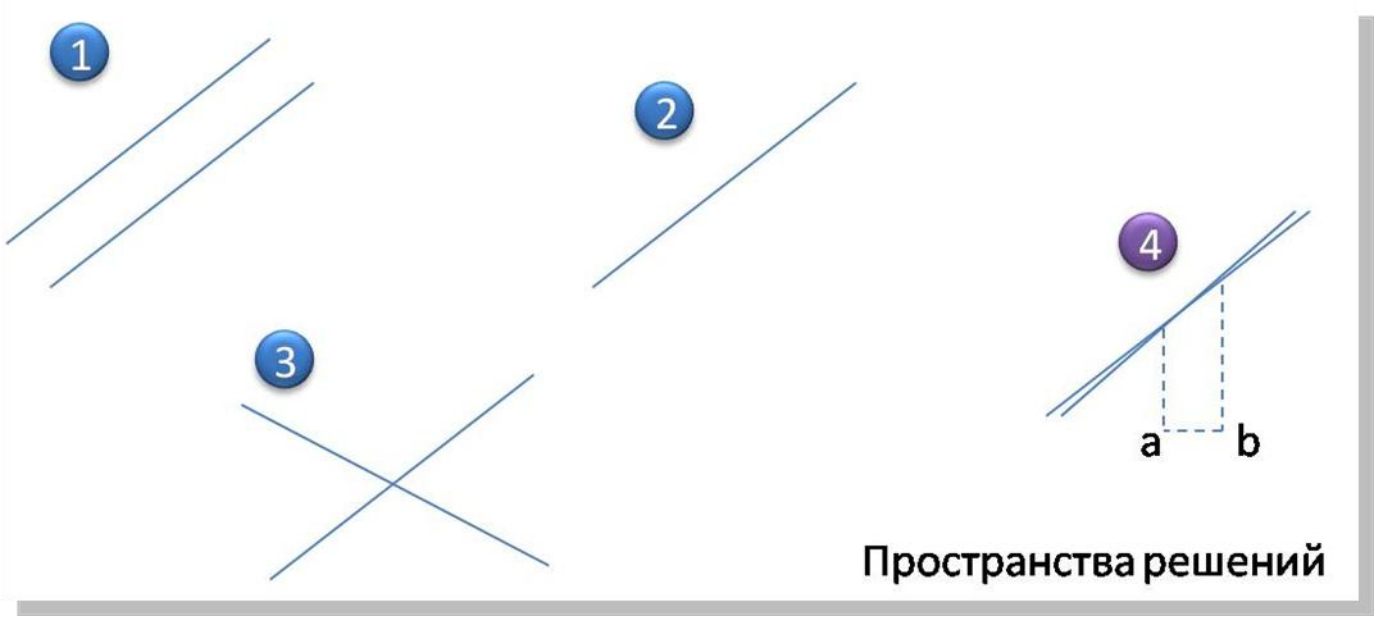
Информационные технологии в наше время являются неотъемлемой частью нашей повседневной жизни.

Авторы благодарят Пронкина Юрия Николаевича за разрешение к печати, предоставление материала, помощь в редактировании текста. Кулагина В.А. за помощь при оформлении сего материала, а также за моральную поддержку в таком нелегком деле, как печать конспекта.

I курс

Лекция №1.
Общая модель решения задач.

1) Постановка задачи - что нужно сделать и что должно получиться.
2) Математическая формулировка задачи
3) Конструирование - выбор метода решения задачи
   К примеру, проверка устойчивости:

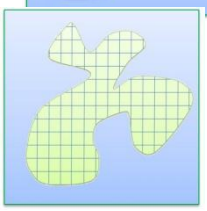
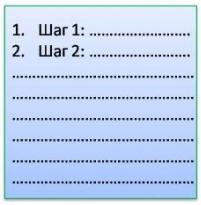


4) Разработка алгоритма решения задачи
   Вопрос стоимости, вопрос времени.

Лекция №2. 20.02.2010

Определение. Устойчивая задача – это задача, решения которой не сильно изменяется при малых изменениях условий. Соответственно, если задача не является устойчивой, то она неустойчивая.

1) Постановка задачи  => 2) Математическая формула $\frac{\partial a}{\partial t} = \Delta U + f$ =>

3) Метод решения  => 4) Алгоритм решения  =>

5) Кодирование программы => 6) Отладка программы => 7) Тестирование программы => 8) Испытание программы

Средства программирования:

- Программирование в машинных кодах.
- Программирование на языке ассемблера
- Программирование на универсальных языках

Испытание программы:

- Программа опознала неправильные наборы данных.
- Программа "упала"
- Программа работает.

Лекция №3. 22.02.2010.
Модель аппаратно-программного обеспечения.

<table>
<tr><td rowspan="4">Новая виртуальная машина</td><td>Прикладные программы</td><td>Java</td></tr>
<tr><td rowspan="2">Системное программное обеспечение</td><td>Инструментальные программные среды
Операционная система</td></tr>
<tr><td rowspan="2">Микропрограммная логика</td></tr>
<tr><td rowspan="2">Системное программное обеспечение</td></tr>
<tr><td>Схемная логика</td></tr>
</table>

1) Аппаратная часть:
Схемная логика - некоторые машинные программы, которые реализованы схемным путем (например, рычаг - положения 0 и 1). Соответствует аппаратной части.
Микропрограммная логика - то, что зашито в самом процессоре. Реализует большинство машинных команд.
~1950 машины имели простые команды (сложение, умножение) =>
~1985 сложные команды (система малых ЭВМ усложнена до предела - VAX // DEC) =>
~2000 переход обратно к простым командам (процессоры семейства RISC). Однако по сравнению с 1950, добавлен развитый кэш.

2) Системное программное обеспечение - слой, который делает системное программное обеспечение чем-то тем, чем мы его хотели бы его видеть.
Операционная система - набор программ, тесно взаимодействующих с железом (3 задачи: виртуализация ресурсов, управление ресурсами, управление средствами взаимодействия с пользователем)
Инструментальные программные среды - компиляторы, редакторы, загрузчики и т.д. Чисто логический слой.
1) + 2) => Виртуальная вычислительная система, виртуальная машина - то, что достается мне, как пользователю. Расширение аппаратного слоя.
Кроссовый компилятор - компилятор, работающий на одной системе, но создающий программы для другой.
Системы виртуальных машин - как бы операционная система, не предназначенная для работы на пользователя. Размножает аппаратные платформы.
Виртуальную машину можно "достраивать"

3) Прикладные программы - те, которые мы разрабатываем сами.
Пример доработки виртуальной машины: мы создали прикладную программу. Другой пользователь взял ее и работает только с ней и больше ни с чем. => для него виртуальная машина включает в себя вашу программу. Получается "новая " виртуальная машина (виртуальная машина + прикладные программы).
Java - язык программирования, напоминающий C, являющийся многоплатформенным компилятором. Компилятор создает не машинный код, а некоторое промежуточное представление (a+b => ab+ - предтрансляционный код). (Java - прикладная программа.)

Лекция №4. 06.03.2010.

Архитектура микропроцессорных вычислительных систем.

I) Логическая модель вычислительной системы

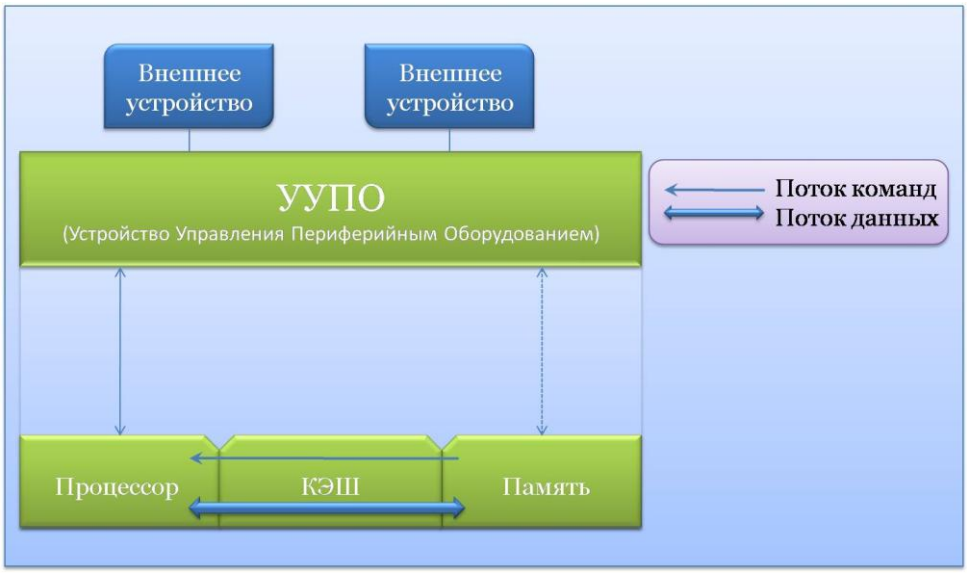


рис. 4.1

Процессор
- вычисление
- управлений процессом вычисления
- управление периферийным оборудованием (ПО - устройства, находящиеся внутри процессора, позволяющие ему выполнять вторую задачу)

Замечание:
В нашем курсе процессор и микропроцессор – синонимы, и хотя исторически это не так, ныне различия сильно сглажены.

Память (оперативная память)
- хранение команд программы
- хранение данных

Оперативная - значит, что память хранит команды и данные лишь на время выполнения соответствующей программы.

КЭШ-память
- временное хранение небольших порций команд программы
- временное хранение небольших порций данных

Проблема: процессор может работать намного быстрее, чем память (алгоритмы обращения к памяти) =>
Решение: введение КЭШ-памяти.

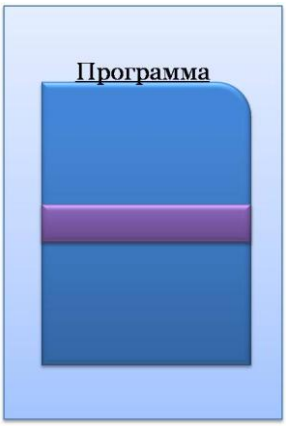


Рис. 4.2 (программа с выделенным объемом команд и данных, использующих КЭШ)

Процессор, на самом деле, не знает КЭШ-памяти. Обращение идет к оперативной памяти, но с помощью соответствующих алгоритмов сначала проверяется на наличие необходимого в КЭШ. Однако, если в КЭШ нет нам необходимого, это займет определенное, немалое по сравнению с выполнением операций, время.
--> - поток команд; <=> - двунаправленный поток данных
Поток данных двунаправлен так как, получив данные и выполнив операцию, мы положим обратно в память результат.
Однонаправленные команды - программы, просто выполняющиеся (например, создающие себе подобные программы)

Устройство управления периферийным оборудованием

Если есть программа, команда способная напрямую использовать устройство, то такое устройство называется внутренним. Если нет, то внешним.
- согласование форматов представления данных (форматов, принятых процессором и устройством) - перевод данных из одного формата в другой, при переброске данных из одной программы в другую.
- согласование скоростей работы процессора и ВУ (буфер)
- контроль над работой ВУ - готовность и работоспособность данного устройства, во многих устройствах, с точки зрения их строения, есть элементы, доступные и недоступные для программы. Нас будут интересовать только доступные элементы.

УУПО - обычно называется контроллер или адаптер.

УУПО не всегда в состоянии самостоятельно выполнить задачи, перед ним стоящие => существуют специальные программы, использующиеся для обслуживания ВУ (драйверы).

Прямой доступ к ОП со стороны ВУ (пунктирная линия на рис. 4.1)

- контроллеры прямого доступа к памяти (Direct Memory Access - DMA) - не возлагаются контрольные и вычислительные функции, только пересылка данных
- процессоры ввода/вывода - пересылка данных, контроль, вычисление
- каналы ввода/вывода (понятие из систем коллективного доступа)

Лекция №5. 13.03.2010

Типы и характеристики микропроцессоров.

Характеристики микропроцессоров:

- Разрядность
- Частота синхронизации
- Организация системы команд

Разрядность микропроцессора (разрядность внутренних регистров процессора)

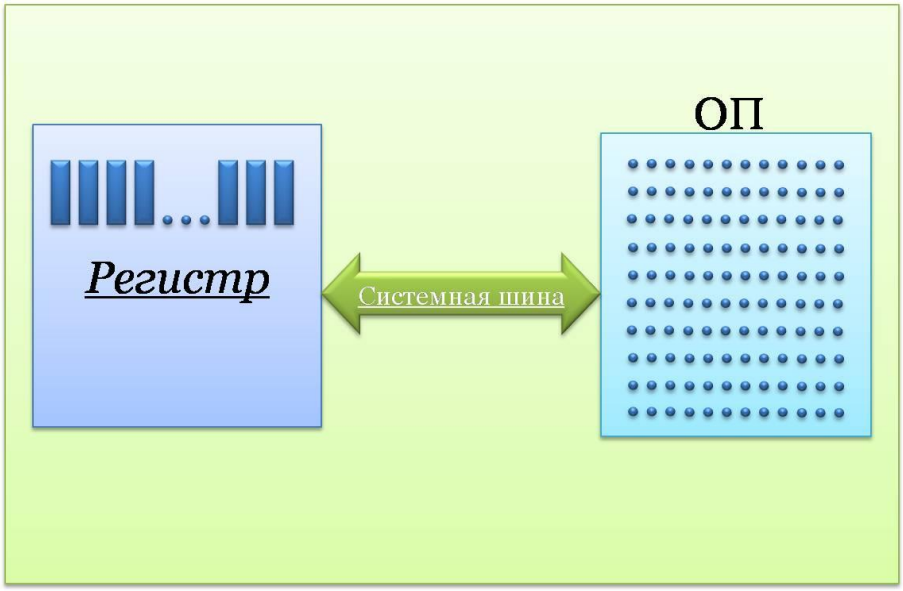


рис.5.1 (связь процессора и ОП) Внутри каждого процессора существует некоторое кол-во регистров

Системная шина - не более чем набор проводов. Разрядность=16 => в шине 16 проводов, и т.д.
Разряд = информационный бит (исторически - 8 разрядов (+рис.5.2) -> 16 р. -> 32 р. -> || 48 р. || -> 64 р. -> || 80 р. ||)
80 разрядов обеспечивает вычисление с 24 знаками после запятой.

Рис. 5.2.

Частота синхронизации

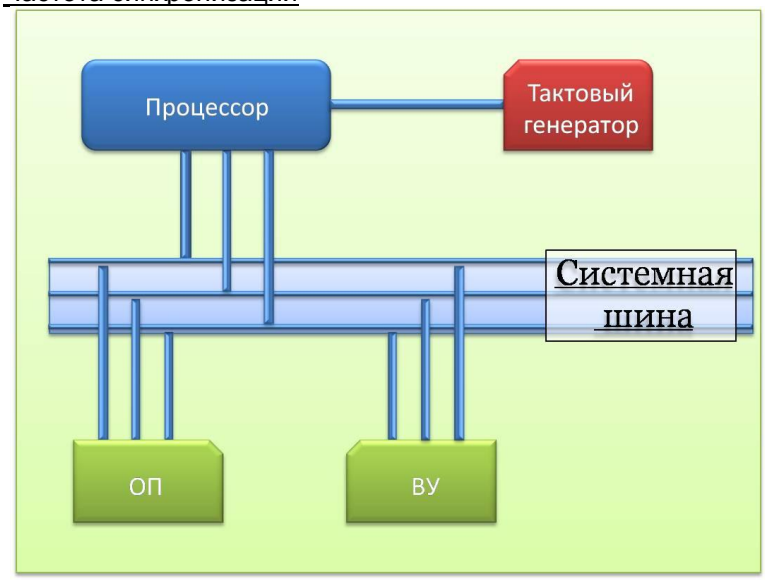


Рис. 5.3.

Тактовый генератор - (с нашей точки зрения) это часы. Все события в процессоре могут происходить только на удары этих "часов", вне этого сигнала нет ничего и быть не может.

Повышение производительности:

- Повышение частоты синхронизации (4МГц ----------> ~4ГГц)
- Укрупнение "элементарных" операций (рис.5.4). Существуют процессоры, которые за 1 удар часов выполняют 4 операции.

Организация системы команд

- Сколько операндов имеет одна команда? (Интуитивно - 3.)

(Операнды - некоторые объекты, над которыми выполняются какие-то действия (c = a+b), хранятся в ОП.)

- Какие действия выполняет одна команда? (Сложение - одно действие, а Вычисление тригонометрической функции - много.)
- Что является операндом команды?

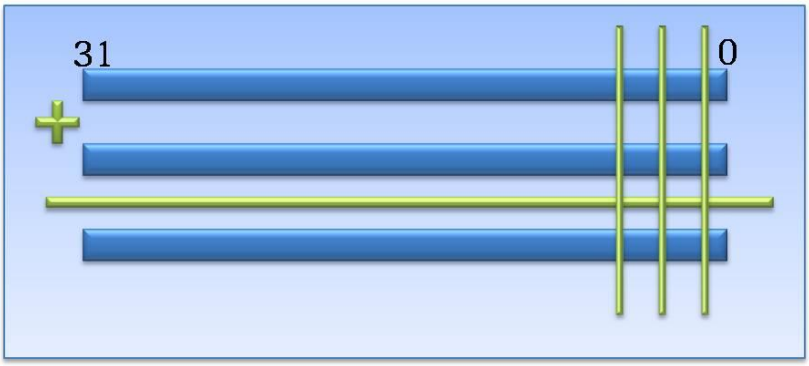


рис.5.4.

Например, операндом команды может только аргумент, загруженный в регистр (кто выполняет действие загрузки)

- Внутренний параллелизм – конвейеризация

Количество операндов в команде

Трехадресная команда

рис.5.5. Код операции. Destination source

Двухадресная команда

рис.5.6. На рисунке: Op1\Op2 - операнды 1 и 2.

Лекция №6. 20.03.2010

Формат команды (количество операндов в команде)

Трехадресные (см. рис. 5.5)

Двухадресные (см. рис. 5.6)

рис. 6.1

Одноадресные (рис. 6.1) => возникает операция пересылки (Mem <- Ac), чтобы иметь возможность сохранять промежуточные результаты. Если данная операция занимает не слишком много времени, то в принципе все хорошо.

Безадресная (советские машины "Эльбрус")

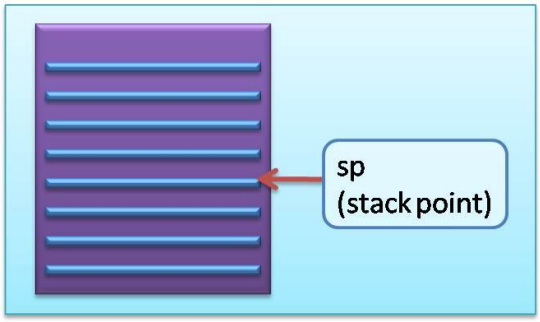


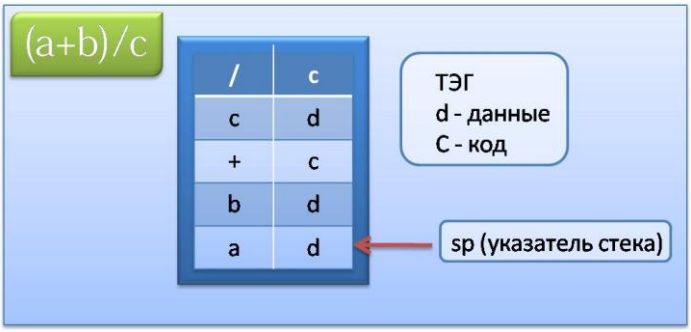
рис. 6.2

Стэк - особый вид организации данных (рис. 6.2) Sp - стэк-поинтер - указатель вершины стэка (стоит на последнем элементе).

Преимущества стэка: имеются максимум два операнда, стэк не нуждается в указании на него, не возникает фрагментации памяти.

Безадресные машины могут быть реализованы на идее стэка (рис. 6.3)- в стэк в определенном порядке будут складываться все операнды и функции.

Тэги - (d - данные, и c - код, на рисунке 6.3) - дополнительная информация о элементах стэка.



Трансляция при помощи стэка (рис. 6.4)

рис. 6.3

рис. 6.4

"Сложность" системы команд

Операции над целыми и вещественными числами принципиально разные => не существует процессора, работающего и с теми, и с другими по одному алгоритму.

Определение сложности команды:

1) Что может делать команда?

2) Что является параметрами?

Пример сложной команды - копирование, пересылка данных (рис. 6.5)

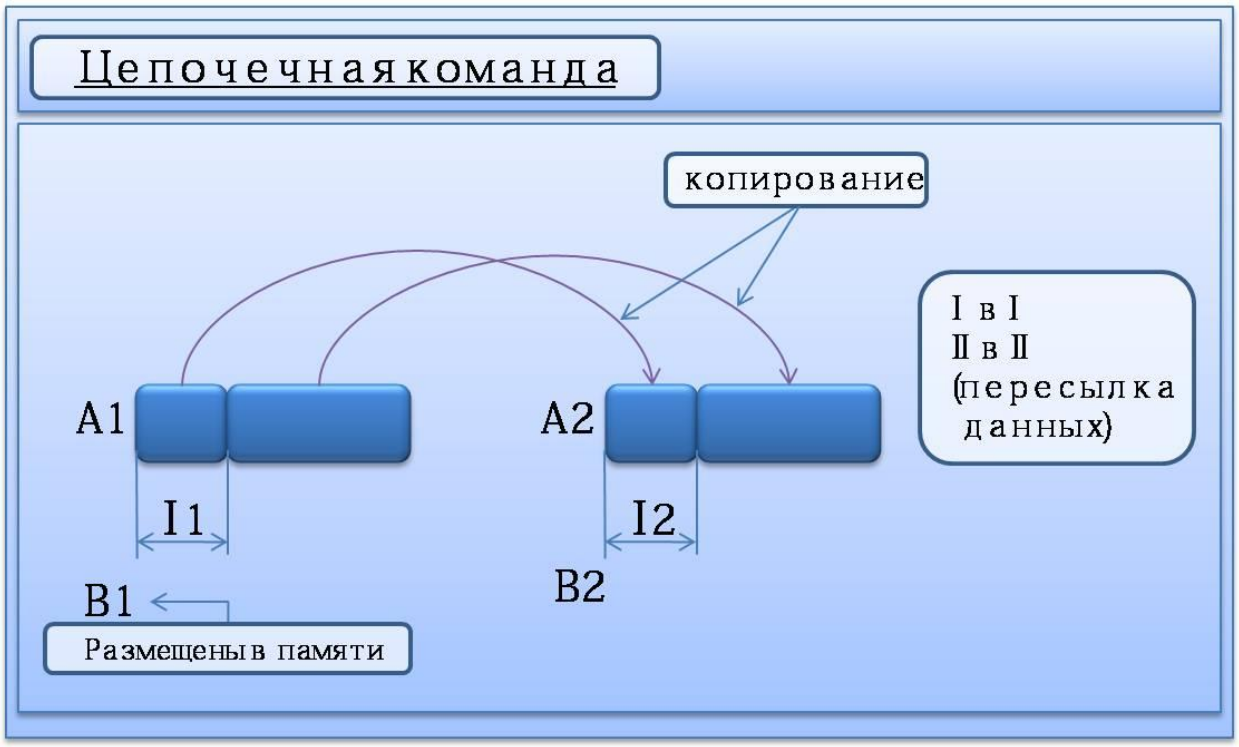


рис. 6.5

Ответ ко второму вопросу: операнды могут находится только в регистрах процессора -> (усложнение) операнды могут быть заданы явно своими адресами в оперативной памяти (Add a,b a<-a+b) => операнды заданы неявными адресами в ОП

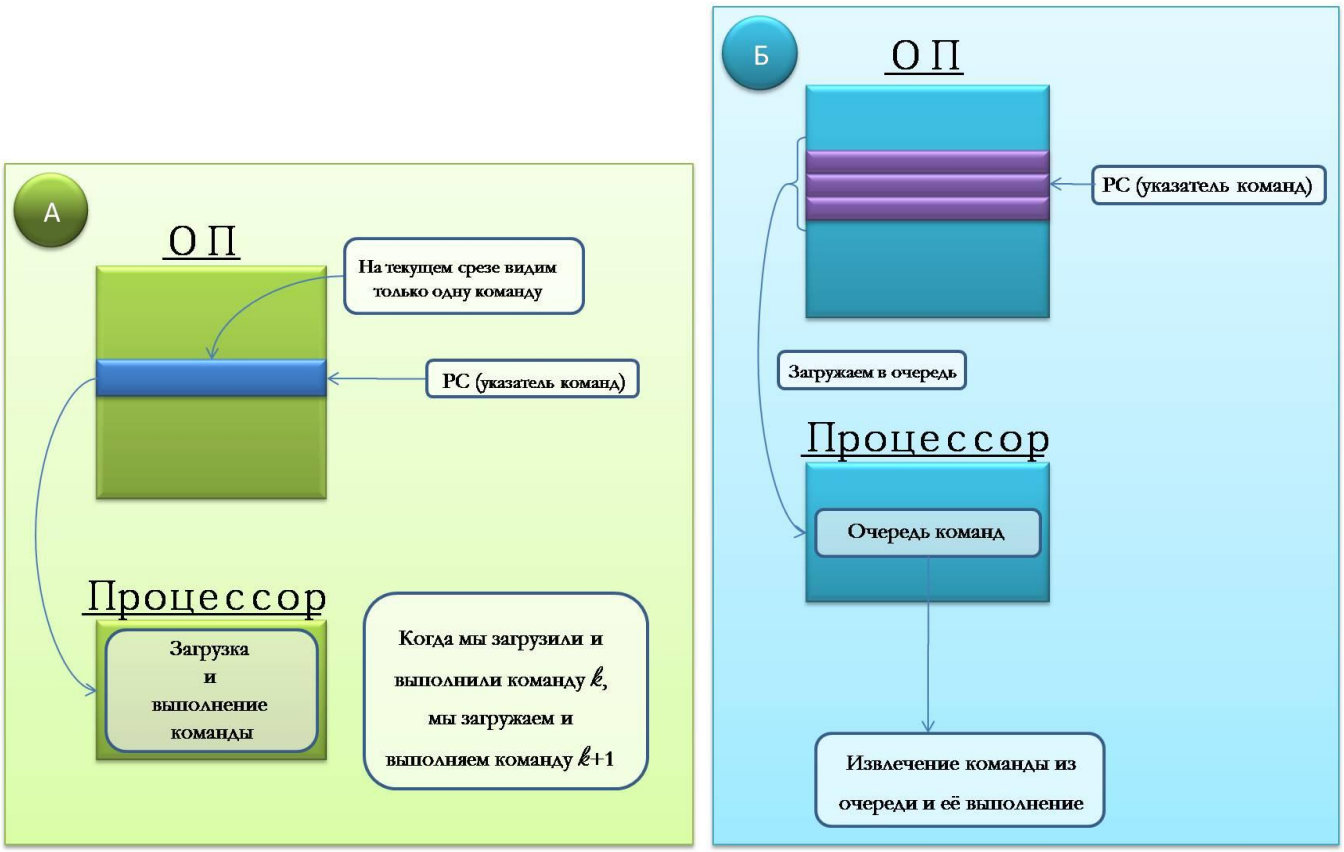
рис. 6.6

CISC - Complex Instruction Set Computer - процессор со сложными командами
RISC - Reduced Instruction Set Computer - процессор с простыми командами
Компьютеры общего пользования - комбинированные. "Наружу" торчит CISC, но процессор в основном работает по RISC.

Почему RISC-машина работает быстрее ?

- использование фиксированной длины команд
- наличие конвейера команд
- наличие КЭШ-памяти для команд и данных

Конвейер команд (рис. 6.7А и рис. 6.7Б)



(рис. 6.7 - схемы А и Б)

Лекция №7. 27.03.2010

Этапы выполнения команд. Использование КЭШ-памяти.

Этапы выполнения команды – 6ти ступенчатый конвейер

1) Загрузка программы
2) Декодирование команды
3) Декодирование операндов
4) Загрузка операндов
5) Выполнение команды
6) Сохранение результата

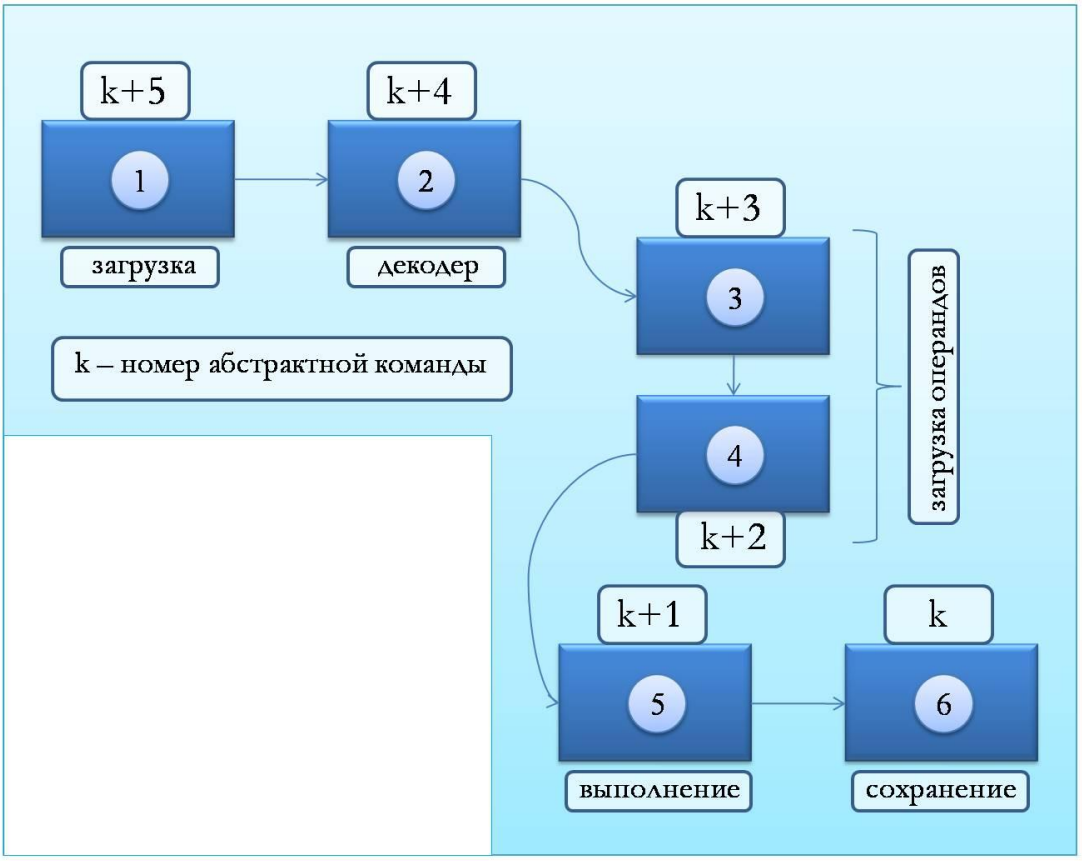


Рис. 7.1 – этапы выполнения команды

Данная схема работает хорошо, пока конвейер не будет давать сбоев и совершать остановки. Так как на разгон конвейера требуется дополнительное время и ресурсы => необходимо разобраться с причинами остановки конвейера.

Основные причины остановки конвейера: (типы конфликтов при работе конвейера команд)

- структурные конфликты
- конфликты по управлению
- конфликты по данным

Структурные конфликты (исп. Рис. 7.1b)

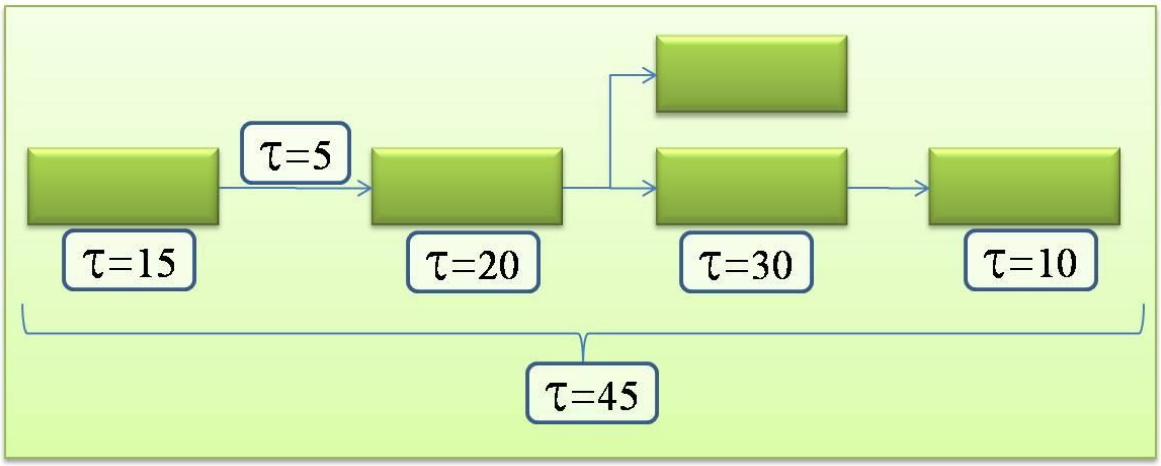


Рис. 7.1b

Каждый этап в конвейерной схеме требует каких-то затрат (время, память и т.д.). T - время, требуемое на выполнение шага.

Мы видим разность в выполнении шагов.
Однако эффект этих задержек не оказывает сильного воздействия на быстродействие схемы. И эти конфликты обычно не решаются. Мы отводим для всех шагов общее время (как правило, усредненное) таким образом, чтобы в большинстве ситуаций время выполнения шага его бы не превышало. Таким образом, получаем отсутствие задержек.

Рис. 7.2.

Бывают ситуации, когда выполнение шага не вписывается в отведенное под него время. Но решать данные проблемы не выгодно из вопросов стоимости.

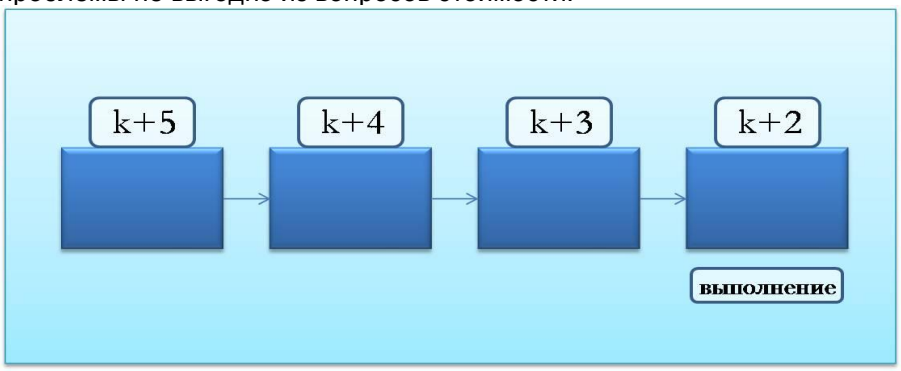


Рис . 7.3.
Конфликты по управлению (рис. 7.2 + рис. 7.3)
Jmp – «jump», команда перехода.
Проблема команды перехода в том, что она сама тоже требует каких-то шагов. Особенно, если это условный переход. И достаточно длинный и разветвленный переход останавливает конвейер.
Конфликты управления вызваны командами передачи управления (ибо мы никогда не знаем, куда мы попадем). Если бы мы заранее знали адрес, проблемы бы не стояло.

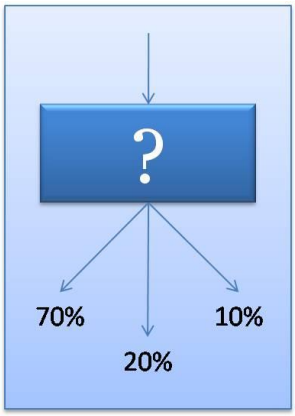
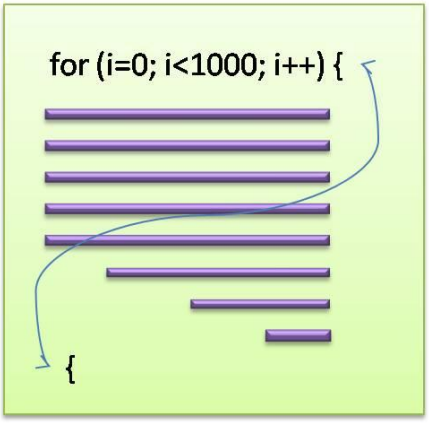


Рис. 7.4.

Следовательно, способы разрешения эти проблем состоят в предсказании адресов переходов (рис. 7.4):
Проводится анализ вероятности прохождения по тому, или иному пути (современный процессор имеет отдельные модули по предсказанию направления перехода и достигают 90 – 95%).
Некоторый ТУПОЙ пример – бесконечный цикл (рис. 7.5)



Конфликты по данным (рис. 7.6)
В данном случае нас интересуют вторая и четвертая ступени.

Суть конфликта:
- команды k и k+2 работают с одним и тем же элементом данных
- команда k+2 выполнила загрузку раньше, чем команда k записала свой результат

Рис. 7.5.

В современных машинах возникает множество конфликтов по данным, так как часто программы выполняются не в строго установленном порядке (попытка улучшения оптимизации, а также внутренний параллелизм процессора)
Рис. 7.7 – отсутствие синхронности в выполнении веток
АЛУ – арифметико-логическое устройство – выполняет все основные процессы вычислений.

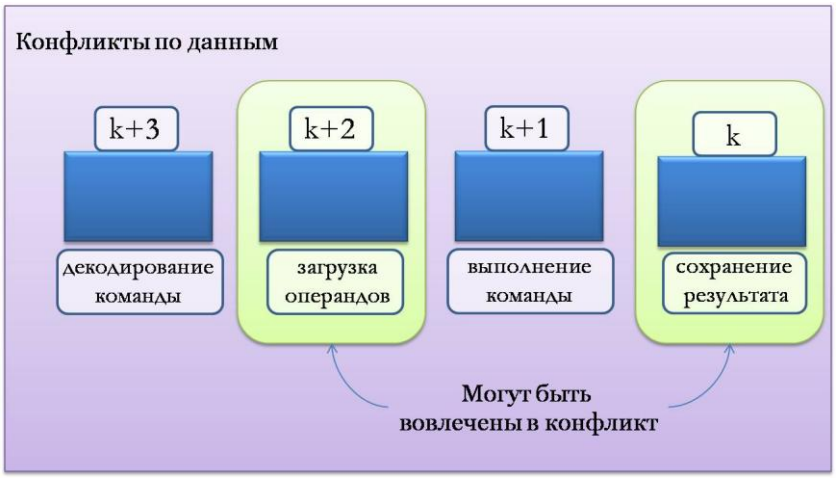


Рис. 7.6.

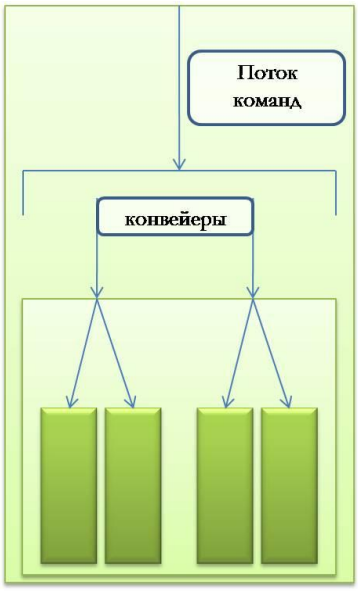


Рис. 7.7.

Конвейерная обработка является неотъемлемой частью RISC-процессоров.

Использование КЭШ-памяти (рис. 7.8)

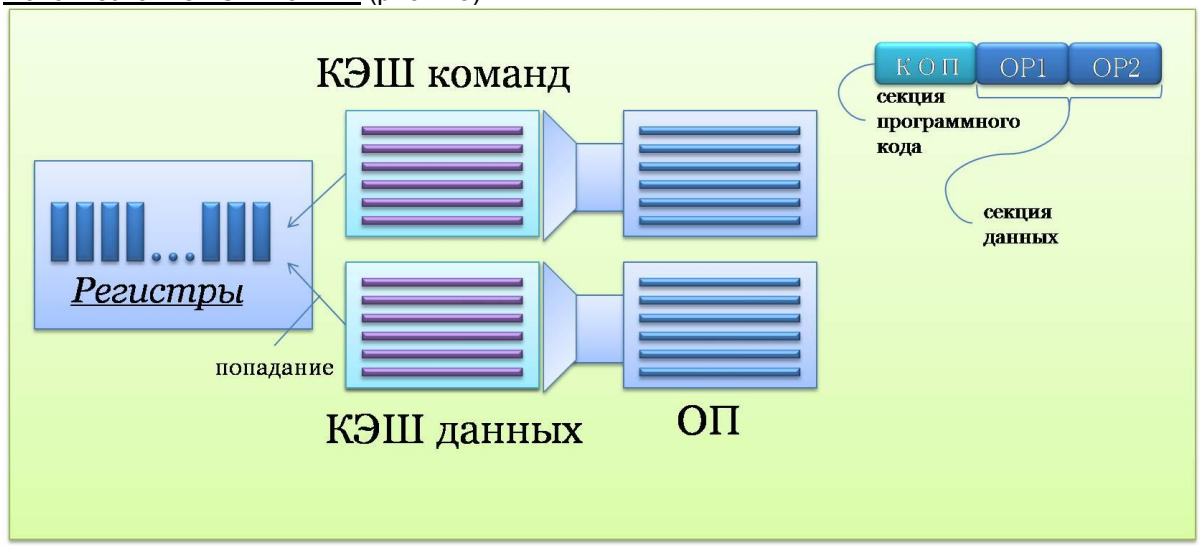


Рис. 7.8.

Рассматриваем уже как RISC, так и CISC процессоры.
КЭШ-память имеет высокую скорость обращения, но малый объем (кстати, в современных процессорах используется КЭШ-память с несколькими уровнями).
Из секций мы отображаем данные на КЭШ-память. Ситуация, когда интересующая процессор информация лежит в КЭШ называется попаданием, обратная ситуация – промах (плохо, так как получаем множественные обращения к КЭШ-памяти и ОП).
Замечание: в современных Pentium отсутствует КЭШ-память команд. Команды раскладываются на простейшие.

Лекция №8. 3.04.2010

Организация основной памяти
Концепция адресного пространства

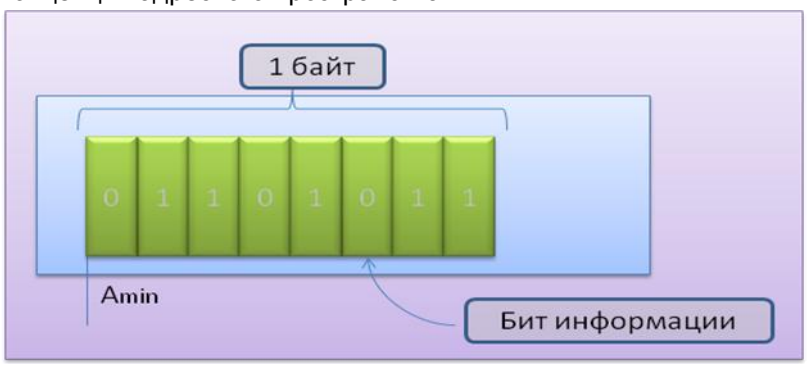


Рис. 8.1.

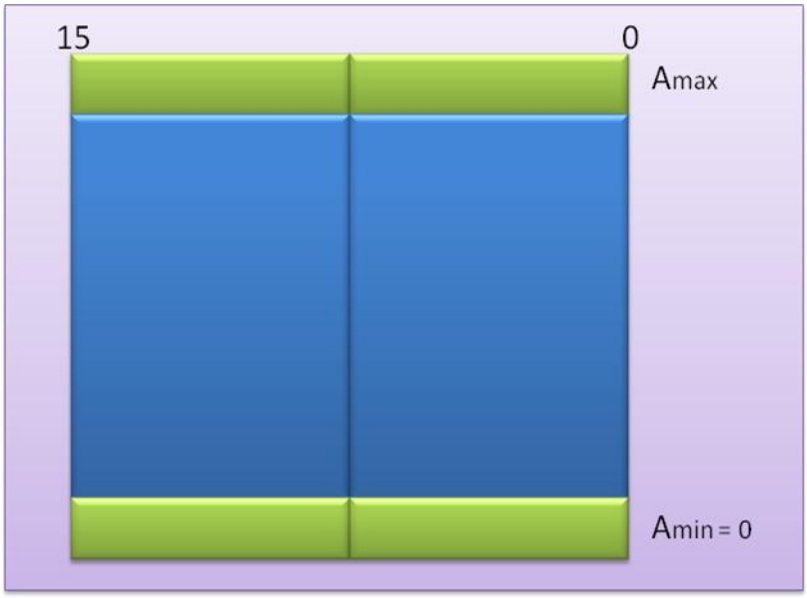


Рис. 8.2.

A’ max – номер самой последней ячейки физической памяти.
A’’ max - наибольший адрес, который может быть обработан.

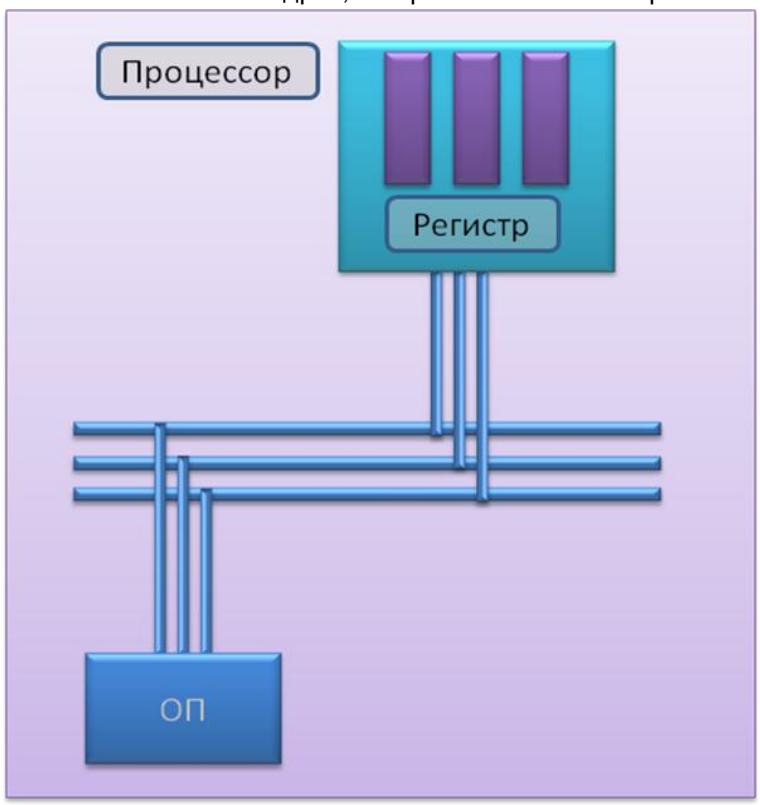


Рис. 8.3.

- A’ max = A’’ max
- A’ max > A’’ max
- A’ max << A’’ max

Виртуальная память

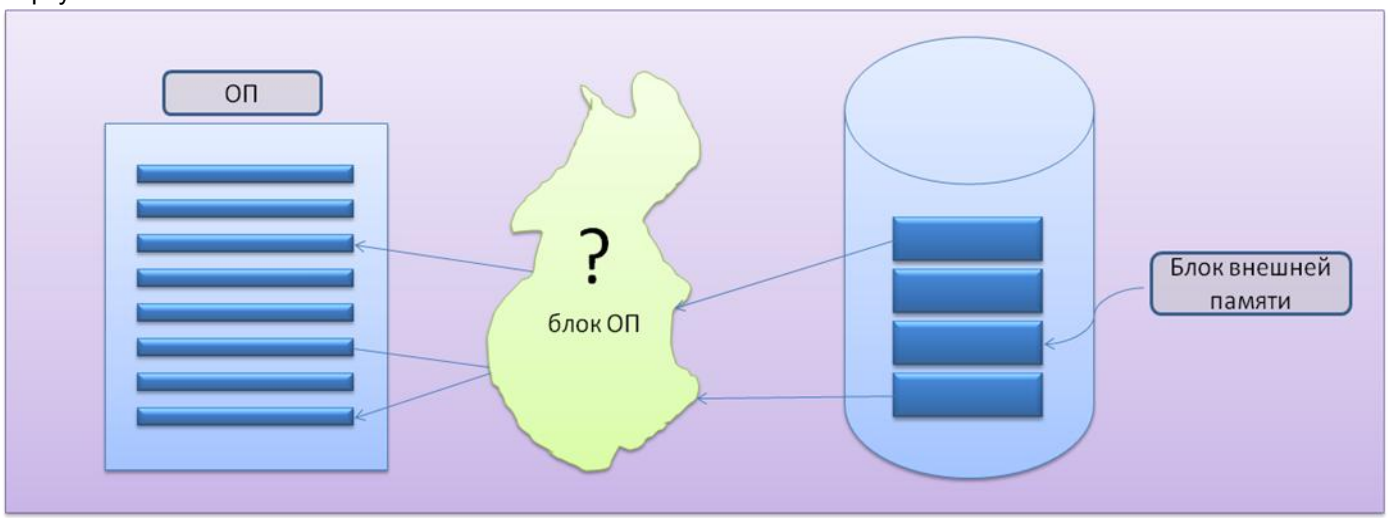


Рис. 8.4

1 Кб = 1024байта
1 Мб = 1024 Кб
1 Гб = 1024 Мб

Рис. 8.3. – В регистрах процессора хранятся машинные слова

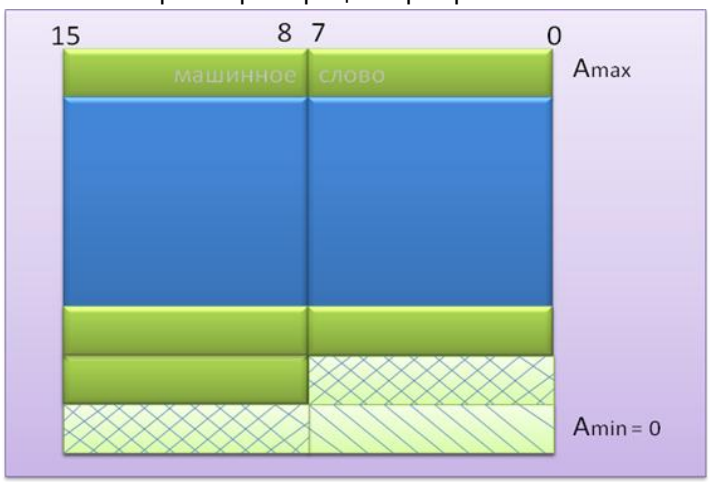


Рис. 8.5.

Что такое адрес слова:

- За адрес слова принимается адрес его младшего байта (Intel…)
- За адрес слова принимается адрес его старшего байта (Motorola; Power PC)

Логическая модель КЭШ – памяти
Многоуровневая организация памяти

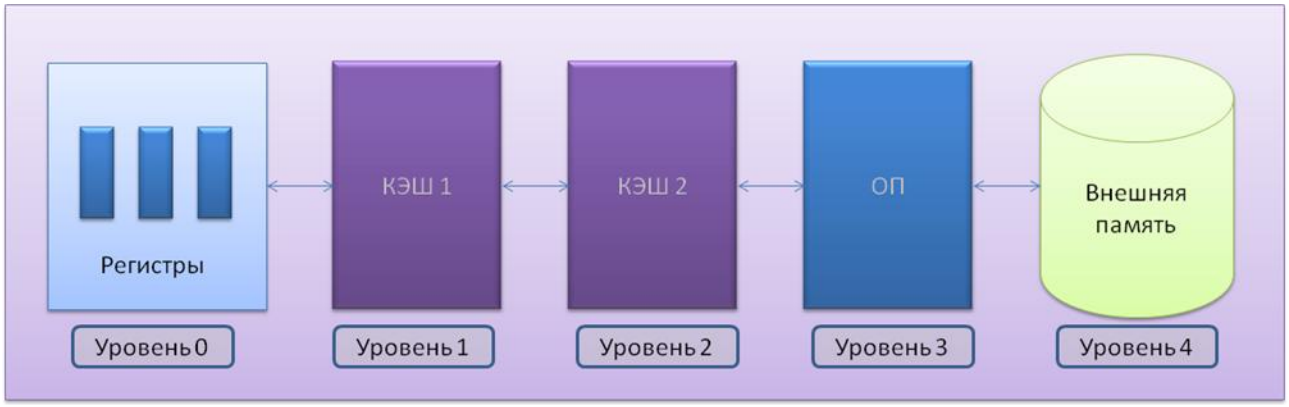


Рис. 8.6.

Лекция №9. 10.04.2010

Мы можем постулировать по крайней мере два основных рассуждения по организации памяти.
1-ое состоит в том, что всякий элемент данных с более высокого уровня может быть найден на всех нижних уровнях. (Некая копия более низкого уровня)
2-ое, если какой-то элемент не найден на более высоких уровнях, то он присутствует на более низких.
Делаем остановку, производим поиск элемента во вторичной памяти.
Кэш память – единица килобайт. Внешняя память измеряется в сотни. Оп гигабайтами. Дисковые накопители(внешние) неограниченны.
Основные различия основываются на адресации.

Работа с адресами

1. В машинных командах используются нефизические адреса(виртуальные и логические).
2. Внутри процессора существует некая логика трансляции адресов (логический адрес превращает в физический). (Рис 9.1)
3. Процессор общается к ОП по сформированному физическому адресу.
4. Контроллер кэш-памяти перехватывает физический адрес.
5. При наличии нужного элемента данных в кэш памяти немедленно передается в процессор.
6. При отсутствии нужного элемента в данных в кэш его нужно искать на более низких уровнях.

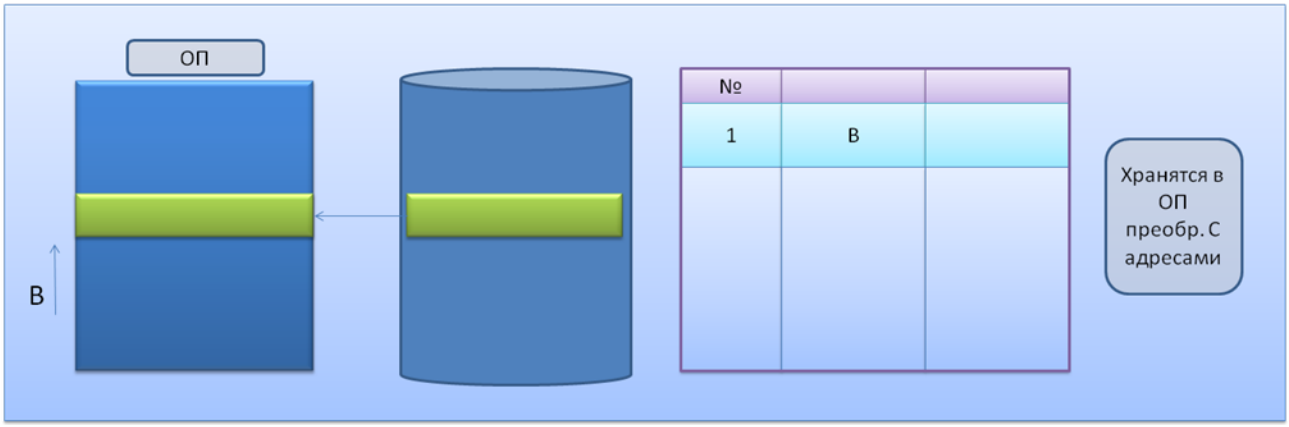


Рис. 9.1.

Попадание(hit): Элемент данных найден в кэш.
Промах(miss): элемента данных в кэш памяти нет.
Коэффициент попадания(hit ratio): общее отношение числа попаданий к общему числу обращений.
Потери на промах(miss penalty): время, затраченное на доступ к элементу данных в случае промаха.

Задачи логического конструирования кэш-памяти:

- Максимизация коэффициента попадания
- Минимизация времени проверки на промах.
- Минимизация потери на промах.
- Выбор стратегии запуска.

Модель работы кэш-памяти определяется 3 стратегиями:
1. Стратегия размещения и доступа:
   Где размещать в кэш элементы данных и как их найти.
2.Стратегия замещения:
   Какой блок замещать в случае промаха.
1. Стратегия записи.

Стратегия размещения и доступа определяет основную модель кэш-памяти.

1. Кэш-память с прямым отображением. Размещая элемент данных, не имеем возможности выбора. Для каждого элемента место фиксировано.

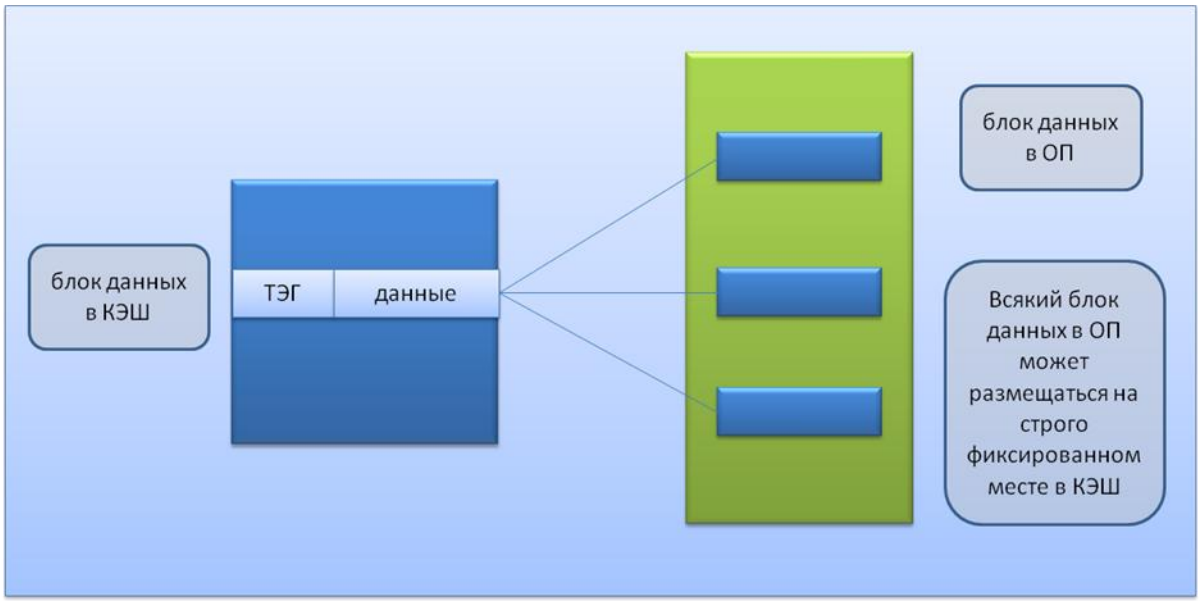


Рис 9.2

Всякий блок памяти их оперативных данных может размещаться на строго фиксированном месте в кэш. Трансляция адреса при доступе к кэш-памяти.

Рис.9.3

Индекс: адрес блока в кэш.
Тэг: ассоциативный признак.

2. Полностью ассоциативная кэш-память. Ограничения на расположение элементов в кэш-памяти сняты.

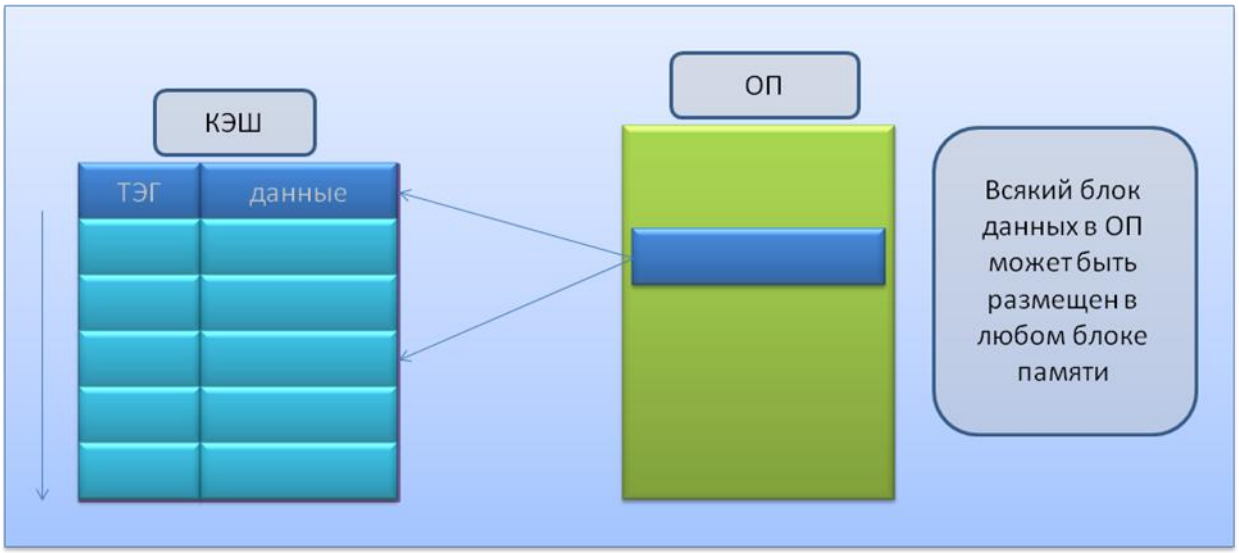


Рис 9.4.

Всякий блок данных из оперативной памяти может быть размещен в любом блоке кэш-памяти. При поиске элемента данных в кэш все тэги сравниваются одновременно. Возникает аппаратная избыточность.

Рис.9.5.

3. Множественно ассоциативная кэш-память.
   При поиске элемента данных в кэш-памяти все тэги блока сравниваются со входным тэгом одновременно.

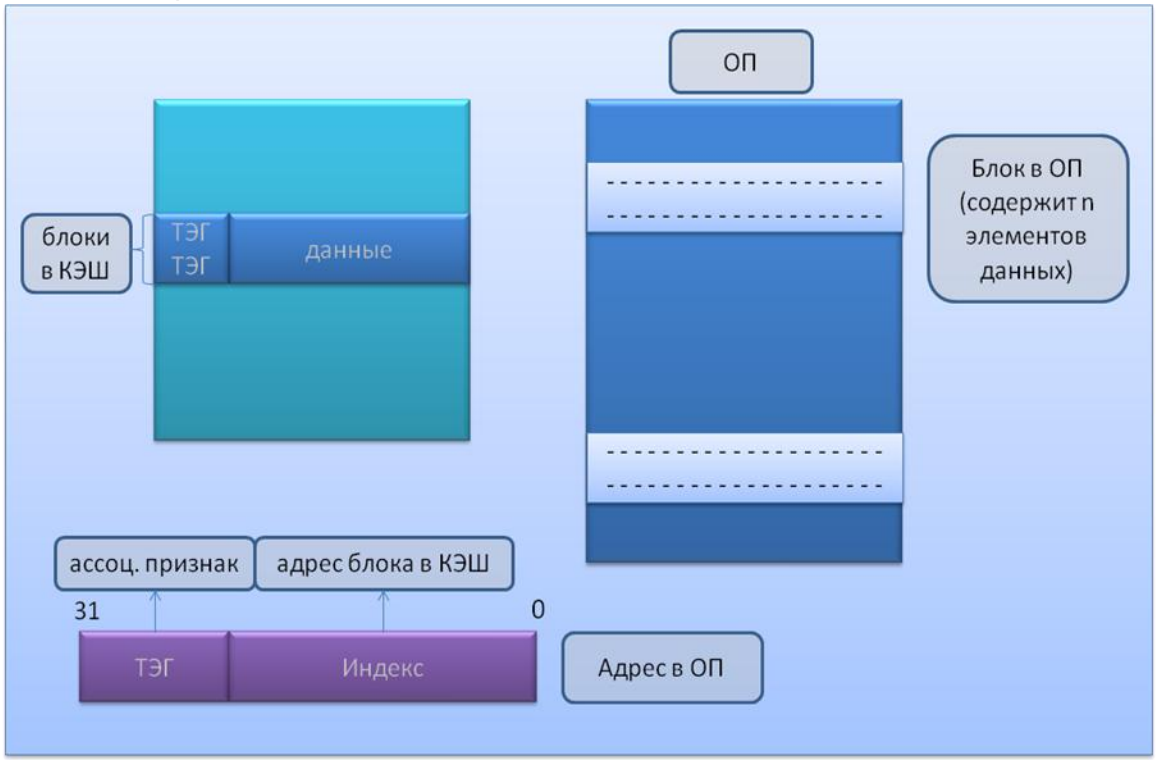


Рис 9.6

Лекция №10. 17.04.2010

Стратегии замещения:

1) Стратегия случайного замещения: замещается случайно выбранный блок.
2) Стратегия FIFO (First In First Out):замещает самый первый загруженный блок.
3) Стратегия LRU (Least-Recently Used):замещает наиболее редко используемый блок.
4) Удалять блоки, к которым не было обращений на отрезке времени t.

Стратегии записи:

Трафик памяти по записи составляет обычно около 30%. Также операции по записи намного сложнее по созданию, чем операции замещения \ уничтожения.

1) Блок, в который мы записываем, находится в памяти КЭШ
    1. Запись со сквозным копированием: одновременно модифицируются верхний и нижний уровни памяти (**«+»**: сохранение когерентности; **«-»**: медленно)
    2. Запись с обратным (или копированием): запись только в КЭШ, копирование в ОП откладывается до момента замещения блока (**«+»**: быстро; **«-»**: нарушение когерентности)
2) Блок в памяти КЭШ отсутствует
    1. Предвыборка при записи: отсутствующий блок предварительно загружается в КЭШ из ОП.
    2. Запись в окружение: данные записываются только в ОП (или в более низкий уровень КЭШ)

Организация системной шины, управление доступом к системным устройствам

Системная шина состоит из трех линий:

1) линия данных
2) линия адреса
3) линия управления

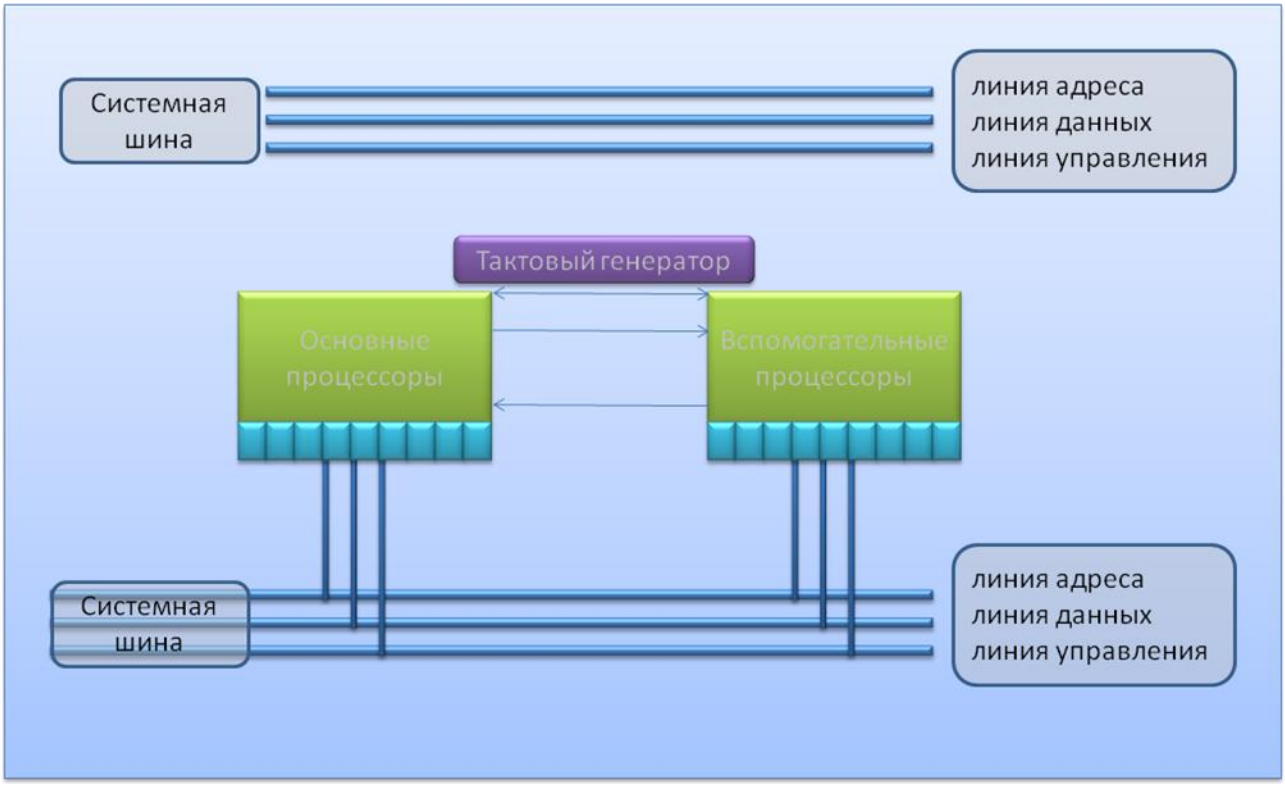


рис. 10.1

Тактовый генератор – «системные часы», любое действие происходит на удар этих часов. Обеспечение синхронизации работы устройств.

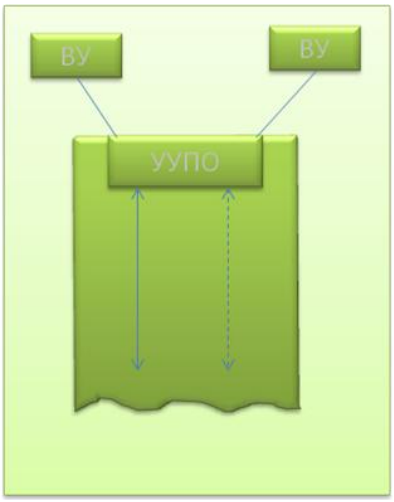


Рис. 10.2 (Фрагмент – напоминание с прошлых лекций)

Основные процессоры

1) Являются ведущими системными шинами
2) Равноправны по отношению к системным шинам

Вспомогательные процессоры

1) Подчинены основному в смысле доступа к системной шине
2) Реализуют один из возможных механизмов доступа к шине:
   - запрос системной шины
   - кража системной шины

Типы многопроцессорных конфигураций.

1. Слабо связанные конфигурации. Все процессоры равноправны -> все процессоры равноправны
2. Сильно связанные конфигурации. Есть основные и вспомогательные процессоры
3. Сопроцессорные конфигурации: сильно связанные + общий программный код

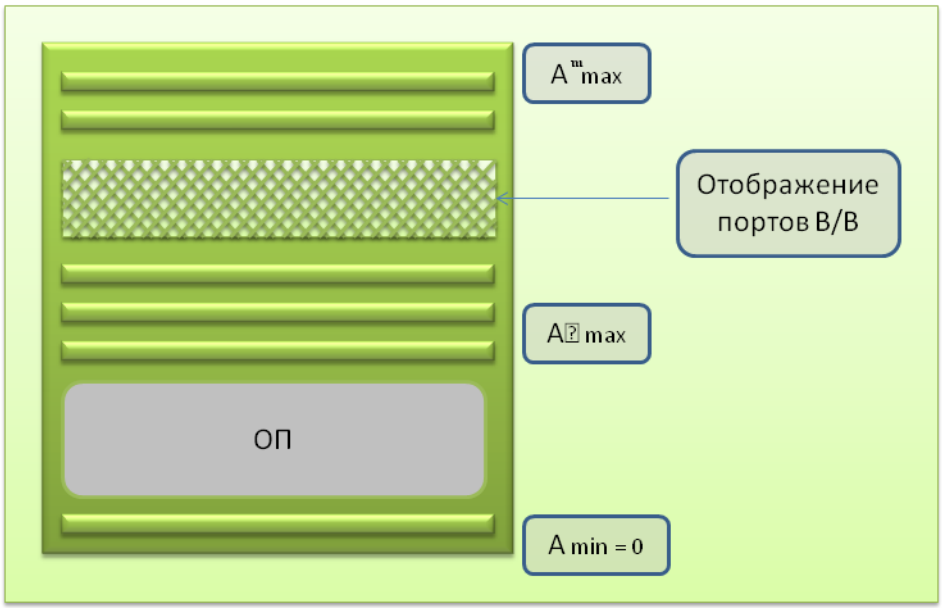


Рис. 10.3

Цитаты:
«По критерию абсолютнейшей простоты…»

Лекция №11. 24.04.2010

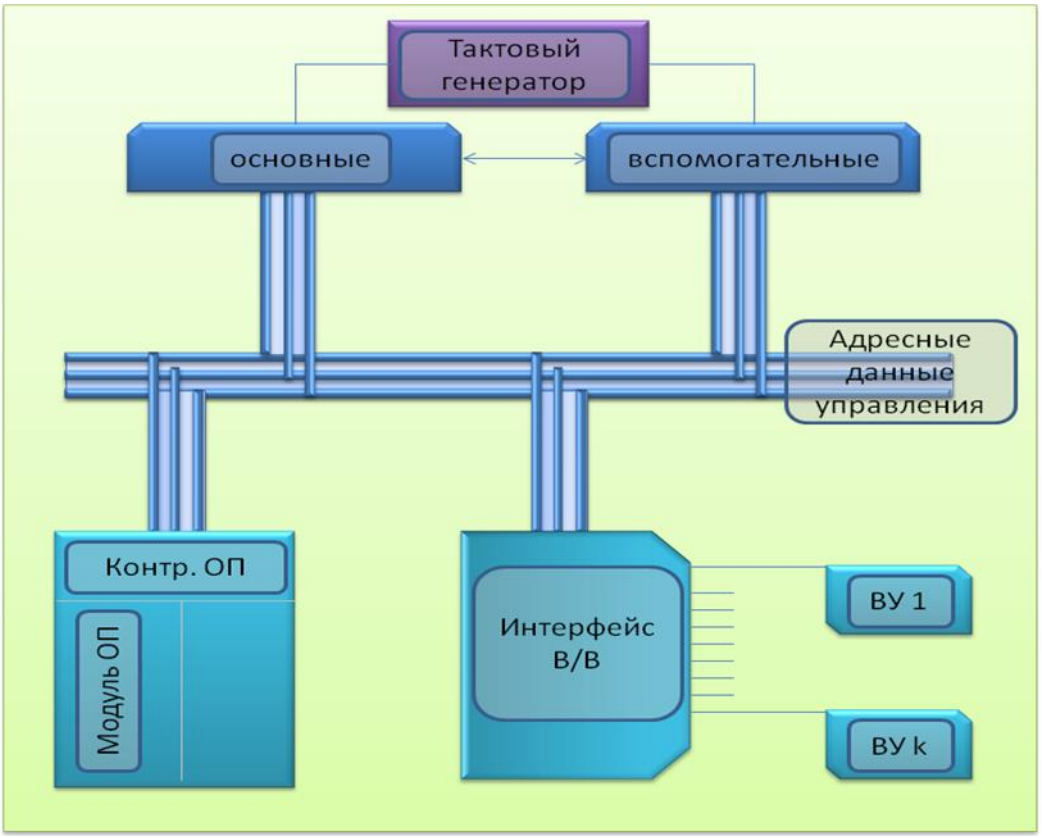


Рис.11.1

Классификация портов:

- Порты состояния
- Порты управления
- Порты данных (буферные порты)

Способы адресации портов В/В

- Включение портов в/в в единое адресное пространство процессора.
- Наличие разделенных пространств оперативной памяти и портов в/в.

  Требует наличия разделенных команд вне работы с ОП и портами в/в:

  ОП mov, Dst,Src

  Порты in, out, ost, Port, Port, Src.

  Dst <-Port; Port <- Src;

Машинные способы представления информации

Данные:

Числа Символы

Целые вещественные: Двоично-кодированные десятичные числа

- Целые без знака
- Целые со знаком

1.1.1. Целые числа без знака

X – основание системы исчисления

$a_{n-1}\ldots a_2\ a_1\ a_0 = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_0x^0$ - расширенная запись числа

В нашем курсе (как это не было бы неожиданно): X = 2;

Пример: $0101_2 = 0*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 5_{10}$;

$0000_2 = 0_{16}$ $0101_2 = 5_{16}$ $1011_2 = A_{16}$

$0001_2 = 1_{16}$ $0110_2 = 6_{16}$ $1100_2 = B_{16}$

$0010_2 = 2_{16}$ $0111_2 = 7_{16}$ $1101_2 = C_{16}$

$0011_2 = 3_{16}$ $1000_2 = 8_{16}$ $1110_2 = D_{16}$

$0100_2 = 4_{16}$ $1001_2 = 9_{16}$ $1111_2 = F_{16}$

Пример: $0111\ 1010\ 1110_2 = 7AE_{16}$

Лекция №13. 15.05.2010

1. С фиксированной точкой d….d.ff…f
2. С плавающей точкой

Короткое вещественное число (float)

Рис. 13.1.

Временное вещественное число (long double)

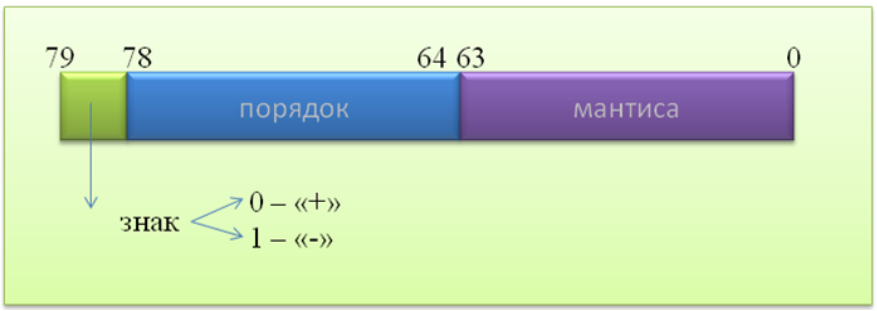


Рис.13.2

Зададим себе вопрос: Что плохо влияет на точность вычислений?

Любые операции над числами сильно различного порядка.

1.2. Двоично - кодированные десятичные числа

$0000_2 = 0_{10}$ $0101_2 = 5_{10}$

$0001_2 = 1_{10}$ $0110_2 = 6_{10}$ 1100_2 - «+»

$0010_2 = 2_{10}$ $0111_2 = 7_{10}$ 1101_2 - «-»

$0011_2 = 3_{10}$ $1000_2 = 8_{10}$

$0100_2 = 4_{10}$ $1001_2 = 9_{10}$

(i) Упакованный формат

Рис. 13.2.

(ii) Неупакованный формат

Рис.13.3.

Пример:

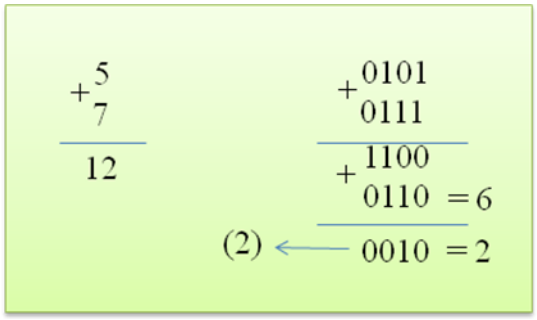


Рис. 13.4

II курс

Лекция №1

04.09.2010

Наше желание – заглянуть внутрь структуры.

Рис. 1.1 (основные логические элементы)

Элементы, играющие пассивную роль, лежат в левой части схемы. Элементы, играющие активную роль, справа.

Указатель команд (специально выделенный регистр) – фактическое воплощение инициатора программы выполнения.

Регистр состояния (иногда, «регистр флагов») – (в некоторых случаях нас могут интересовать мелкие изменения в процессе выполнения программы) регистр, который фиксирует микро изменения во внутреннем состоянии процессора.

Регистры управления памятью – некоторые регистры, обслуживающие процессы конвертации и преобразования адресов из одной системы в другую.

Арифметико-логическое устройство – выполняет все действия, вычисления процесса, инициированные нашей программой.

Устройство управления – логический модуль процесса, «говорящие» АЛУ, что конкретно ему необходимо делать.

Устройство управления памятью

Устройство управления шиной – логические элементы, планирующие загрузку магистрали.

3 нижних элемента:

- Очередь команд

- КЭШ команд

- КЭШ данных

1) Рассмотрим теперь каждый элемент более подробно

Регистр общего назначения (General Purpose Registers)

Временное сохранение данных при выполнении команд программы (под данными понимаются данные и адреса). Идейно, можно разделить РОН на Регистры данных и Регистры адресов (например, указатель стека). Данные регистры работают, соответственно, с данными и адресами.

Лекция №2

8.09.2010

Адрес, занимающий стек 32, часто не является целым объектом длины 32, а некоторой группы (таблицы) элементов, определяющих его.

Рис. 2.1 (нет в наличии)

Указатель команд (Instruction Pointer)→ хранит адрес следующей команды.

Последовательность действий при выборе команды:

1) Выборка команды из ОП или очереди команд по текущему значению IP
2) Декодирование команд
3) Изменение значений IP
4) Опрос подсистемы прерываний (например, нажатие клавиш клавиатуры при выполнении команды вызывает прерывание команды, для выполнения действия клавиши)
5) Выполнение команды

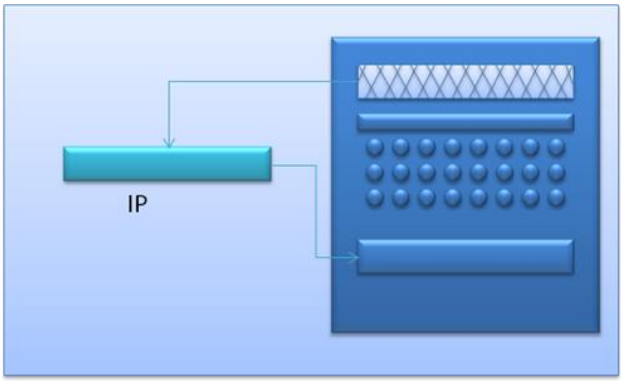


Рис.2.2

Выполняем команды по A, пока не встретили вызов некоторой операции B. Переходим в B, выполняем, возвращаемся в A и продолжаем выполнение в указанном порядке. Но как понять, на какой адрес вернуться после выполнения B? Этот адрес сохранен в IP, и сохраняет до момента нашего возвращения.

Изменение значения IP

Возможны два случая:

1) текущая команда не является командой передачи управления (команда перехода)
2) текущая команда есть команда передачи управления

В первом случае: значение IP увеличивается на длину текущее команды после ее декодирования.

Во втором случае: (важно понимать, что команды CALL и GO TO на командном языке есть практически одно и то же, с той лишь разницей, что при CALL сохраняется обратный адрес, а при GO TO нет) любое значение IP явно или неявно определяется аргументом команды передачи управления.

CALL B, jmp L – примеры явного определения IP.

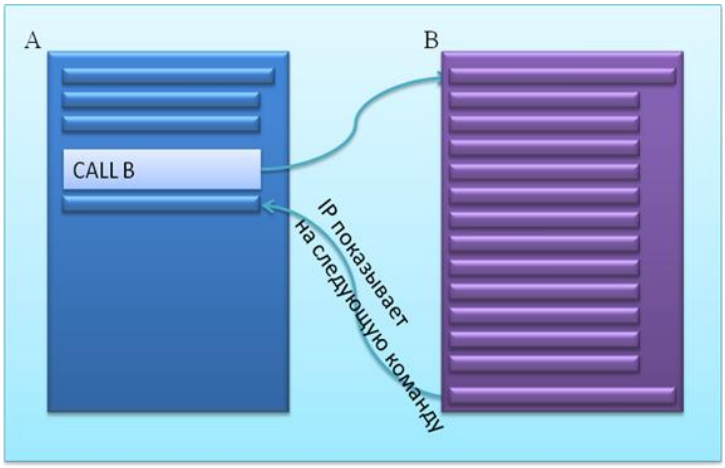


Рис. 2.3 – косвенные переход – пример неявного задания IP.

Регистр Состояния (Status register, Flag register, Program Status Word)

Фиксирует внутренние изменения состояния процессора и результаты выполнения команд. Но результат выполнения команд здесь нужно понимать, как микро результаты выполнения микро задач (0 или не 0, например).

Вспомним понятие Flag – индивидуальные двоичные разряды (или очень малые их группы), регистрирующий некоторый элемент в состоянии выполнения команды в виде 0 и 1. (см. рис. 2.4).

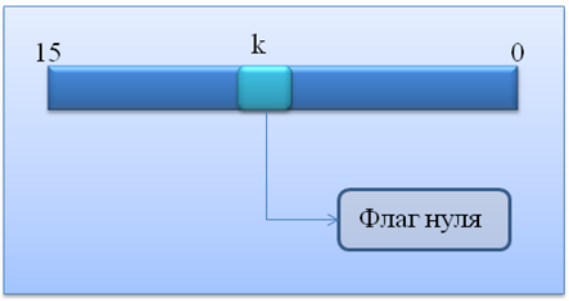


Рис 2.4

Типовые флаги состояния процессора.

CF (Carry Flag) – флаг переноса – равен 1, если перенос разряда за пределы разрядной сетки. В противном случае равен 0.

Теперь пусть выброс БЫЛ. Тогда надо охарактеризовать причины этого происшествия:

- переполнение разрядной сетки. Нечетко определен выброс элемента за младший член (см. рис 2.5), но мы его определим позже.

- команды сдвига

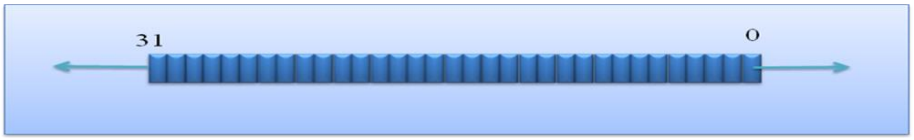


Рис 2.5

PF (Parity Flag) – флаг паритета, или флаг четности.

Контроль паритета – простейший способ контроля правильности результата. (Рис.2.6)

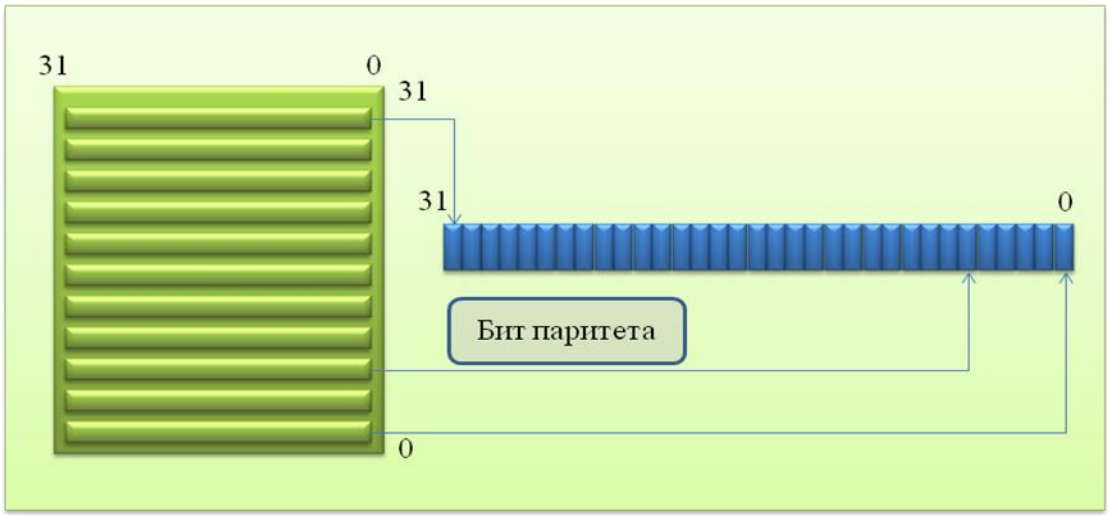


Рис. 2.6.

Принимается соглашение, что в каждом элементе памяти всегда четное количество бит. Например, договоримся, что в каждом числе четное количество 1, а у нас получилось нечетное. Тогда в бит паритета добавляется 1.

Следовательно, флаг паритета дополняет количество 1 в машинном элементе данных до принятого в системе.

ZF (Zero Flag) – флаг нулевого результата – равен 1, если результат выполнения команды равен нулю. В противном случае равен 0.

Пример:

Sub B, A - из A вычли B и результат помещаем в B

jnz L - переход во нулевому флагу

ИЛИ

Test B, A - аналогично Sub, но не заменяет B на результат, а только проверяет, что получится.

SF (Sign Flag) – флаг знака – равен 1, если результат выполнения команды меньше нуля.

И равен 0, если результат больше нуля.

OF (Overflow Flag) – флаг переполнения – равен 1, если возникло переполнение в команде обработки целых чисел со знаком.

$0101 = 5_{10}$

1010

$\underline{\quad 1^{+}}$

$1011 = -5_{10}$

Регистры управления памятью (Memory Management Register)

Используются устройством управления памятью, для преобразования адресов.

Лекция №3

11.09.2010

Арифметико-логическое устройство (Arithmetic-logic Unit)

Реализует операции, предусмотренные машинными командами, но не интересуется, зачем это делалось – выполняет команды Управляющего устройства.

Устройство управления (Control Unit)

1) Выборка команд
2) Расшифровка кода операции
3) Декодирование и извлечение операндов
4) Передача или загрузка операндов в АЛУ
5) Управление работой АЛУ

Устройство управления памятью (Memory Management Unit)

1) Преобразование адресов
2) Реализация механизмов защиты памяти
3) Разделение адресного пространства задач и операционной системы (абстрактная картинка)

Устройство управления шиной (Bus Control Unit)

1) Синхронизация работы на системной шине
2) Буферирование данных
3) Разрешения возможных конфликтов на системной шине

Очередь команд (Instruction Queue)

Временное хранение команд перед их передачей на выполнение.

КЭШ данных (Data Cache)

Сохранение часто используемых данных

КЭШ команд (Instruction Cache)

Сохранение часто используемых команд

Лекция №4

18.09.2010

Пример микропроцессорной архитектуры: x86

Год создания 1979 версия 8086

- Разрядность регистра:16
- Разрядность адреса:20
  Сегментная модель организации памяти

- Данных: 16
- Целая арифметика
- Режим реального (*физического*)адреса

Сопроцессорная конфигурация 8086/8087.

Негативный момент процессора 8086 то, что он 16-радрядный, без возможности реконструкции до 32-ух.

«8087 одно из самых выдающихся изобретений человечества!»

Параллельные очереди команд в сопроцессоре. Это ведет к возможности рассинхронизации, влекущей к временной задержки работы обоих процессоров.

Год создания 1980 версия 8088/8087

- Разрядность регистра:16
- Разрядность адреса:20
- Данных: 8
  Было сделано ради удешевления.

- Целая арифметика
- Режим реального адреса

В итоге производительность уменьшилась всего в 10%, выиграв в цене большие проценты.

Год создания 1980 версия 80186

- Разрядность регистра:16
- Разрядность адреса:20
- Данных: 16
- Целая арифметика.
- Режим реального адреса.
- Встроенный контроллер прерывания.

Встроенный контроллер – непосредственное обслуживание контролером при внешней команде.

Рис 4.1.

Режим болинга - проверка внешних команд через некоторый период.

Год создания 1982 - версия 80286/80287

80287 – арифметический процессор.

- Разрядность регистра:16
- Разрядность адреса:24
- Данных: 16
- Целая арифметика
- Режим реального адреса и Режим виртуального защищенного адреса.

Год создания 1985 - версия 80386/80387

- Разрядность регистра:32
- Разрядность адреса: 32
- Данных: 32
- Целая арифметика
- Режим реального адреса и Режим виртуального защищенного адреса.
- Аппаратная поддержка страничной виртуальной памяти. (см. рис. 4.2)

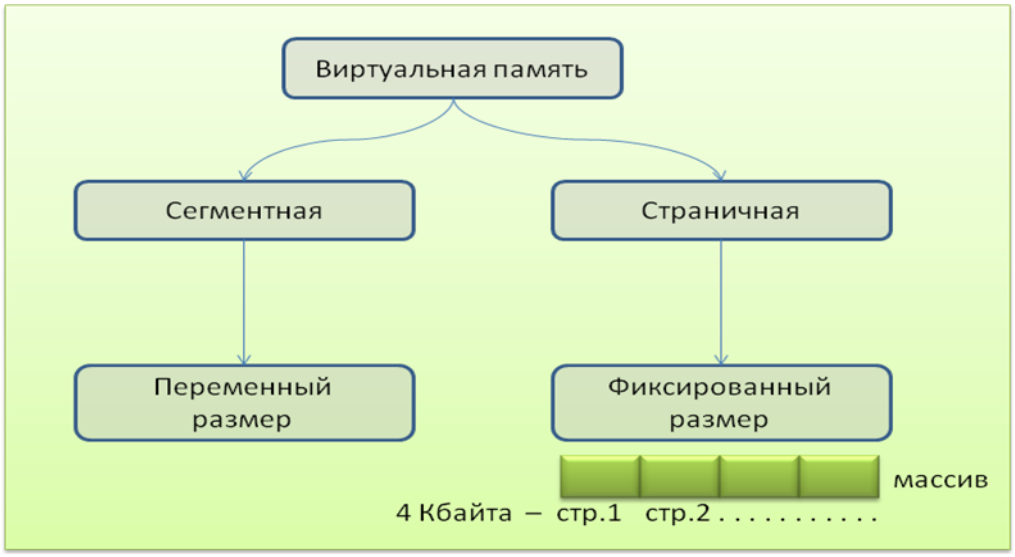


Рис 4.2

Сегмент – некий фрагмент, блок памяти, имеющий переменный адаптированный размер.

Страница – единица памяти, обладающая фиксированный размер. При нехватке памяти берется следующая страница.

Год создания 1987 версия 80486 (80386 + вычисления с плавающей точкой)

Реализован принцип внутреннего удвоения тактовой частоты – при выполнении внутренних операций используется частота в два раза больше, чем у входного тактического генератора.

Год создания 1990 → Наше Время

Pentium X, где X = 1, … , 4.

1. Конвейерная обработка команд (см. предыдущий семестр)
2. Использование КЭШ-памяти команд и данных
3. Наличие внутреннего параллелизма
4. Увеличение количества рабочих регистров
5. Повышение тактовой частоты

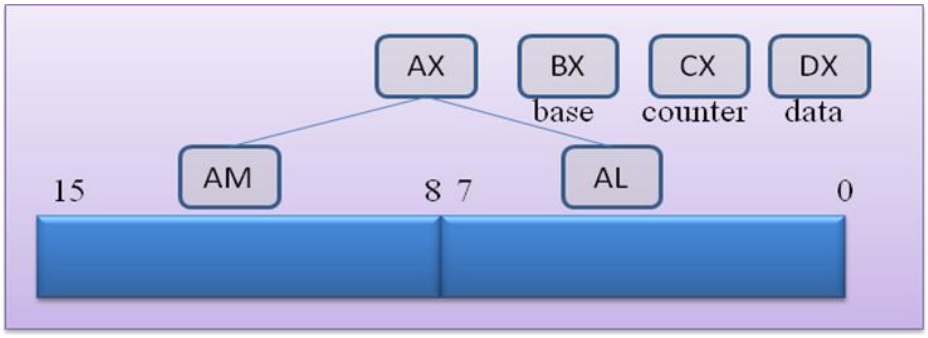
Лекция №5

22.09.2010

x86, регистр реального адреса

Регистры общего назначения

Рис. 5.1



BX-(Base - база) – предполагается, что регистр может использоваться для базирования (см. рис.5.2). Но он имеет двоякую роль, так как может хранить как числовые аргументы, так и базы. Его двоякость называется асимметрией.

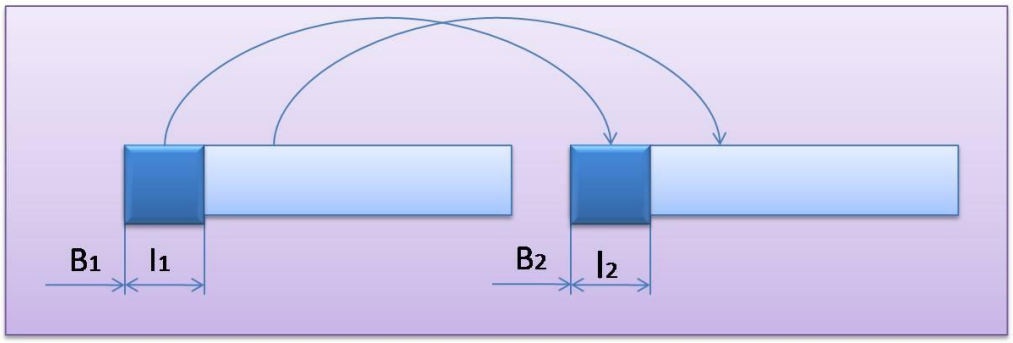


Рис. 5.2.

Напоминание: mov Dst, Src Dst ← Src

mov AX, Src1

mov DX, Src2

add AX, DX

Указатель команд

IP (Instruction pointer)

1) Хранит адрес следующей команды

2) Адрес следующей команды заносится в IP сразу после декодирования текущей команды

Адресные регистры

Регистры индекса (Рис. 5.3.)

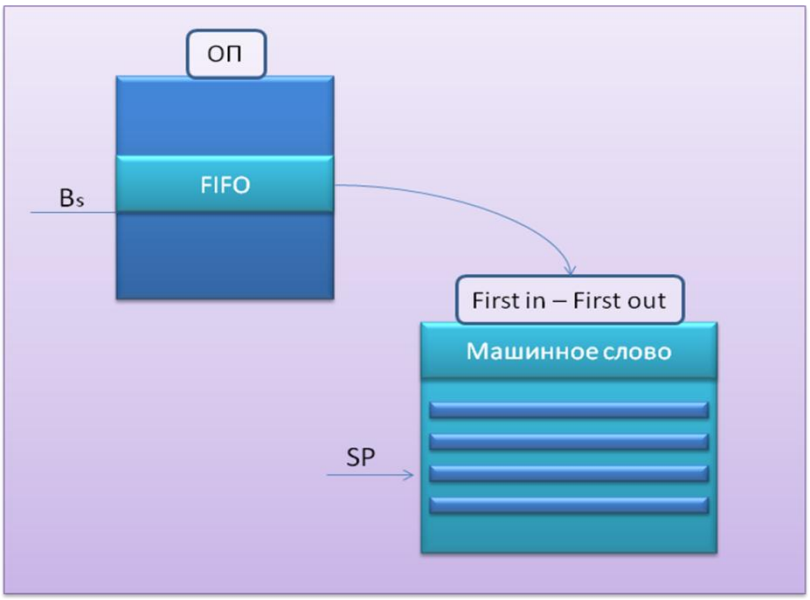
SI (Source Index)

DI (Destination Index)

Регистры управления стеком

SP (Stack Pointer)

BP (Base Pointer)



Команды работы со стеком Рис. 5.3

Push Src	SP ← Sp – z – добавить в стек
	(SP) ← Src
Pop Dst	Dst ← (SP) – взять из стека
	SP ← SP+2

Pusha – добавить в стек все регистры общего назначения

Popa (читать ПОП А) – Загрузить из стека –«-

Pushf – добавить в стек содержимое регистров флагов (PSW)

Popf – загрузить PSW с вершины стека

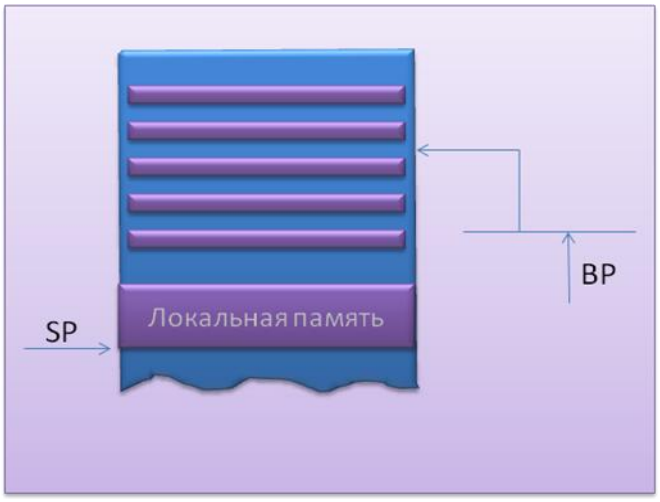


Рис.5.4

Слово состояния программы (Program Status Word)

Мы рассматривали ранее только универсальные флажки, приемлимые всем системам. Каждая система может добавлять свои.

Рис 5.5

CF (Carry flag) – флаг переноса

PF (Paring flag) – флаг паритета

ZF (Zero flag) – флаг нулевого результата

SF (Sign flag) – флаг отрицательного результата

OF (Overflow flag) – флаг переноса

Лекция №6

25.09.2010

AF (Aux Flag) – вспомогательный флаг переноса. Полный аналог флага CF, только относительно четвертого бита результата (если четвертый бит переполнен, то это фиксируется в третьем бите).

Двоично-кодированные десятичные числа.

$0 = 0000_2$

: 79 = 0111.1001

$9 = 1001_2$

Флаги управления

1) **DF (Direction Flag)** – флаг направления (к примеру для цепочечной команды пересылки данных movs)

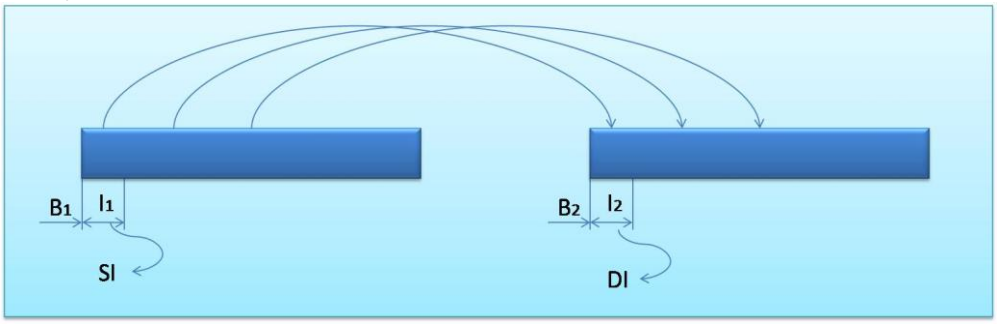


рис.6.1

Если DF = 0, то цепочечные команды исполняются в направлении возрастания адресов. Если DF = 0, то цепочечные команды выполняются в направлении убывания адресов. Stol – команда установки DF; clol – команда сброса DF.

2) **IF (Interrupt Flag)** – флаг прерываний.

Концепция прерываний. ,Код прерываний – мы останавливаем процессор, выполняем команду с большим приоритетом.

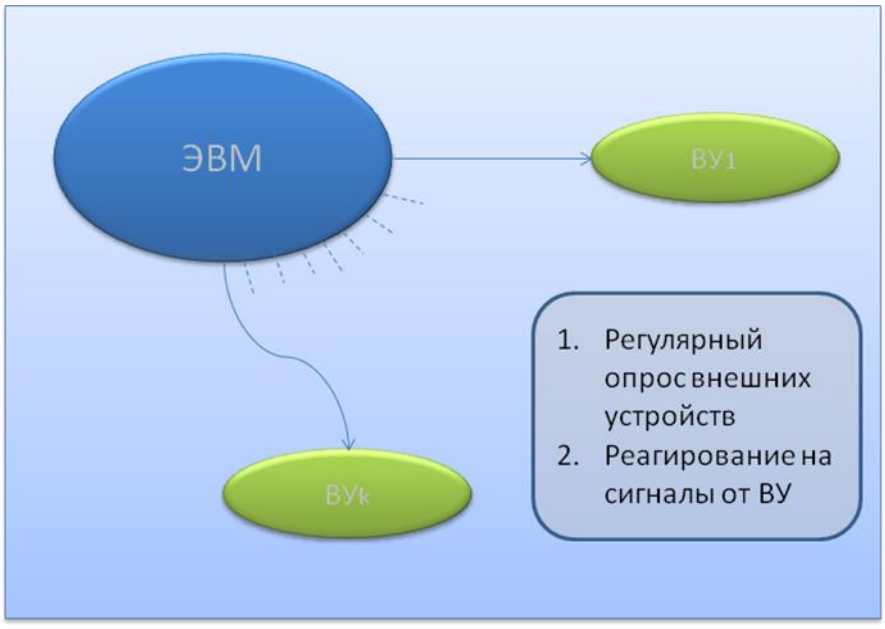


Рис.6.2 (историческая идеология –режим регулярного кругового опроса).

На старых моделях процессор останавливался примерно каждые 5 секунд, чтобы опросить все устройства о необходимости выполнения каких-либо команд. Общее: выполняется сканирование и опрос; разница: кто и когда это делает (в первом это зашито в ОП, для второго это часть семантики машинных команд)

1. Реакция на запросы на прерывание встроена в семантику машинных команд.
2. Переход по прерыванию выполняется до начала использования текущей задачи.

Рис.6.3

Если IF = 0, то переходы по прерываниям запрещены.

Если IF = 1, то переходы по прерываниям разрешены.

Команды управления флагом IF:

1. sti – установить флаг

2. cli – сбросить флаг

3) **TF (Trace Flag)** – флаг трассировки

Если TF = 1, то прерывание возбуждается после каждой машинной команды (режим трассировки). Режим работы микропроцессора, дающий возможность для работы отладчика. Команд для работы с флагом TF нет: изменение состояния TF возможно только путем его записи в стек (см. Pushf и Popf)

Регистры управления памятью (сегментные регистры – Segment Registers)

CS (Code Segment) – сегмент кода

DS (Data Segment) – сегмент данных

SS (Stack Segment) – сегмент стека (также именуемый SUPER Segment)

ES (Extra Segment) – дополнительный сегмент данных

Сегмент – некоторая область памяти, в которой расположена целевая информация. (логическая единица при разрезании памяти)

8086

Регистры: 16 бит

Линия данных: 16 бит (рис.6.4)

Линия адреса: 20 бит

Базы сегментов (разделенные на 16) хранят адреса сегментов различного типа.

Модель формирования исполнительного адреса

SR – Segment Register

$SR_{16} << 4 =_{+} SR'_{20}$

Offset (рис.6.5)

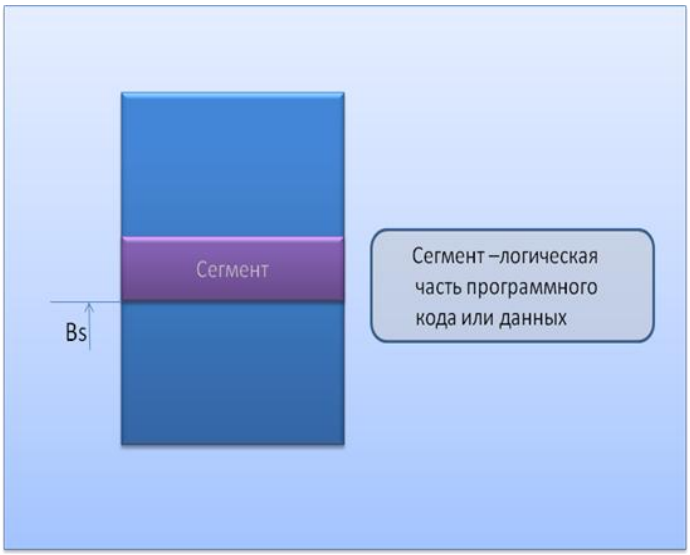


Рис. 6.4.

Рис.6.5.

Лекция №7

2.10.2010

Теперь будем говорить, что в SP лежит не полный адрес. К нему необходимо добавить базовый адрес SS:

SS: SP – адресация стека

CS: IP – адресация команд

DS и ES управляют операциями пересылки памяти.

Одним из главных видов передачи управления (при надежде на возвращение обратно) являются функции (подпрограммы).

Виды передачи управления

Работа с подпрограммами

Подпрограмма; Функция; Процедура (хотя слово «процедура» используется обычно только внутри программы, так там нет необходимости конкретизировать этот вопрос).

Имя команды не несет никакого значения.

Подпрограмма – p(a, b), но нельзя q = p(a, b)

Функция – y = Sin(x)

Главная программа, на самом деле, тоже кем-то запускается. Она вызывается ОС.

Рекурсивный вызов (рекурсия) – вариант вызова, когда подпрограмма перезапускает сама себя.

RET – выполняет возврат в ОС.

Основой для создания рекурсивных вызовов является стековая модель памяти. Единственная проблема такой модели – рекурсивный момент должен быть конечен, иначе в некоторый момент времени стек, использующийся для данной процедуры, закончится.

Семантика программы CALL

Вызовы:

1) прямой (адрес явно указан в аргументе команды)
2) косвенный (в аргументе стоит ссылка на место, откуда можно забрать адрес команды). Например, в языке C: int **p; (указатель)

Если A напрямую ссылается на B, то все хорошо. Но если A ссылается сначала на C, то необходимо сохранить Сегмент 1, перейти в Сегмент 2, а потом вернуться. Короткий переход – внутрисегментный. Длинный переход – межсегментный.

И прямой, и косвенный вызовы делятся на короткие и длинные переходы.

Шаги команды CALL:

1. Сохранение адреса возврата в стеке. Значение указателя стека меняется на величину командного слова.
   SP ← SP-2; (SP) ← CS; SP ← SP-2; SP ← IP

   CS ← Segment (адрес перехода)

   IP ← Offset (адрес перехода)

   RET – автоматически вытекает из CS и IP.

2. Загрузка адреса перехода из аргумента команды.

Семантика команды RET

1. Восстановление адреса команды из стека.
   IP ← (SP); SP ← SP+2; CS ← (SP); IP ← SP+2;

Лекция №8

06.10.2010

В машинном языке нет понятия параметра. Там существует «философия условного места» Перед тем, как вызвать процедуру B, процедура A кладет в некую область параметры, а потом вызывает B. Но эта область не контролируется ни A, ни B. То есть, B не знает, что туда необходимо положить, а A не знает, что от туда будут брать.

Как область передачи параметров берется стек.

Вызывающая процедура (A)	Вызываемая процедура (B)
Push p	proe far
Push q (кладем их в наш стек)	push BP
Call B	mov BP, SP
Cs и Ip кладутся в стек, как результат Call	mov AX, [BP+8]
	mov DX, [BP+6]
Far – межсегментный вызов	add AX, DX
Near – внутрисегментный вызов	
BP – адрес базы	
Теперь необходимо начинать подготовку к возврату	pop BP
Далее применяем команду RET (описанную в пр. лекции)	RET 4
Заканчиваем выполнение процедуры	End p

Реальные Компиляторы настолько суровые, что используют RET вообще без параметра.

Прерывание. Программа обработки прерываний.

Прерывание дало возможность реагирования на внешние события.

При выполнение команды выполняются следующие этапы: 1. Декодирование команды

2. Опрос подсистемы прерываний

3. Выполнение команды

Прерывания:

1) Программные (активируются посредством машинных команд)
2) Аппаратные (запрос приходит от аппаратуры)

2.1) Внутренние (реагирования на исключительные ситуации внутри процессора, например деление на 0). Отказы на реагирование в этом случае не принимаются.

2.2) Внешние. Процессор имеет право не реагировать, если не считает это необходимым.

2.2.1) Маскируемые – можно отменить

2.2.2) Немаскируемые – отменить нельзя

Лекция №9

9.10.2010

Продолжаем разговор о прерываниях

Общая команда прерываний

int k, где k – номер обработчика прерываний

Однобайтовое прерывание

int ~ int3

Прерывания по переполнению

int0 ~ int4, но только если OF=1

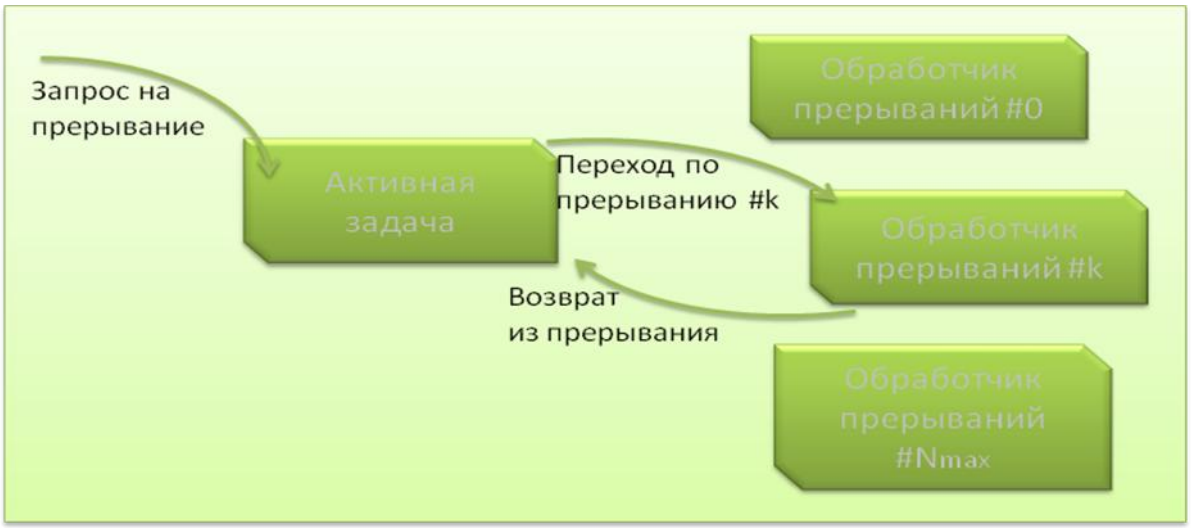


рис. 9.1

Таблица векторов прерываний

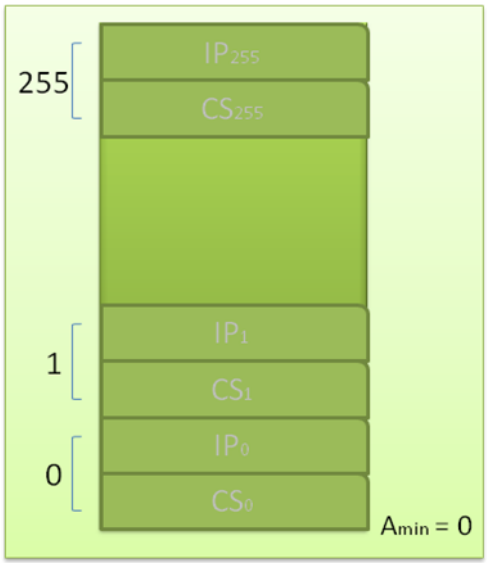


Рис. 9.2

Зарезервированные векторы прерываний (режим реального адреса x86)

Вектор 0: ошибка деления

Вектор 1: прерывание одношаговой работы (TF=1)

Вектор 2: немаскируемое прерывание

Вектор 3: команда int

Вектор 4: команда int0

Мы связали некоторые вектора с событиями. И таким образом создали маршрут защиты реакции. Вопрос: зачем могут понадобится реакции с такими векторами?

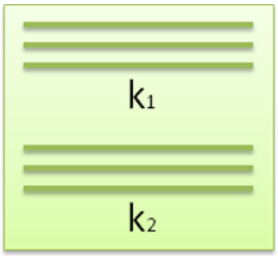


Рис. 9.3

Подставляем в нашу программу команду с векторами прерываний:

Рассмотрим MS Dos.

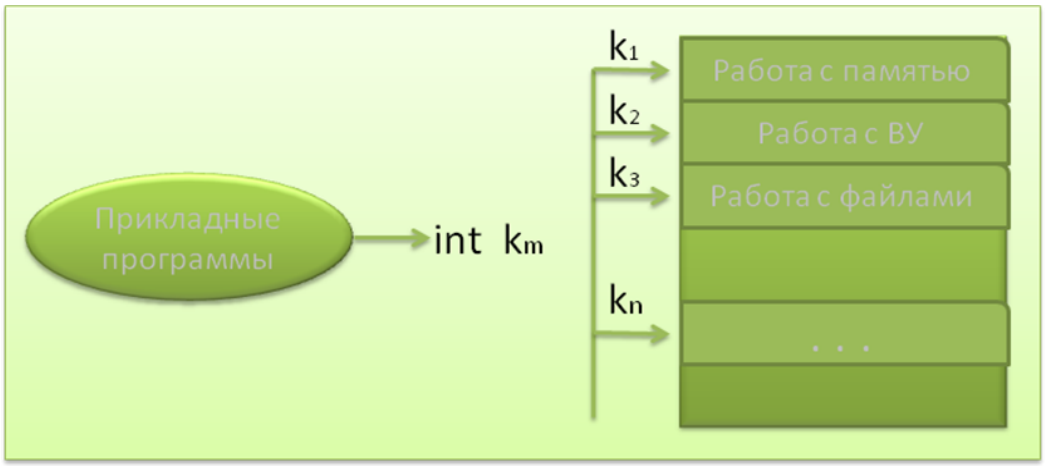


Рис . 9.4

Наша программа не может работать без ОС – так как она дополняет ее.

Семантика прерываний → последовательность прерываний

1) Сохранение PSW в стеке
2) Сброс IF и TF
3) Сохранение CS: IP в стеке
4) Загрузка адреса перехода по требованию из вектора с №k

Семантика CALL

1) Сохранение адреса возврата в стеке (для x86 CS: IP)
2) Загрузка адреса перехода из аргумента команды

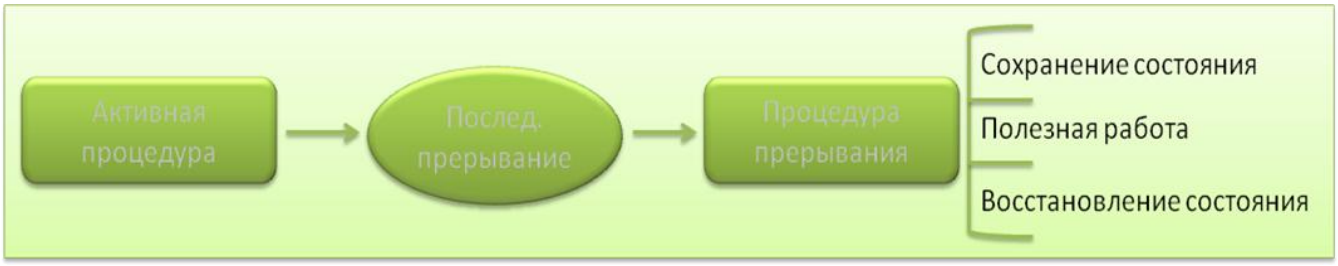


Рис . 9.5

Если TF не сбросить, то трассировка может распространиться на прерывание

Пример с сохранением: нажатие клавиши на клавиатуре → прерывание на короткое время.

Лекция №10

16.10.2010

Напомним: Последовательность прерываний

1. Сохранение PSW в стеке
2. Сброс флагов TF и IF
3. Сохранение в стеке адреса возврата (CS:IP)
4. Загрузка CS:IPиз вектора прерывания.

Рис. 10.1 (Модель обработки прерывания)

Команда возврата из прерываний iret

1. Восстановление адреса возврата из стека
   IP ← (SP)

   SP ← SP + 2

   CS ← (SP)

   SP ← SP + 2
2. Загрузка PSW из стека

Обработка внешних аппаратных прерываний

Рис. 10.2 (Модели I и II, IIIсвязи ВУ с процессором)

I – проблема в синхронизации устройств.

II – проблема в обработке прерываний.

III – Контроллер прерываний. Эта модель будет основной моделью нашего изучения.

Процессор имеет три линии подключений – две входные и одну выходную.

INTR – линия запрос на прерывания

INTA – линия подтверждения приема запроса

NMI – линия запроса на немаскируемое прерывание

Обслуживание в режиме немаскируемых прерываний

1. Схема контроля памяти
2. Схема контроля каналов ввода/ вывода
3. Сторожевые таймеры (зная приблизительные оценки времени срабатываний, ставим проверку по времени)

Каждый контроллер имеет 8 линий подключений. Рис. 10.3 (модель двух каскадных контроллеров).

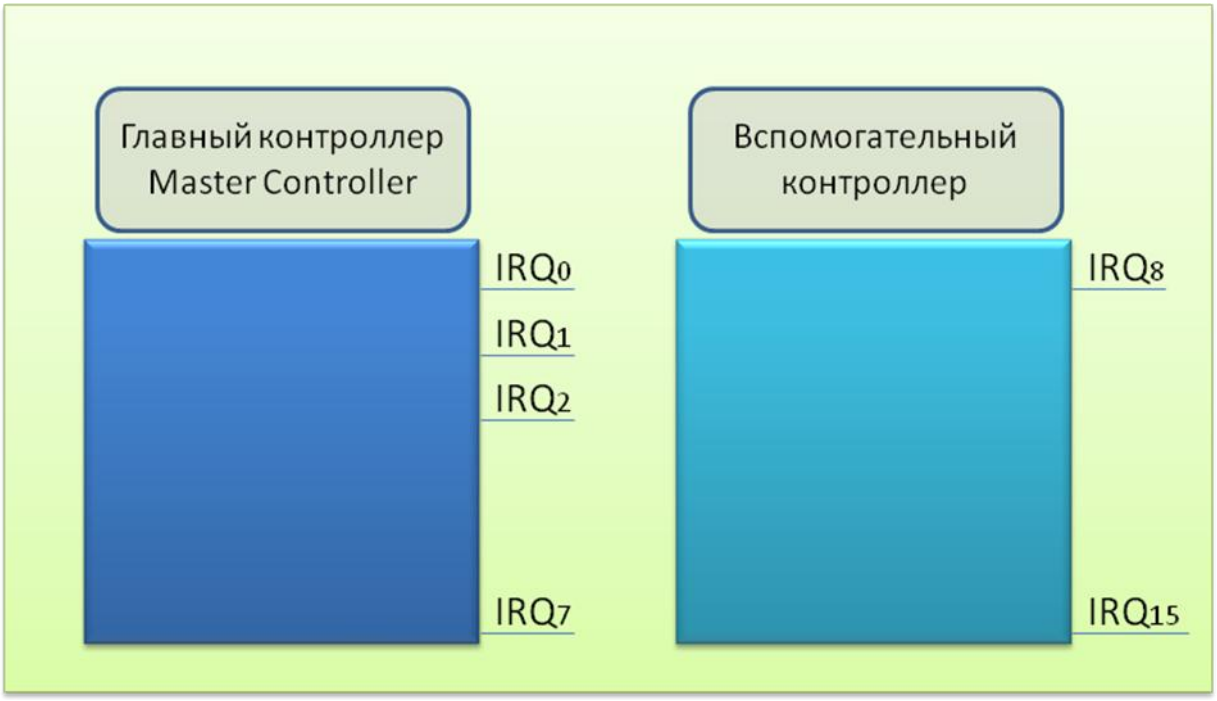


Рис 10.3

IRQi (где i = 1, … , 15) – линии подключений.

Чем меньше номер линии, тем выше ее приоритет.

Линия	Вектор	Устройство
IRQ0	8h	Системный таймер
IRQ1	9h	Клавиатура
IRQ2	Ah	Каскадный контроллер
IRQ3	Bh	Доп. последовательность передача данных
IRQ4	Ch	Основной послед передача данных
IRQ5	Dh	Резерв
IRQ6	Eh	НГМД
IRQ7	Fh	Параллельная передача данных

Системный таймер


Базовая частота
синхронизации
0
1
2
циклы регенерации
памяти
звук
сторожевой таймер
Регистр счетчик – "…в него кладется числовая величина, при каждом импульсе сбрасывается 1, если стало «0» подается сигнал…"

Рис. 10.4

Лекция №11

20.10.2010

Способы передачи данных между PC1 и PC2 (рис. 11.1)

1. Последовательная передача данных
2. Параллельная передача данных

Параллельная передача много проще в реализации, так как не стоит проблема перепутывания данных. Она очень хороша при передаче небольших объемов, однако лимитирована по объему данных.

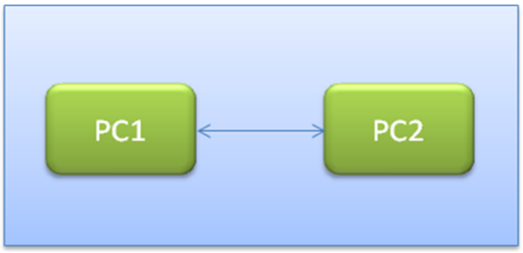


Рис. 11.1

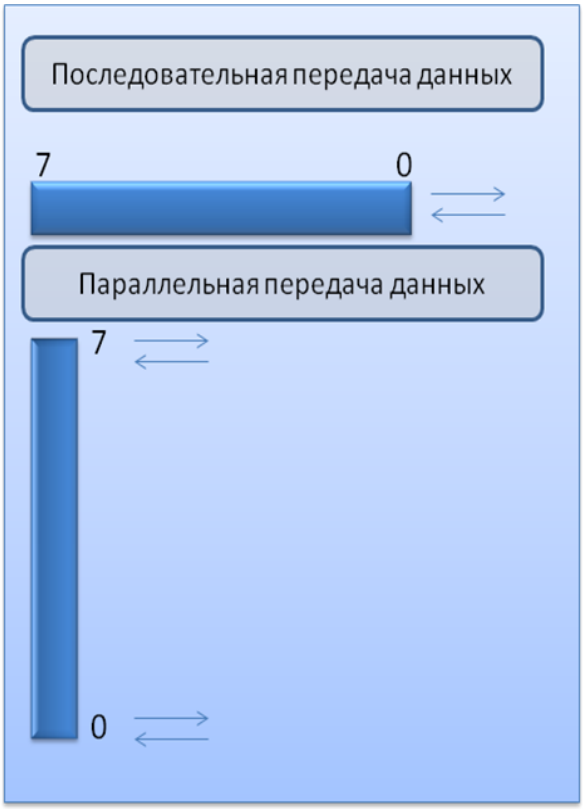


Рис. 11.2

Последовательная передача обычно делится на:

А) Синхронная передача – осуществлена синхронизация по времени. То есть на один «удар часов» PC1 отправляет элемент и PC2 знает, что в этот момент ему придет следующий элемент. Однако в общем случае синхронизировать PC1 и PC2 невозможно.

Б) Асинхронная передача. (Рис.11.3.)

Рис. 11.3

Протокол передачи –

1) Способ кадрирования информации
2) Дисциплина передачи кадров

(Напоминание: приоритеты убывают при возрастании номера в пределах одного контроллера)

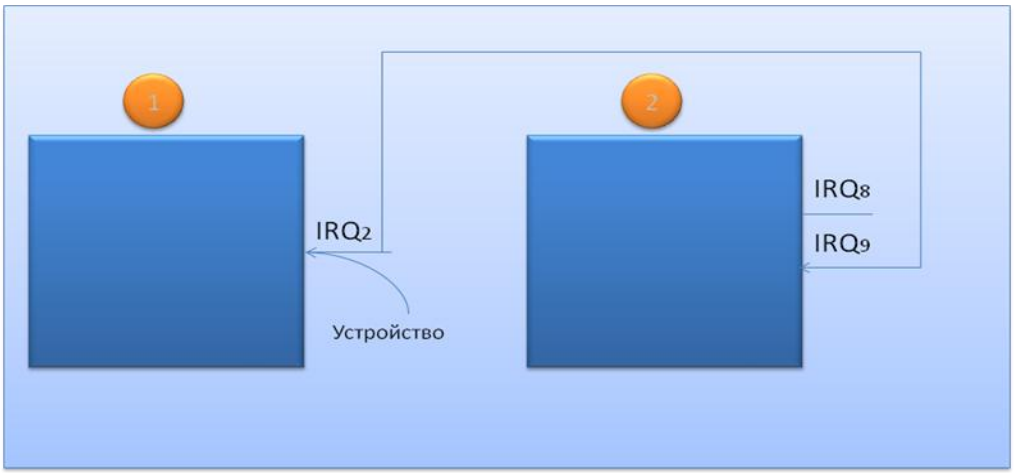


Рис. 11.4

Линия	Вектор	Устройство
IRQ8	70h	Часы реального времени
IRQ9	71h	Переадресация каскадного прерывания
IRQ10	72h	Резерв
IRQ11	73h	
IRQ12	74h	«мышка»
IRQ13	75h	Арифметический процессор
IRQ14	76h	Дисковая подсистема
IRQ15	77h	Резерв

Аналитический процессор может быть возвращает код ошибки при ошибке в арифметических действиях.

Логическая модель контроллера прерываний

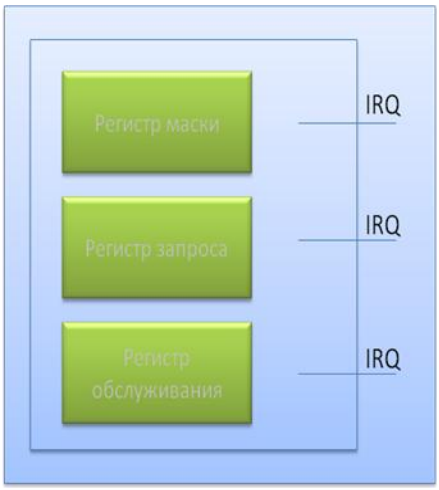


Рис. 11.5

Регистр маски: запреты или разрешения запросов прерываний на отдельных линиях.

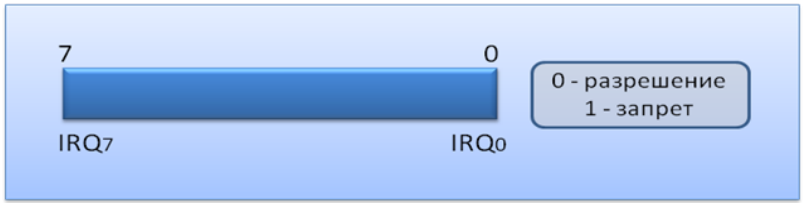


Рис. 11.6

Последовательность аппаратного прерывания

1) Поступление запроса на линию IRQ контроллера прерываний.
2) Проверка регистра маски.
   Обработка или запрет. При запрете происходит прекращение.
3) Сравнение приоритетов поступившего и текущего запросов.
4) Передача запроса процессору по линии INTR.
5) Проверка флажка IF в PSW.
   Если IF=1 то прерывания разрешены.
6) Подтверждение приема запроса по линии INTA.
7) Передача процессору номера вектора прерываний по линии данных системной шины.
8) Реализация стандартной последовательности прерывания.
   - Сохранение PSW в стеке.
   - Сброс IF TF
   - Сохранение адреса возврата
   - Загрузка адреса обработки прерываний из вектора с заданным номером.
9) Сброс линии IRQ контроллера прерываний.

Работа x86 в режиме защищенного адреса

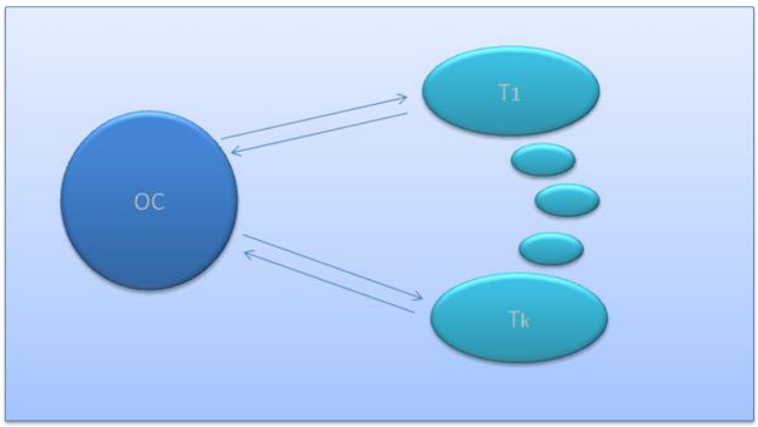


Рис. 11.7

Возникающие проблемы:

- Задачами надо управлять.
- Операционную программу нужно защитить от прикладных программ.
- Прикладные программы нужно защитить друг от друга.
- Прикладные программы должны иметь доступ к операционной системе.

Лекция №12

23.10.2010

Идея многозадачного режима (рис. 11.7). Чтобы было похоже на одновременное выполнение нескольких задач, выполняется переключение между ними. Если переключение идет со скоростью 10раз в 1 миллисекунду, то малые остановки будут не заметны.

Основные концепции защищенного режима

1. Замена физических адресов адресами виртуальными
2. Отделение адресного пространства ОС от адресных пространств прикладных программ
3. Отделение адресных пространств прикладных программ друг от друга (т. е. программы не могут вызывать функции другой напрямую)
4. Разделение уровней привилегий

Модель адресного пространства защищенного режима. (рис. 12.1)

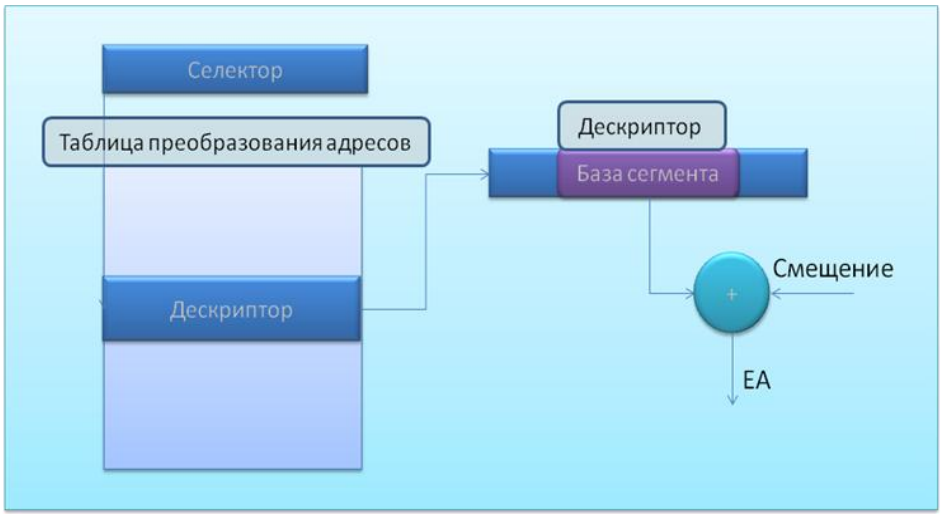


Рис. 12.1.

SR – сегментный регистр (CS, DS, ES, SS)

Таблица преобразования адресов (для ОС) – дескрипторная таблица.

Дескрипторные таблицы

1. Глобальная дескрипторная таблица (Global Descriptor Table - GBT)
   - существует в единственном экземпляре

   - используется ОС

   - доступна лишь в режиме высоких привилегий

   - находится в ОП (т. е. работает медленно)

2. Локальная дескрипторная таблица (Local Descriptor Table - LBT)
   - своя для каждой прикладной программы

- переключаются при переключении задач (на самом деле, переключение задач – суть переключение их таблиц)

- доступны в режиме прикладных программ

3. Дескрипторная таблица прерываний (Interrupt Descriptor Table - IBT)
   - аналогична GDT

   - хранит шлюзы задач обработки прерываний

Доступ к дескрипторным таблицам (регистры управления памятью)

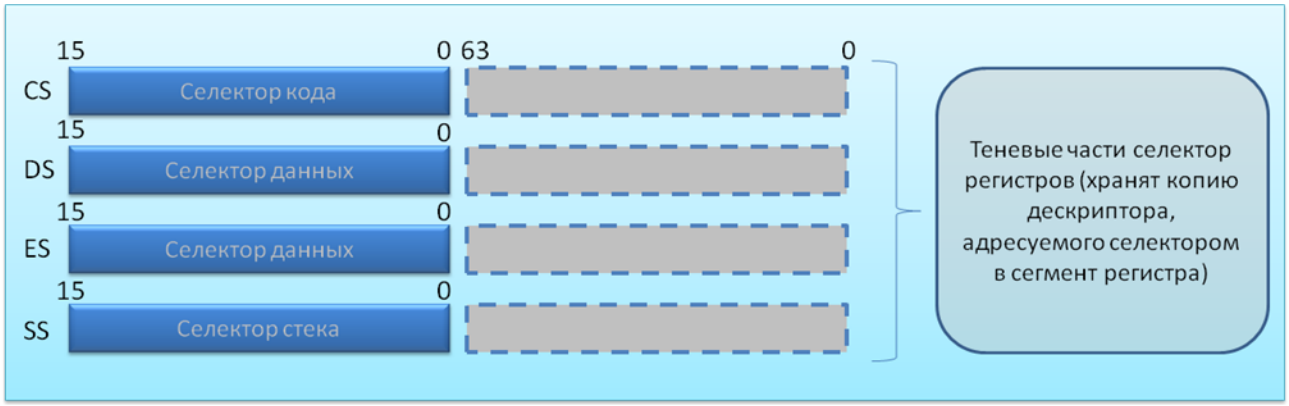


Рис. 12.2

Global Descriptor Table Register (GDTR). Рис. 12.3

LGDT – загрузка GDTR → доступны только в режиме

SGDT – сохранение GDTR → максимальных привилегий

Local Descriptor Table Resistor (LDTR). Рис. 12.4

LLDT – загрузка LDTR → доступны только в режиме

SLDT – сохранение LDTR → максимальных привилегий

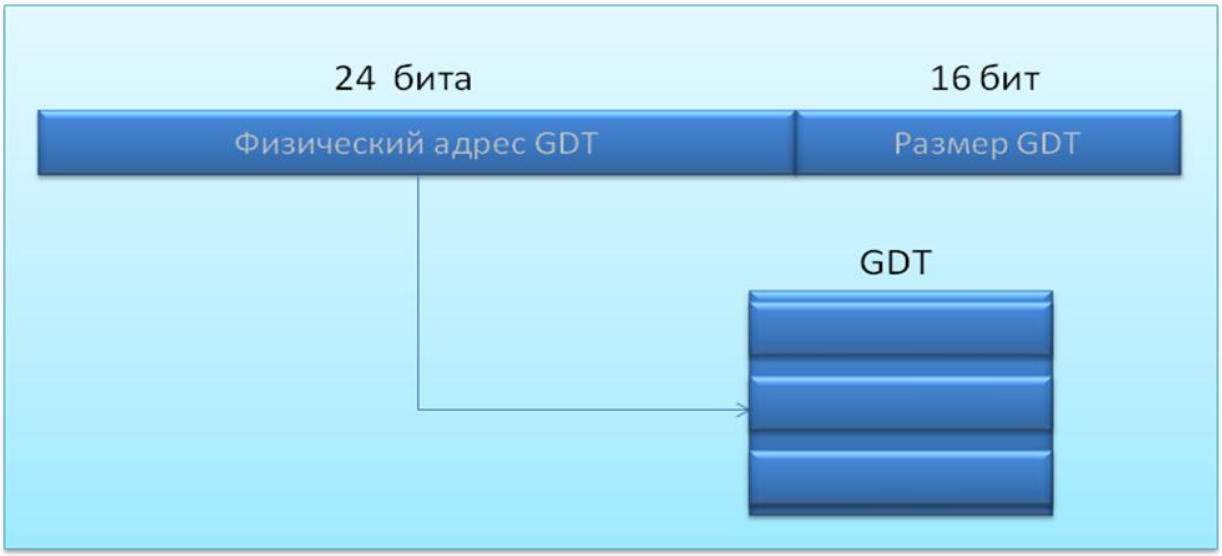


Рис. 12.3.

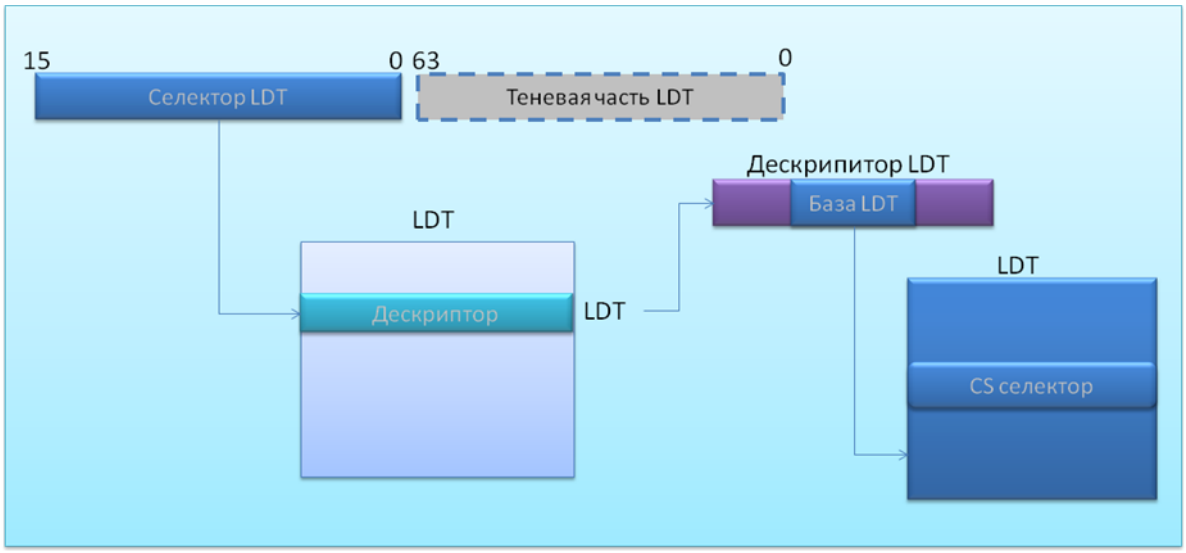


Рис. 12.4.

Уровни привилегий (кольца защиты)

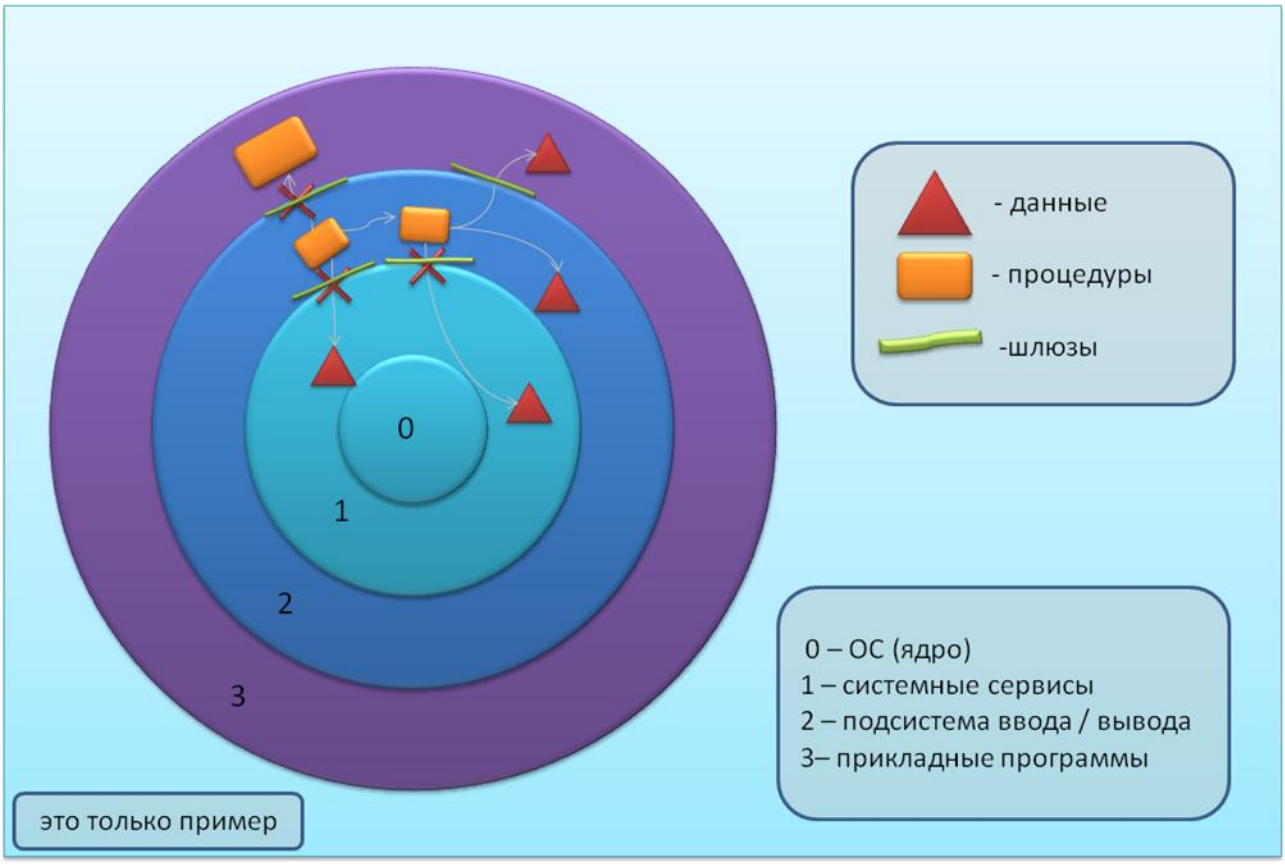


Рис. 12.5

В начале находимся во 2-ом кольце. Наша процедура может вызвать только процедуру из кольца 2. Из внешних колец она этого сделать не может. Также она может вызвать процедуру из внутреннего кольца через шлюз. Процедура может вызвать данные только из своего кольца и из внешних колец.

Лекция №13

30.10.2010

Структура селектора

Рис. 13.1

TI – выбор типа дескрипторных таблиц (TI = 0 → GDT; TI = 1 → LDT)

Index – номер входа в дескрипторную таблицу

RPL (Required Privilege Level) – используется кольцами защиты для отражения «атаки Троянского коня».

DPL (Descriptor Privilege Level) – уровень привилегий сегмента.

CPL (Current Privilege Level) – уровень привилегий текущего сегмента кода. Поле DPL текущего сегмента, сканированное на место RPL.

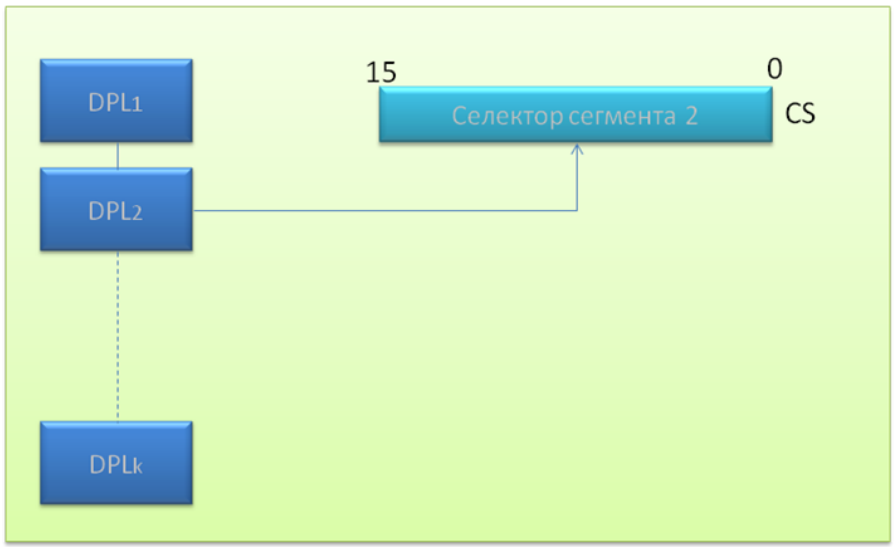


Рис. 13.2

Структура дескриптора сегмента

Рис. 13.3

Доступ – определяет тип и права доступа сегменту (т. е. что он есть, и что с ним делать)

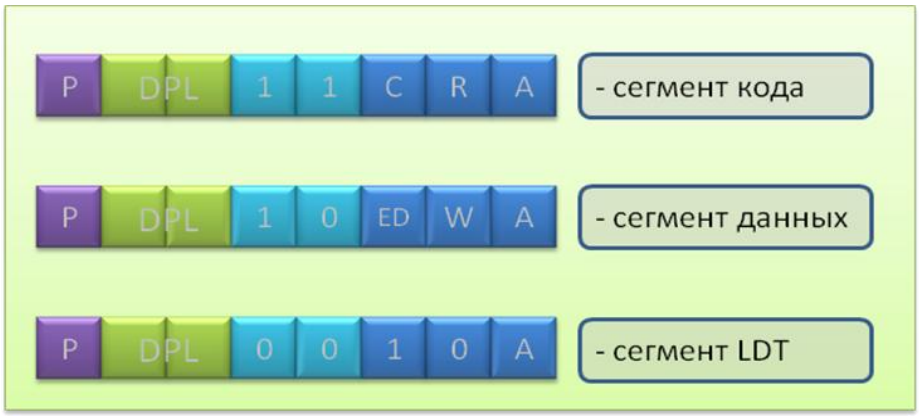


Рис. 13.4

|DPL(Descriptor Privilege Level) – уровень привилегий сегмента как номер кольца защиты.

P(Presence) – вид присутствия сегмента в оперативной памяти.

P=1 – да, P=0 – нет.

A(Access) – бит обращения(устанавливается в единицу при использовании сегмента, сбрасывается операционной системой). Используется подсистемой виртуальной памяти.

R(Read) – разрешение системы для сегмента кода (если R=1, то чтение разрешено: запись всегда запрещена, исполнение всегда разрешено).

W(Write) – разрешение записи для сегмента данных (если W=1, запись и чтение разрешено, исполнение запрещено).

ED(Expanded Down) – признак «расширяемый вниз» (Если EX=1, то элементы данных доступны в направлении убывания адресов). Используется для контроля переполнения.

C(Confermed) – признак согласованности сегментов. (Если C=1, то механизм межкольцевой защиты при борьбе процедур не используется: номер кольца не меняется, стеки не переключаются)

Лекция №14

3.11.2010

Адресное пространство задач

I. Локальные дескрипторные таблицы – есть потенциальная предоставленная нам возможность. Глобальная есть всегда, а локальную можем не создавать, так как локальная «передаётся» через глобальную.

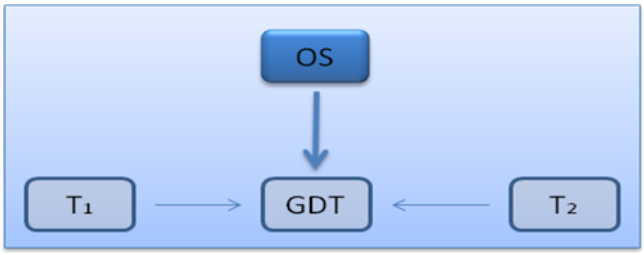


Рис 15.1. (Т.е. можно ЛТД не создавать)

II. Каждая задача может иметь что-то и в глобальной таблице. (Хотя ГДТ предназначена для ОС, и локальные есть)

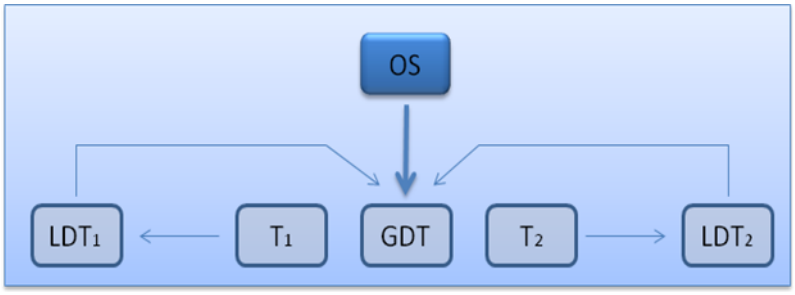


Рис. 15.2. (Т.е. можно задать ЛДТ, но никому не давать)

III. Две задачи, имеющие ЛДТ1, могут хранит некоторые дескрипторные параметры и в таблице семейства ЛДТ1, так и в ГДТ.(Т.е. можно создать ЛДТ, и предоставить её многим)

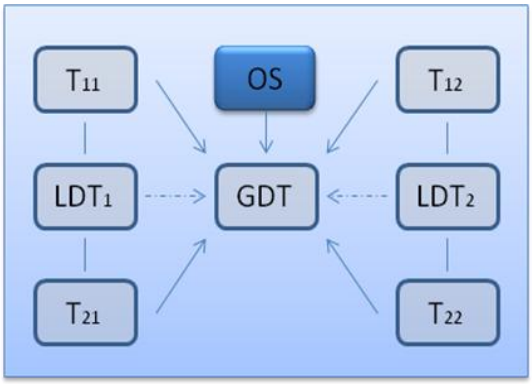


рис. 15.3

IV. Могут быть Дескрипторы в ЛДТ. Но когда мы прочтем и вынем оттуда базовые адреса, то увидим, что ведут к одному сегменту.

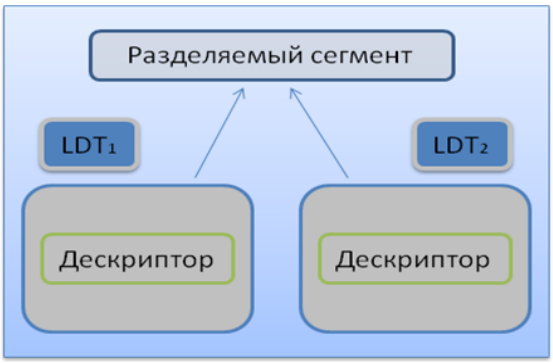


Рис.15.4

Осталось два вопроса – активации процедур и активации задач. Задачи пока оставим наконец. А с процедурами – мы уже половину сделали, а для другой есть заготовка.

(Вспомнили колечки)

Межкольцевые вызовы программ

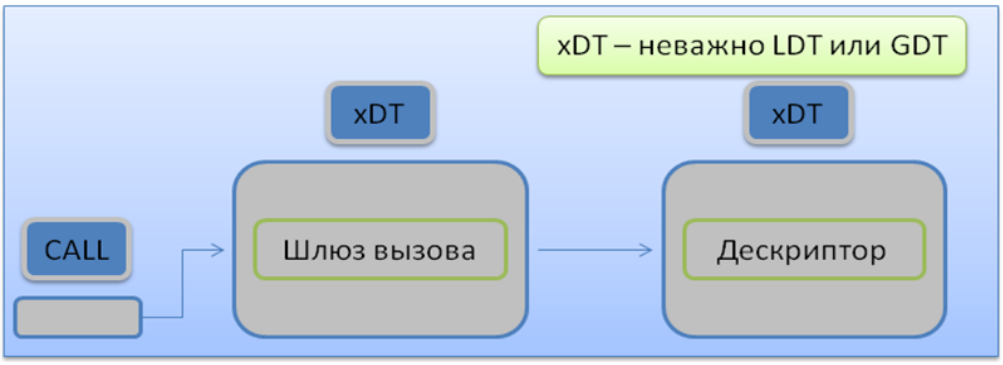


Рис15.5

Мы не напрямую идем к сегменту вызова, а идем к шлюзу вызова. А из него извлекли параметры реальной процедуры. Вполне вероятно, что дескриптор переадресует нас в новую дескрипторную таблицу, где уже есть дескриптор подпрограммы.

Внимательно рассмотрим шлюз.

Дескриптор шлюза подпрограммы

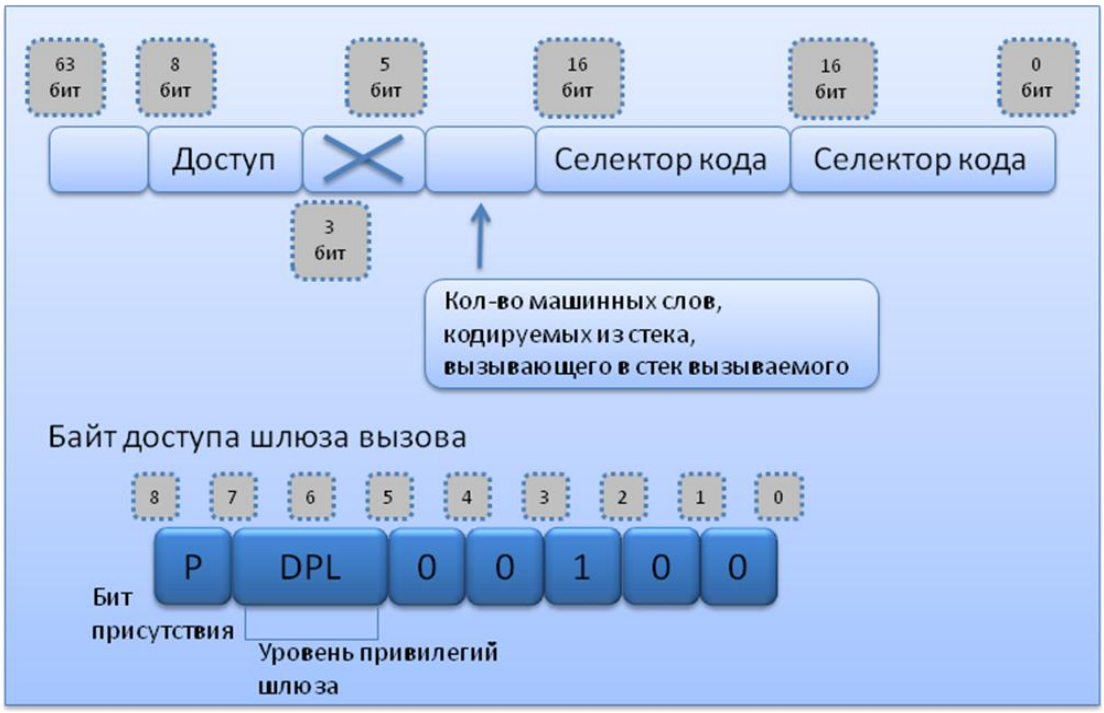


Рис 15.6

Задача шлюза не просто дать разделяемый способ, а согласовать две процедуры из разных колец, так как они берут (и кладут) в свои места(чужих не знают). Шлюз работает с машинными словами – ему сказали скопировать пять параметров – он скопировал.

Вызовы:

- Короткие
- Длинные

Смещение берется из шлюза. Смещение в аргументе команды CALL игнорируется.

Команда CALL – это аргументы моей задачи. А код задачи – относится к ОС, ОС их перекомпанует, поэтому трогать таблицы трогать не нужно – смысла нет, ею управляет ОС. Селектор – это только номер входа.

Нас интересует стек вызываемой процедуры.

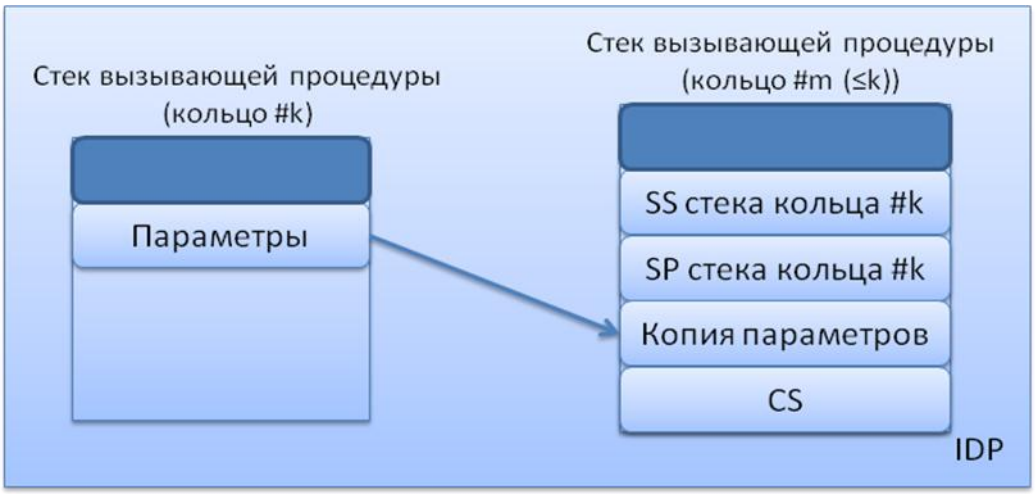


Рис15.7

Переходя из одного кольца в другое нас интересует адрес. Стек вызывающей в стеке вызываемой.

Вопрос – откуда вызывающая знает адрес вызываемой? – потом.

Семантика команды return при межкольцевых возвратах

1. Восстановление CS:IP (возврат)
2. Удаление параметров в стеках колец #k и #m
3. Восстановление стека кольца k
4. Сброс селекторов в DS и ES, если они адресуют сегмент данных кольца #m(кольца с высокими привилегиями)

Управление многозадачностью

Мы локализовали для задачи её код и адреса. Теперь пусть у нас k задач – что с ними делать?

TR (Task Register) – регистр задачи – он хранит селектор сегмента состояния активной задачи в GTD

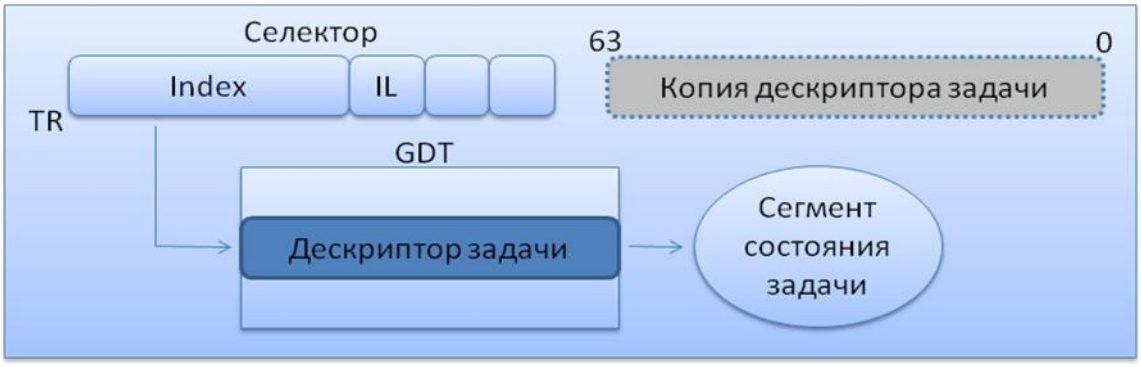


Рис 15.8

TSS (Task Status Segment) – сегмент состояния задачи

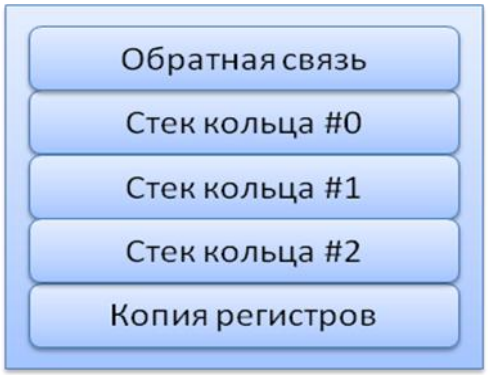


Рис 15.9

Дескриптор задачи (сегмента состояния TSS)

Рис 15.10

Выделенной команды для переключения задач нет (Юрий Николаевич винит в экономности разработчиков, так как мы её хотим и все обосновали логически). Переход по команде Jmp к селектору TSS вызывает переключение задач.

Лекция №15

06.11.2010

Межкольцевые вызовы задач

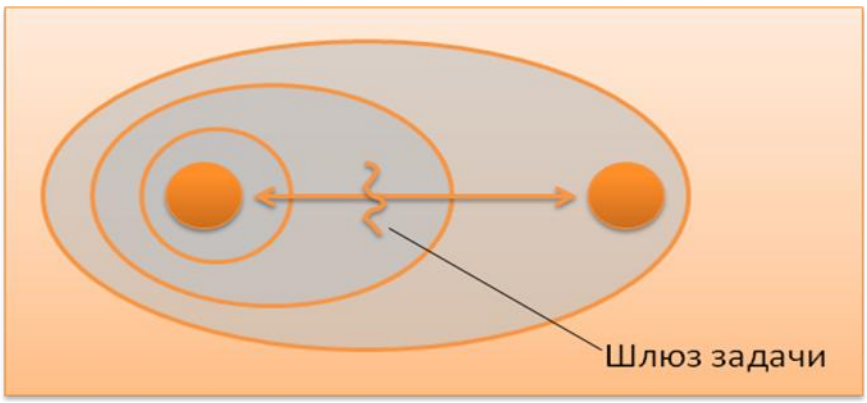


Рис. 15.1

Используется функция jmp <селектор шлюза TSS>

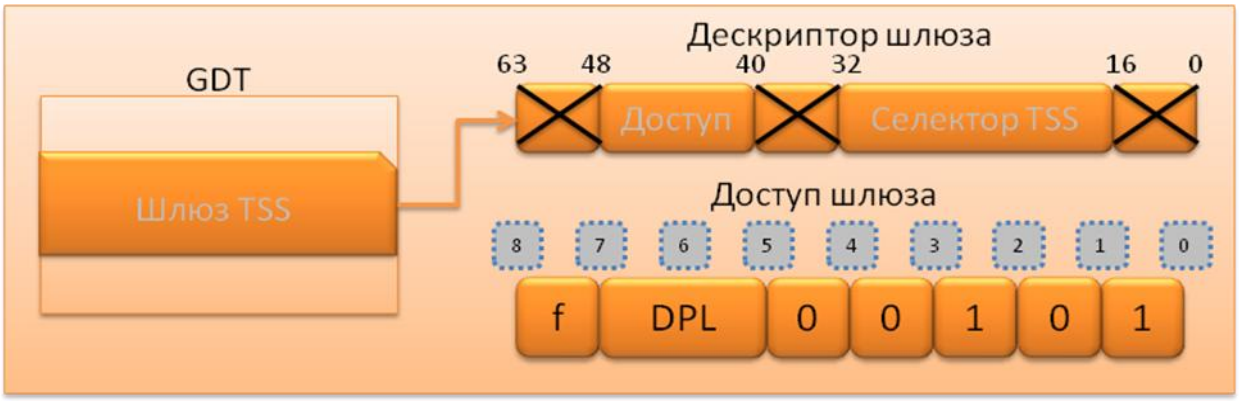


Рис. 15.2

Вложение задач

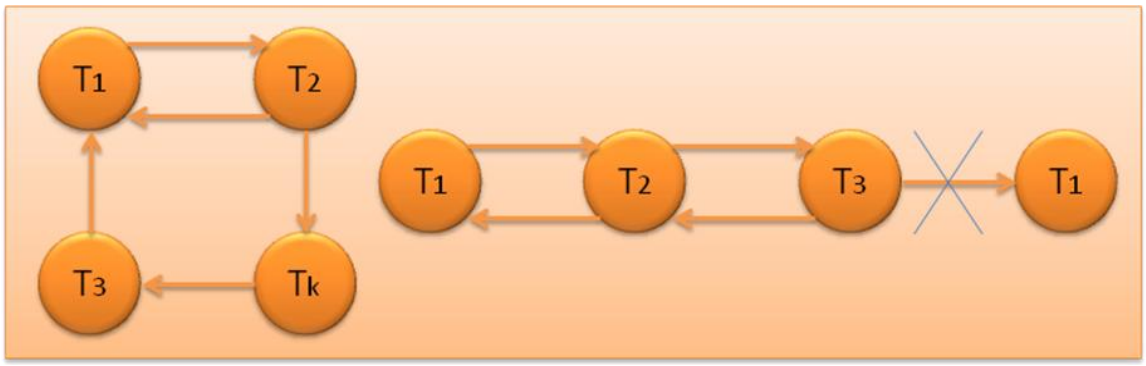


Рис. 15.3

Активация вложенных задач

CALL <селектор TSS>

CALL <селектор шлюза TSS>

Байт доступа дескриптора TSS

При вложенных вызовах задач бит занятости B остается установленным во всех задачах цепочки (если задача уже занята, то обратиться к ней невозможно). Для возврата из вложенной задачи используется команда IRET → переход к задаче путем обратной связи.

Рис. 15.5

IOPL (Input/Output Privilege Level) – уровень привилегий ввода/вывода. Номер кольца, в котором разрешено выполнение команд ввода/вывода (in, out).

NT (Nested Task) – признак вложенной задачи. Устанавливается в единицу, при активации задачи команды CALL. Предполагается, что если команда CALL устанавливается, то команда jmp сбрасывается.

Введение в теорию операционных систем

Классификация ресурсов

1. Реальные ресурсы
2. Виртуальные ресурсы
3. Воспроизводимые
4. Невоспроизводимые
5. Критические
6. Некритический
7. Разделяющиеся
8. Не разделяющиеся

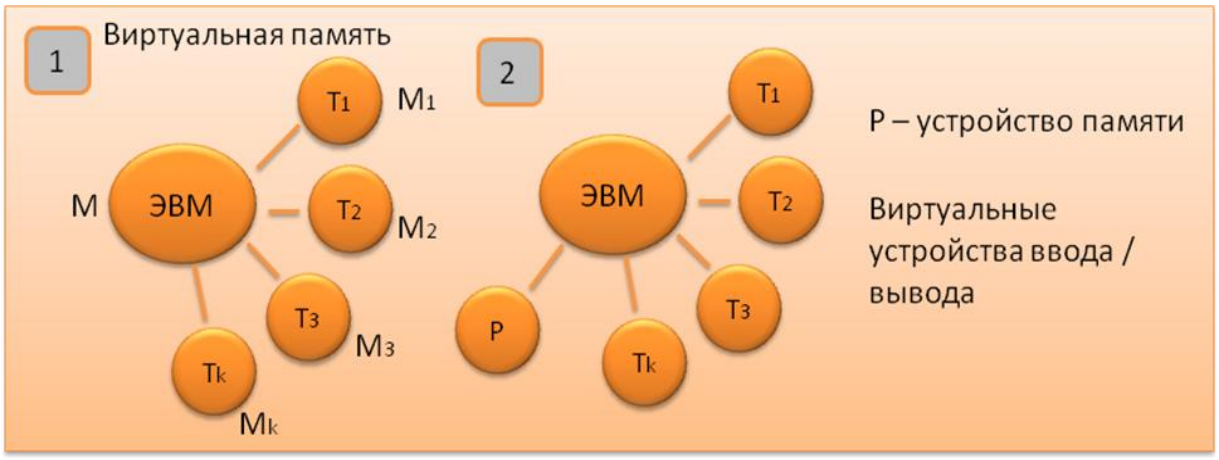


Рис. 15.6

ОС – набор программ, вплотную взаимодействующих с аппаратурой и решающих задачи:

1) Виртуализация ресурсов
2) Управление реальными и виртуальными ресурсами
3) Организация интерфейса с пользователем

Лекция №16

13.11.2010

Операционные системы

Операционные системы бывают (на классификацию забили (точнее она по исторической логике)):

1) Однозадачные (не существует проблемы конфликтов с решением задач) мы рассмотрим её как частный случай в конце
2) Многозадачная
    - Система с разделением времени
    - Система реального времени
    - Событийно – управляемая система
3) Распределённые

Системы разделения времени

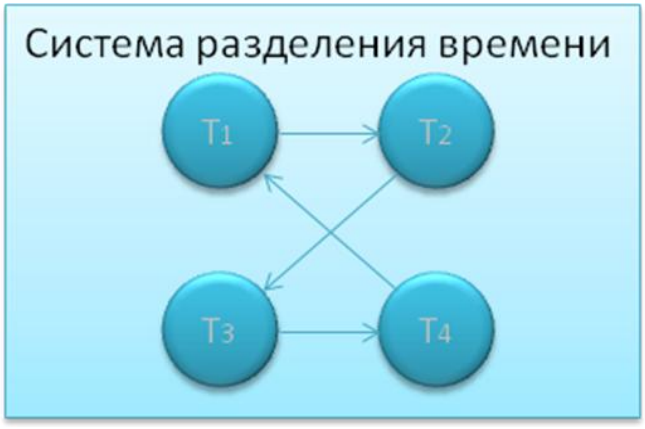


Рис. 16.1

Такие системы самые распространенные на данный момент.

1.Задаче Tk выделен квант времени τ.

2. По истечении кванта τ происходит переключение на задачу Tm.

В системе разделения времени нет возможности сказать системе как много ресурсов и какое время потребуется для выполнения задачи – то есть она работает в «неопределенных» условиях. Происходит «спонтанный» подход к использованию ресурсов. И так идет развитие дальше с развитием гибкости интерфейса.

Рис. 16.2

Разделение времени + Приоритет задачи – комбинация вот этого ведет к тому, что мы не совершаем круговой обход задач, а идем по более сложной системе.

В современных системах (того же класса Windows) понятие приоритета реализовано достаточно сложно. Приоритет сделан в виде номера – чем меньше – тем важнее. Раньше они шли просто по линейной цепочке. Далее были реализованные классы, внутри которых градуирование.

Рассмотри OS/2 – один из первых проекторов Microsoft. Здесь впервые был применён «забавный» принцип динамического управления приоритетов. Если проходит время максимального ожидания – приоритет повышается искусственно, задача выполняется и далее приоритет опускается.

Системы реального времени

Рис. 16.3

Есть задачи, которые нужно выполнять немедленно. С системой разделения времени это не работает, так как текущая задача отработает свой квант времени (>миллисекунды), а нужно реагировать быстрее (<миллисекунды).

Pk - приоритет k-той задачи.

Рассмотрим пример:

Пусть: P1 - текущая задача. Работает без ограничений квантов времени до тех пор, пока не будет готова задача P2. P2 - готовая к выполнению задача.

1. Если P1 > P2, то P1 продолжает выполняться. P2 в режиме ожидания.

2. Если P1 < P2, то происходит мгновенное переключение на P2.

Фактически, программы могут занять всё процессорное время. Реально, предполагается, что программы не жадные. Они отрабатывают некоторое небольшое время и приостанавливаются до востребования.

Из системы реального времени можно сделать систему разделения времени. (НО наоборот – нельзя)

Моделирование режима разделения времени

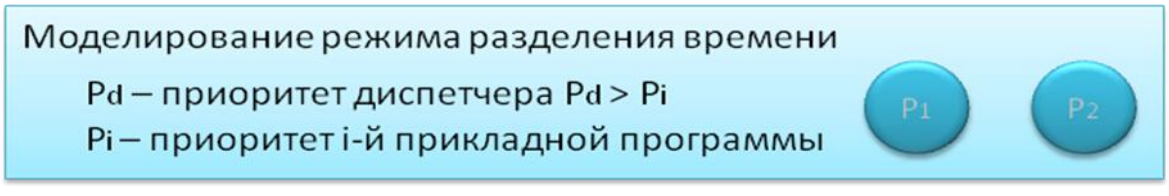


Рис.16.4

Pd - приоритет диспетчера, Pi - приоритеты i-той прикладной задачи. Pd > Pi.

1. Pd - монопольно владеет процессором в состоянии готовности.

2. Диспетчер принимает решение об активации задачи Pk.

3. Диспетчер повышает приоритет Pd > Pk > Pi. Диспетчер себя приостанавливает на время t.

Пример. iRMX86 в каком-то смысле основополагающая для современных ОС реального времени.

Событийно-управляемые системы

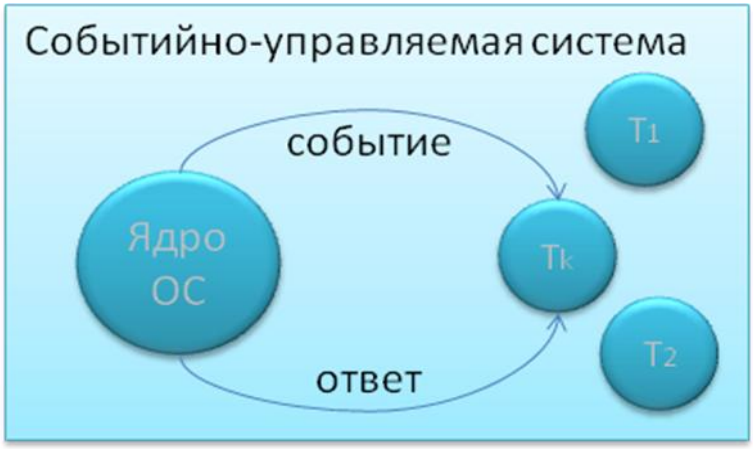


Рис. 16.5

Основные: Windows-подобные.

ОС. Задачи. В задачах возникают события (Нажата кнопка, Выставлен нужный регистр). Задача докладывает системе. От системы задаче присылается ответ.

Лекция №17

Событийно-управляемые системы

Задача информирует систему о наступлении некоторого события. ОС управляет задачей в соответствии с наступившим событием.

Идея возникла не просто с диалоговыми системами, а с развитыми интерактивными графическими оболочками. В системах Windows 3-11.

Однозадачная ОС

Примеры – CP/M; MS DOS. (Может быть, они уже не столь популярны в потребительской среде, но в качественном уровне они могут вполне жить и более чем актуальны для специальных устройств)

Возникает вопрос: Как оказывает услуги ОС задачам?

Программа завершила своё действие – это означает, что она прекратила своё существование и освободила все свои ресурсы.

Если нам нужно управлять прямым отменном (Пр1) – мы не будем привлекать ядро ОС. ЗА обмены будут отвечать системные процедуры, которые будут вынесены на эти процессоры. ОС как бы размазана по системе, но при этом чаще всего имеет управляющий узел.

Компоненты ОС разбросаны по множеству вычислительных узлов.

Кластерные системы

Наш управляющий компьютер каким-то образом принимает решение о запуске, обработке программ. Сам он решением не занимается, а отправляет их в вычислительную сеть. В качестве вычислительной сети нам не важно что выступает. Вычислители в кластерной системе совершенно автономно, в отличие от других систем.

Программы. Диаграммы состояния процессов.

Процесс – это задача, вместе с её текущим системным окружением.

Базовое состояние процессора:

Деятельность любой программы зависит от двух параметров – ресурсов и доступа памяти, и квантов времени процессора. В связи с этим существуют следующие состояния:

Готов: процесс имеет все необходимые для его работы ресурсы, кроме процессорного времени.

Выполнения: состояние Готов + процессорное время.

Блокируется: у процессора нет каких-то ресурсов, необходимых для его работы (выделять ему какое-то время бессмысленно, оно ему уже не поможет =)).

Теперь нарисуем диаграммы.

I Диаграмма состояния в самом простейшем виде. Но существует небольшая проблемка – есть два состояния процесса, когда задача не решается и только одно активное. Задача не может сама управлять своей активностью и пассивностью. Пример программы управления активностью и пассивностью.

Лекция №18

II Диаграмма. Вводим ещё два состояния – «Приостановлен - готов», и состояние «Приостановлен - блокирование». Связанны они с деактивацией процесса. Многие программные пакеты содержат следующие программы – если одна программа не успевает выполнять задачу, совместно с партнером – она может попросить приостановить её (её сбор данных и пр.)

За выполнение этих состояний отвечает в Linux (Windows) – Sleep, Nanosleep и пр.

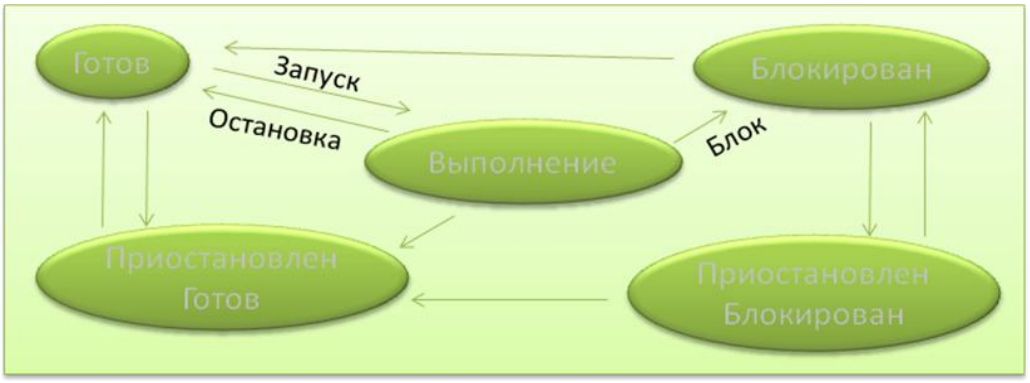


Рис 18.1

III Диаграмма. Все предыдущие состояния могут относится как к режиму ядра так и к режиму пользователя.

Рис 18.2

Родительский процесс

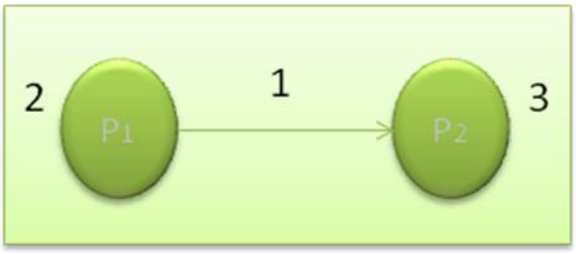


Рис 18.3

(Одна программа может порождать другую. Между ними может быть как односторонняя связь, так и двусторонняя. К примеру, P1 запустила P2 сказала ей – «Пользуйся моей памятью». (1)Запуск процесса и передача ему некоторых ресурсов. (2)После этого, через некоторое время P1 решила, что он всё делала и заканчивает работу – завершение родительского процесса. (3) Дочерний процесс, использующий ресурсы родительского процесса продолжает работу.)

<u>Параллельное выполнение программы</u>

Многопоточные системы

Процесс (классика)

1) Обладатель ресурсов
2) Объект квантования процессорного времени

Процесс (многопоточность)

1) Обладатель ресурсов
2) Имеется особый ресурс: поток
3) Поток является объектом квантования процессорного времени

(Особое внимание обратить на ресурс – поток. Он может иметь его не в одном экземпляре – см. 18.4)

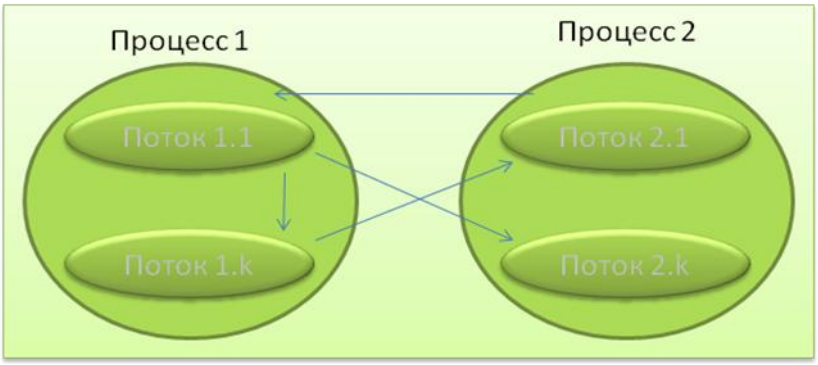


Рис. 18.4

Приведем пример:

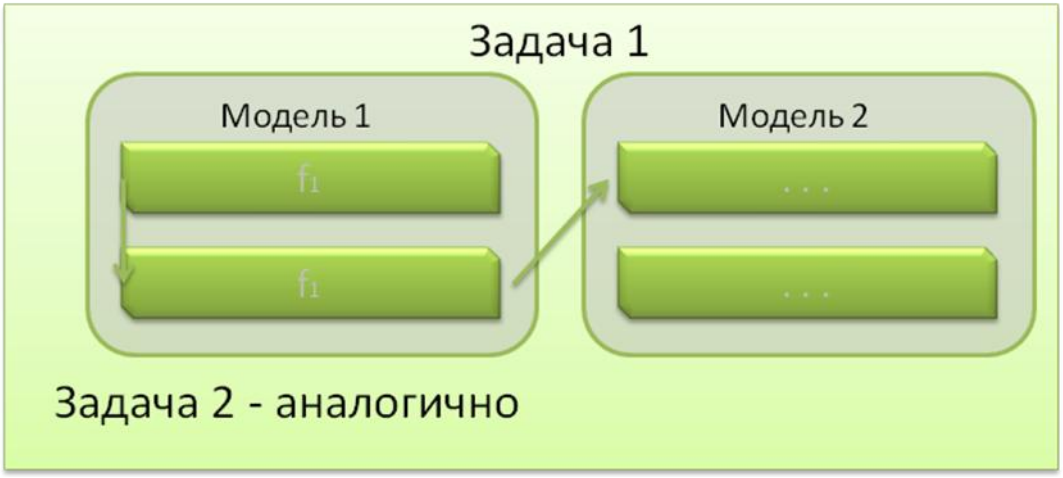


Рис 18.7 – пример. Это хорошо знакомая нам структура Си-программы. Стрелки теперь – это переключение процессорного времени (а не состояний, как ранее)

Классика:

1)Происходит переключение задач.

2)Внутри задачи реализуется последовательный вызов процедур.

Многопоточность:

1)Различные процедуры внутри одной задачи могут быть заданы в различных потоках.

2)Происходит переключение потоков

Параллелизм может состоять в одной задаче. Ныне он есть практически во всех системах и программах. Для задания чисто вычислительной природы – это не слишком характерно, но для интернет - проектов, серверно - клиентских систем – когда существует поток клиентов – для обслуживания различных клиентов мы создаем различные образы нашей сущности – программы, активной задачи. Можем обслуживать отдельным поток внутри одной задачи, можем запустить несколько модулей, задач и пр. И квантовать в отдельных кодах, ветвях.

Иерархия подсистем. Понятие системного вызова.

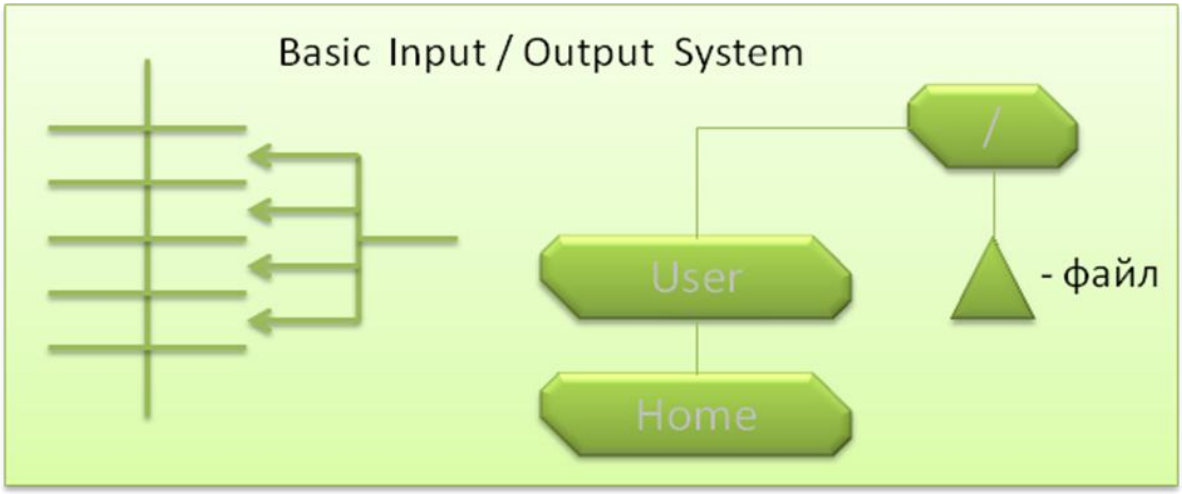


Рис.18.8 Для нас все компоненты ОС представлены в виде некоторых системных вызовов.

Ядро

- управление критическими ресурсами (память и процессорное время)

Базовая подсистема ввода/вывода(BIOS)

- обычные операции нужного уровня.

Рис.18.9 (Сектор – минимальная единица информации)

Расширенные подсистемы ввода/вывода (EIOS – External Input Output System)

- обменные операции верхнего уровня + логическая модель организации данных (Она скрыла сектора, дисковые накопители и пр. – например файловые системы)

Те библиотеки, которые мы используем в С правильнее было бы назвать прикладным программным интерфейсом.

Прикладной программный интерфейс

-это набор функций и/или библиотек, реализующих универсальный (в частности упрощенный) доступ к системным вызовам.

Основные задачи системного программирования

1.Задача управления памятью – хотим выделять память из общего пула для каких-то k задач.

2.Проблема критической секции – в асинхронном режиме получать доступ к разделённой области память (ранее назвалось критической памятью), т.е. нужно синхронизировать.

3.Проблема тупика – ситуация – есть некоторое кол-во процессов, они вступили в конфликт, в борьбу за ресурсы.

Задача управления памятью

Введем терминологию:

1.Управление реальной (физической) памятью (когда оперируют адресами).

2.Управление виртуальной памятью (оперирование виртуализованными адресами)

Лекция №19

27.11.2010

Задача управления памятью

1. Управление реальной памятью
2. Управление виртуальной памятью

I Управление реальной памятью

1.1. Связанные распределением памяти разделение фиксированным размера

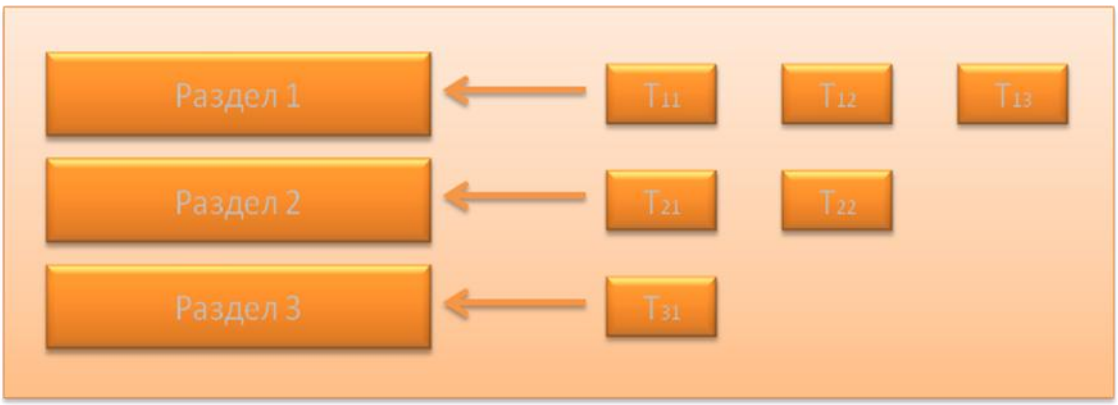


Рис 19.1

1.Память разбита на ресурсы различного, но фиксированного размера.

2.В каждом разделе выполняется некоторая множество задач

Рассмотрим плюсы и минусы

+	-
❖ Простота	❖ Возможный простой задач даже при наличии свободной памяти ❖ Возможна существенная потеря памяти

1.2.Связное распределения памяти разделение памяти разделами переменного размера

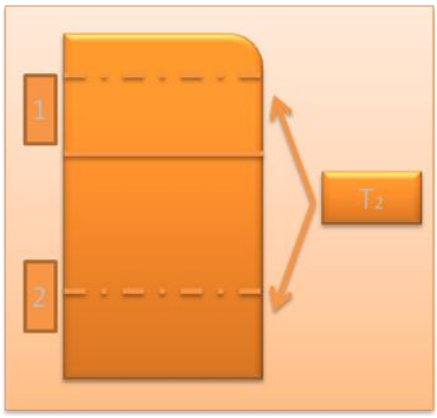


Рис 19.2

1.Память выделяется связным разделом по запросу задачи.

2.Размер раздела определяется потребностями задачи.

+	-
❖ Гибкость ❖ Уменьшение потерь памяти	❖ Существование проблемы управление памятью при завершении задачи, при

	повторном запуске задач

При этом возникает такая задачка: «Какой размер выбрать, если по размеру подходит несколько».

Возможны следующие стратегии:

1.Выбор случайного

2.Выбор наиболее подходящего по размеру.

3.Наименее подходящего по размеру – из стохастического анализа.

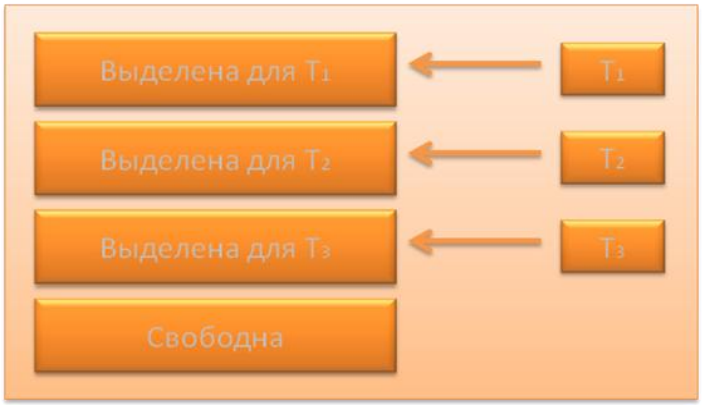


Рис 19.3

1.3.Несвязное распределение памяти разделами фиксированного или переменного размера

Переменное разделение сводится так или иначе к фиксированному.

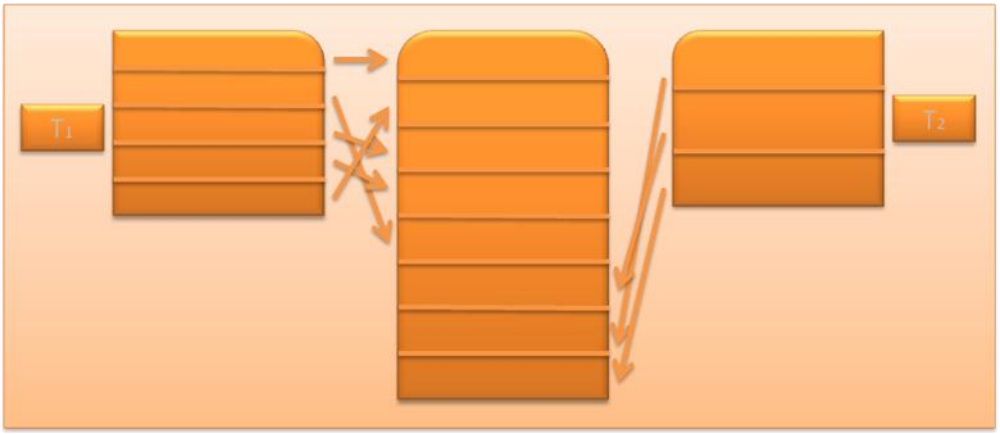


Рис 19.4

1.Задаче не требуется распределение памяти связным образом.

2.Память по запросу задачи выделяется с помощью несвязных разделов.

+	-
❖ Гибкость и эффективность	❖ Сложность управления

1.4 Система со свопингом

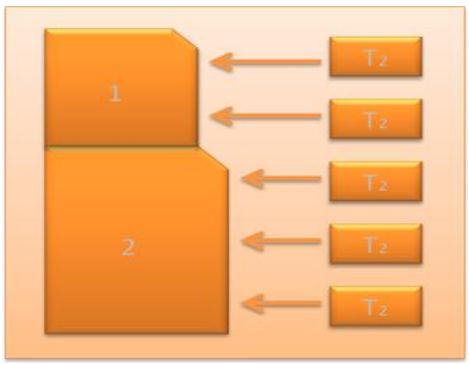


Рис. 19.5

1. Отдельные разделы памяти разделяются несколькими задачами.
2. Данные и код неактивной задачи выгружаются во вторичную память.
3. Задача или ее фрагменты загружаются в оперативную память при обращении к ней.

1.5 Оверлейные структуры.

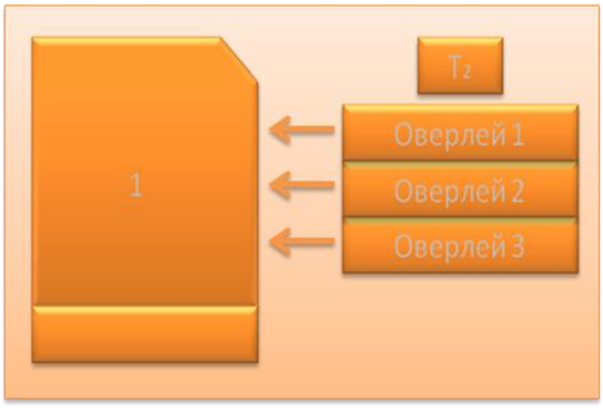


Рис 19.6

1. Размер доступной оперативной памяти меньше, чем необходимо задаче.
2. Код и данные задачи разбиты на множество логических фрагментов (оверлеи)
3. В каждый момент времени доступная память занята одним из оверлеев.

II Управление виртуальной памятью.

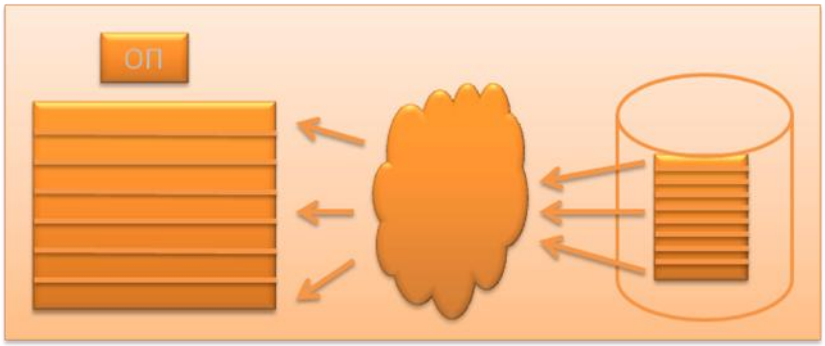


Рис 19.7

1. Память задачи представлена множеством логических блоков (обычно сохраненные во вторичной памяти)
2. Какая-то память разбита на множество блоков.
3. Блоки памяти задачи перемещаются между физической и вторичной

памятью.

Проблемы и задачи:

1. как связать логические адреса с физическими.
2. Как выбирать разделы при замещении.

Управление виртуальной памятью

2.1 Сегментная виртуальная память (сегмент имеет переменный размер)

Из минусов: сложность управления.

2.2 Страничная виртуальная память (ориентированный размер)

Из плюсов: Простота и эффективность.

Из минусов: нет адаптации под структуру задачи.

Лекция №20

1.12.2010

Модели виртуальной памяти

1.Сегментирование

+	-
❖ Хорошая адаптация под структуру задачи	❖ Высокая сложность

2.Страничные (фиксированный размер)

+	-
❖ Простая и достаточно эффективная	❖ Плохая адаптация под модель данных задачи

Рис 19.7 ->преобразование адреса

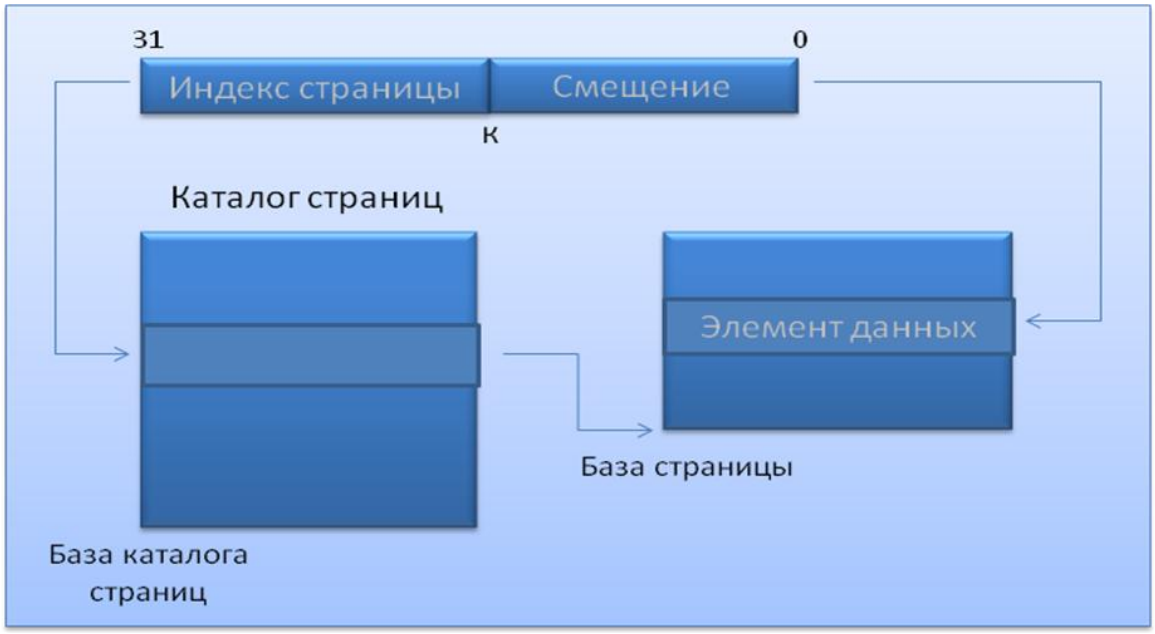


Рис 20.1

Механизм замещения страниц

1.выбор случайного.

2.Выбор редко используется

3.Выбор страниц, к которым дольше всего не было обращения

4.Выбор страниц, к которой не было обращения на отрезке времени τ.

(1) и (4) – наиболее часто используемые

3.Сегментно-страничная модель x86

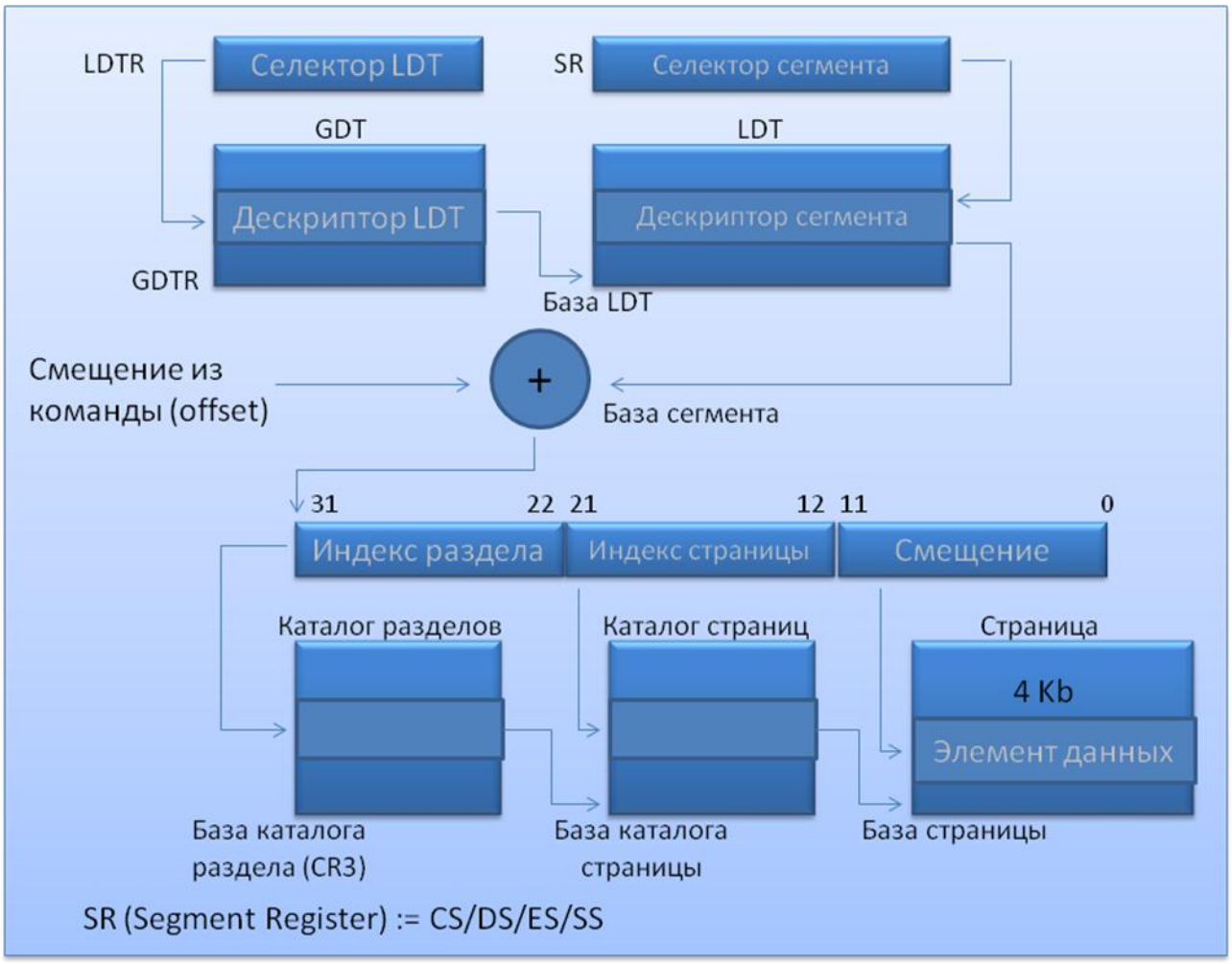


Рис 20.2

Все каталоги хранятся в ОП

TLB(Translator Lookside Buffer) – кеш активные страницы – способ повышения скорости работы со страницами.

Проблема критической секции

Рис 20.3

Критическая секция – секция кода, обращаясь к разделу области

Процесс выполняются асинхронно

P1 P2

X=X+1 X=X+1

Reg1<-X Произошло переключение Reg2<-X

P1->P2

Reg1<-Reg1+1 Reg2<-Reg2+1

X<-Reg1 X<-Reg2

Современный механизм сводится к «семафору».

Семафор S – примитив, используемый для синхронизации процессов: S∈[0,p], p>0

Если p=1, то двоичный семафор

Если p>1, то семафор – счётчик

P(S) |операции над

V(S) | семафорами

P(S): Если S>0, то S<-S<-1 и войти в критическую секцию, иначе остаться на семафоре

V(S): S<-S+1

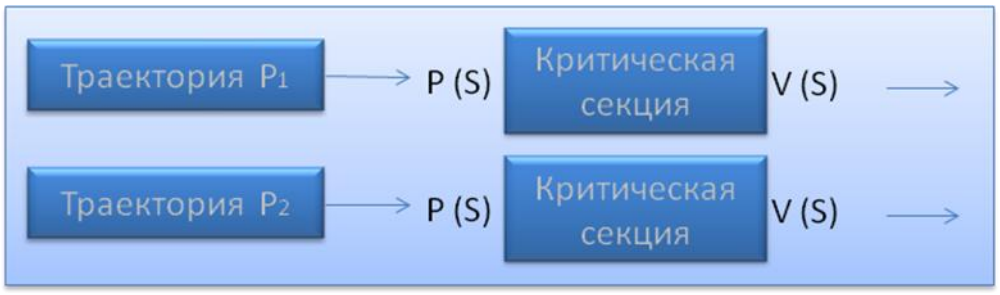
Операции над семафорами являются неделимыми.

Reg<-S

S<-S+1

S<-Reg

Все операции происходят за один прием, иначе если это будет не так, то проблема приедет вместе с прошлым.



Далее понятно – должны использовать примитивы перед каждой критической секции.

Рис 20.4

(P1, P2 – >P(S)далее квадратик –> далее V(S))

Проблема тупика

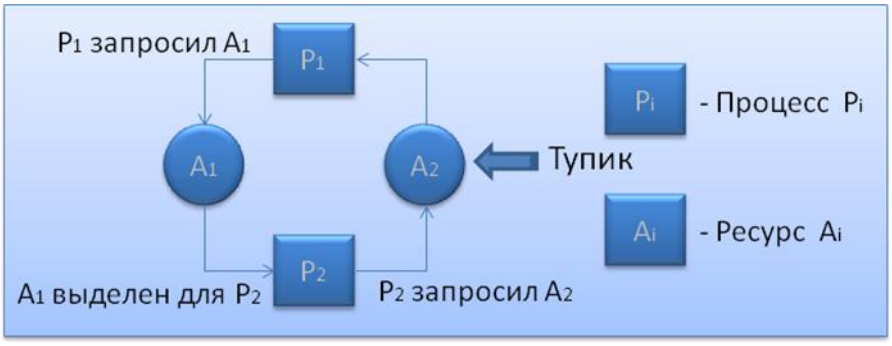


Рис 20.5 (Классический тупик)

A2 – выделен для P1.

P1 – запрашывает A1.

Процессы Pi – выполняются асинхронно.

Сформулируем причины – т.е. «необходимые условия тупика»:

1. Процесс монопольно владеет выделенными ресурсами.
2. Ресурс нельзя отобрать у процесса до полного их использования.
3. Процесс удерживает выделенные ранее ему ресурсы, ожидая выделения новых.
4. Существует кольцевая зависимость процессов.

Пути решения проблемы тупика

1. Построение безтупиковой систему – исключения тупиков.
2. Обход тупиков – балансирование на грани.
3. Обнаружение тупиков
4. Восстановление после тупиков

Исключение тупиков – сотри любое из 4-х условий – но ведет к резкому снижению производительности системы

Обход тупиков – полностью проблему не лечит – попытка не попасть в тупик, когда такая опасность существует. -> алгоритм «банкира»

Рис 20.6

∑ = 13, Роздано = 11, Свободно =2

Состояние называется безопасным, если существует возможность завершение всех процессов. -> не надо допускать перехода в опасное состояние

Лекция №21

4.12.2010

Из прошлой лекции:

1. Исключение тупиков
2. Обход тупиков
3. Обнаружение тупиков
4. Восстановление после тупика

Обнаружение + восстановление

- ❖ Установить факт тупика
- ❖ Выявление процессов и ресурсов, вовлекаемых в тупиковую ситуацию
- ❖ «мягкое восстановление»

<u>Модели данных языков программирования</u>

- ❖ управление памятью (как конкретная модель это делает – может быть связная, невсвязная и пр.)
- ❖ поддерживаемые методы структуризации (насколько сложные агрегаты может и поддерживать модель)

Видимость объекта (все что есть в программе – переменные, функции и пр.) программы

- ❖ где можно использовать
  - везде – такой объект называется объектом глобальной видимости
  - не везде – такой объект называется объектом локальной видимости

Время существования объекты программы

- ❖ как долго находится в оперативной памяти
  - всегда – глобальное время существования
  - временно – локальное время существования

Мы пройдем три языковых модели (Алгол и Фортран, Си и организация управления классами памяти)

Модульная модель языков программирования (Фортран (США), ~1960)

Рис. 21.1

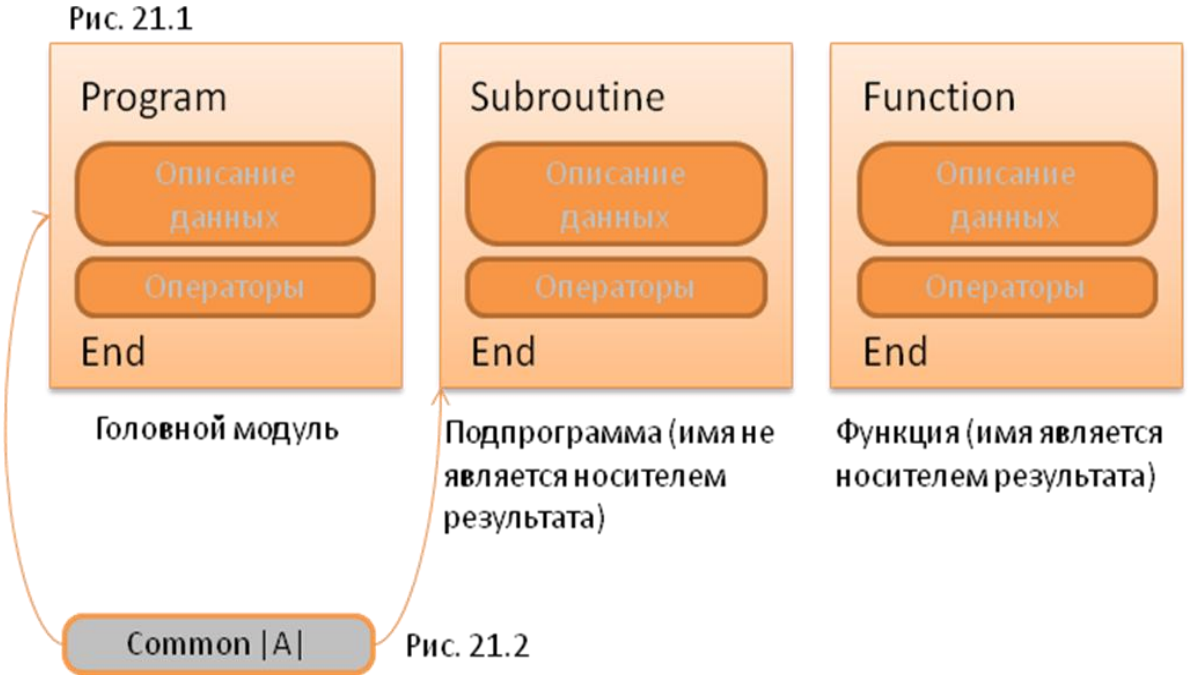


Рис. 21.2

Рис 21.1 (Обратите внимание на подпрограмму и функцию – ранее мы их не различали. Именно функция является носителем результата)

Модульная модель:

- Каждая подпрограмма (или функция) является самостоятельным модулем.
- Определенные внутри модуля переменные локализованы в нём (не глобализованы)
- Память по объекты (переменные) выделяется статически
- Наличие областей памяти, разделяемых различными подпрограммами (common - блок) – имитация глобальной среды (Рис. 21.2)

Принцип модульности ляжет практически во все языки программирование. Нам хочется, чтобы модуль проявлялся как надклассовый объект. Сейчас понятие функции и модуля тождественно.

Обрисуем полную альтернативу

И Блочная модель языков программирования (Алгол (Европа), ~1960)

Центр разработки находился в Дубне. Сильно использовался в СССР. Теоретическая база – Алгол, но более практическая – Фортран. Си, Паскаль – «обглоданный» Алгол.

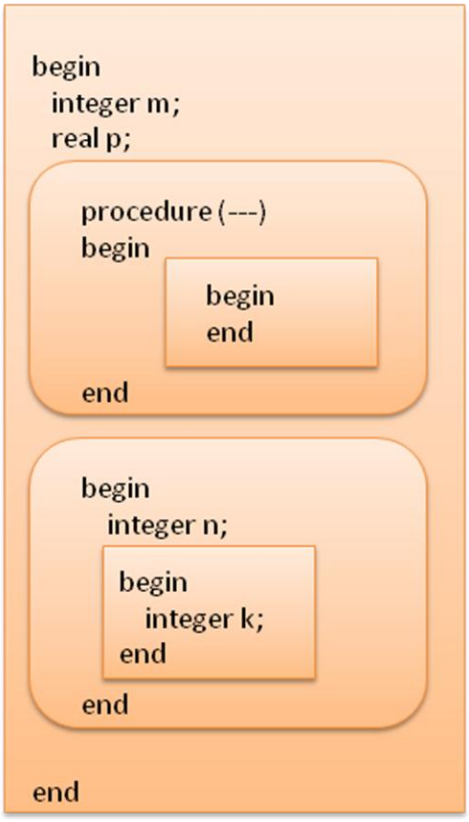


Рис 21.3

Дикая по сложности иерархия. Можно определять функцию в функции. Блоки имеют неограниченную глубину вложению. Вся программа представляет собой единый моноблок.

- Существует единственный программный «модуль»
- Вся программа представляет собой систему вложенных блоков
- Процедуры (как и ранее обобщение функции и подпрограммы) и функции вкладываются в блоки, их тела являются полноценными блоками
- Данные локализованы в блоках
- Внутренний блок наследует данные объемлющих его блоков (Из рис. 21.3. (1) – объемлющий для (2) и (3))
- Память под переменные выделяется при входе в блок и удаляется при выходе из него (т.е. полная динамика – статическая модель отсутствует)

На Алголе-ГДР написаны практически все ядерные исследования и исследования в квантовой химии. На Си, и более современных постАлгольных языках такие задачи реализованы не были.

Серьёзных методов агрегирования на Алголе и Фортране не были. Массивы были и только они, более сложных структур не было.

Лекция № 22

11.12.2010

Блочно - модульная архитектура (Си)

Сюда приходит блочная структура, но сильно лимитированная. Блоки есть, но нельзя создавать операцию одну в другой.

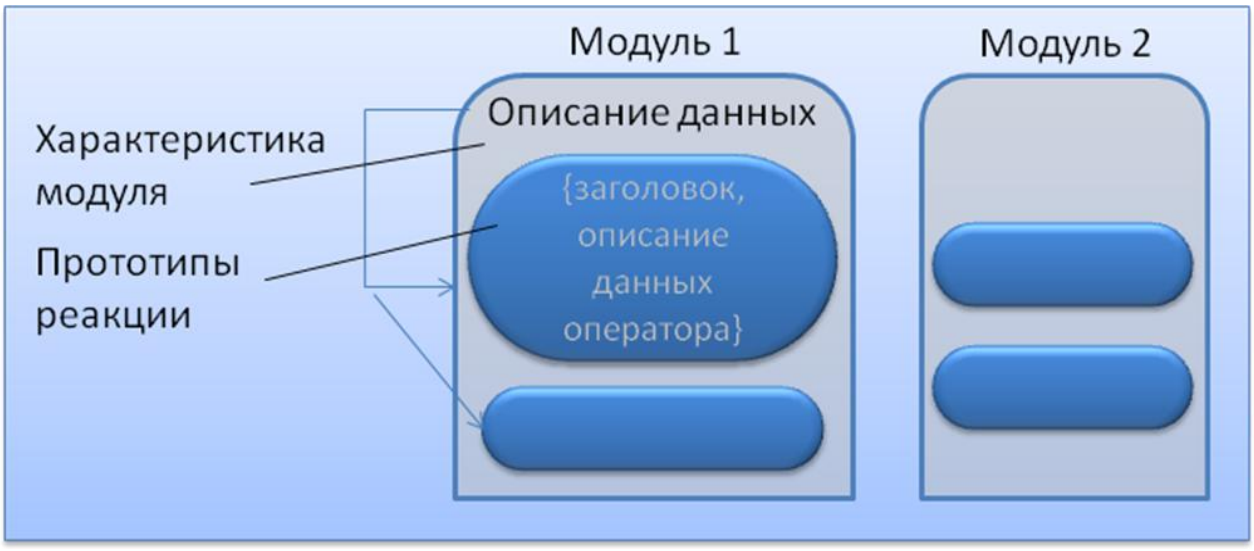


Рис.22.1. – Все активные компоненты вкладываются внутрь модуля. Модулей уже не один – Ура, Фортран задействован, ему повезло. Функцию в функции мы создать уже не можем, но блоки у нас уже есть. Слева мы можем видеть саму характеристику модуля.

1.Размещается на статической памяти.

2.Глобальнен по времени и пространству (действия в пределах модуля и возможность (с разрешения) экспорта в соседний модуль)

3.Инициализирована нулем.

Прототипы реализации:

1.Размещаются на динамической (стековой) памяти.

2.Локальны по времени и пространству.

3. Имеем неопределенное значение в момент создания. (Т.е. грязь в памяти с прошлого раза пользования)

Классы памяти.

Для управления памяти используется 4 ключевых слова:

auto – автоматическая переменная. (автоматическая активация при входе в блок – на самом деле оно холостое)

Пример:

```
 int main (void)

{
.....
auto int m; //на самом деле ничего не изменилось – т.е. исторически оставшаяся вещь
.....
```

}
Register – регистрационная переменная (динамическая, памяти не имеет, рекомендуится располагать в регистре)

register int n; //пожелание транслятору располагать не в памяти, а на регистре. В современных

условиях не используется. Чаще выносятся счетчики, и то не факт.

…..

}

Два действительно полезных слова:

static – статическая переменная

extern – внешняя переменная

Static:

1.На уровне функции –

Размещается в статической памяти.

Глобально по времени, но локально внутри функции.

Инициализирована нулем.

Пример:

int func (void)

{

static int m;

…….

}

2.На уровне модуля.

Ограничивает видимость объекта одним модулем. Т.е. ограничение на экспорт.

Рис. 22.2. Обозначает разные вещи в различных модулях.

Определение переменной

1.Выделение памяти.

2. Возможность использования.

Описание переменной.

1.Возможность использования.

Т.е. у нас есть оттеночные отличия определить и описать переменную. Таким образом мы в каком-то модуле определяем переменную, а в других модулях, импортирующих, мы описываем.

Возможность, право на использование и есть extern. Т.е. описание. Память, кстати, не выделяется.

Напоминание: модели языков программирования:

1.Способ хранения.

2.Способы структуризации.

1.Порождение новых типов.
2.использование аппарата указателей.
3.использование структур (struct)
4.использование объединений (union)

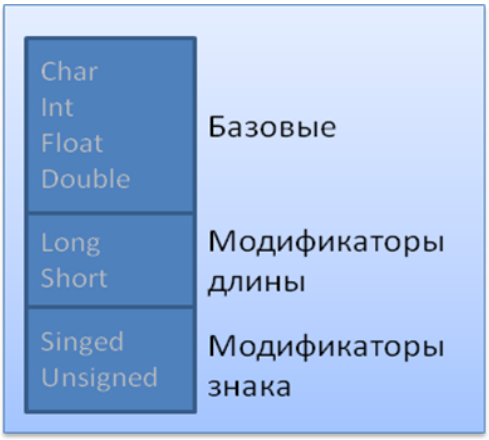


Рис.22.3

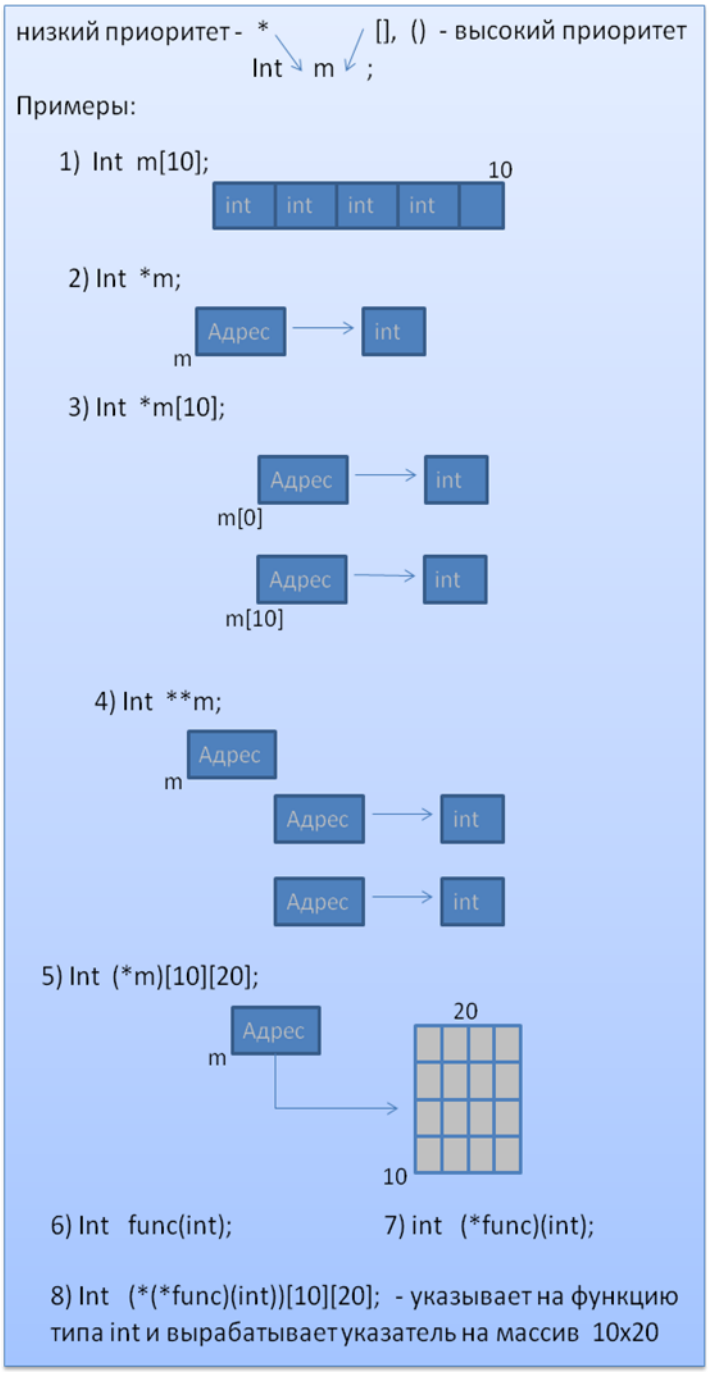


Рис.22.4.

У нас так же есть возможность породить НОВЫЙ кирпичик. (естественно мы не откажемся)

Итак, typedef

Пример:

```
typedef struct {
  char Autor [100];
  cahr Title[256];
  int Year;
  int Pages;
} BOOK;
```

BOOK Catalog[100];

BOOK *CafPtr;

Struct – структура (агрегирование с последовательным размещением полей в памяти)

Union – объединение (агрегирование с использованием общей памяти (наложение))

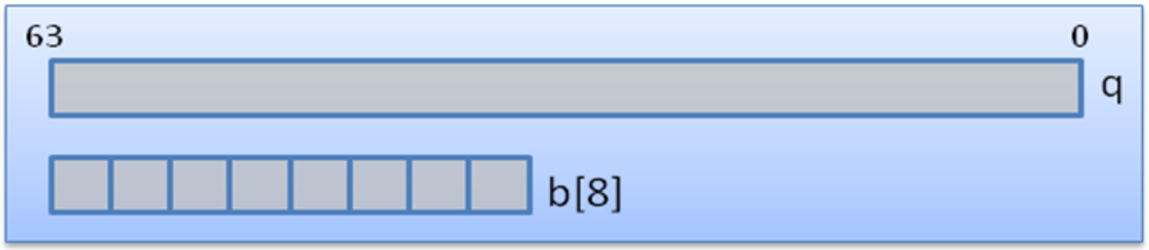


Рис. 22.5.

```
{ typedef union {
                        double q;
                        char b[9];
                       } VALUE;
VALUE V;      v.q. v.b[0]
}
```

Лекция № 23

15.12.2010

Структуры данных

I Классификация

1.Статические структуры (количество переменных не изменяется)

2.Динамические структуры (количество переменных изменяется)

(3.)Полустатические – «…эмуляция на тему динамических структур…»

II Классификация

1.Однородные структуры (много одинаковых элементов)

2.Неоднородные структуры (есть неодинакового типа элементы – к примеру, в таблицах с разными заголовками)

III Классификация

1.Оперативные структуры(скорость доступа велика, «..и все хорошо»)

2.Внешние структуры (скорость доступа мала (к примеру, на дисковом накопителе), перемещения замедлены, передача невозможна) – к примеру, файловые системы

IV Классификация

1.Непрерывные структуры (есть порядок следования - «есть первый, второй…. есть данный, есть следующий…», и есть физическое проявление)

2.Ссылочные структуры.

Будем много говорить о стэке.

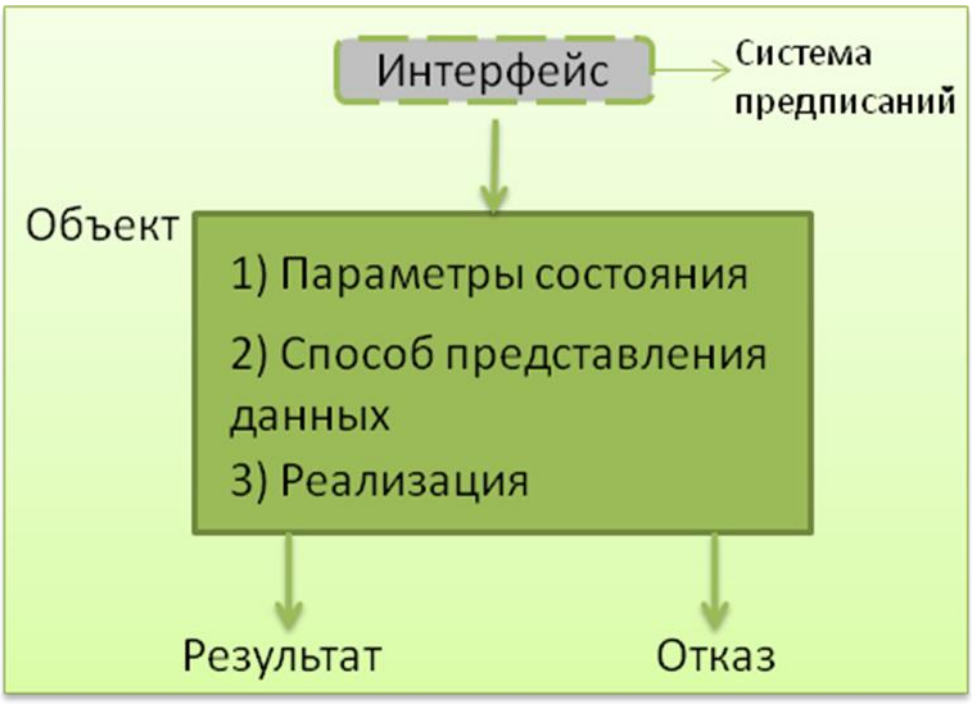


Рис.23.1. (Общий пример)

Набор предписаний – некоторый набор действий, которые могут быть выполнены. Если набор предписаний правильный – мы получим результат. Если он неверен или не может быть выполнен, тогда получим отказ.

Обычный пример «…из жизни…»:

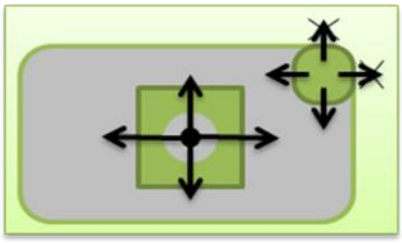


Рис.23.2. (Есть рисующая (пишущая) машина. Бегунок движется влево, вправо, вверх, вниз. А также поднять и опустить перо. Следовательно, имеем 6 предписаний. В углу же мы получим отказ, так как нельзя будет выполнить часть предписаний.)

Рис.23.3. (Между черчением и умением рисовать литеру лежит огромная дистанция. Проблема даже не в этом. Дело в том, что модуль, который обслуживает человека, не должен делать все. Поэтому мы дробим нашу программу. И мы начинаем реализовывать верхний уровень за счет процедур промежуточного. Проектирование ведется сверху вниз дроблением.)

Что принципиально дают объектно-ориентированные языки. Доступ к параметрам состояния должен иметь только этот объект. Также это может делать интерфейс. Больше никто. Если же мы берем обычный язык Си

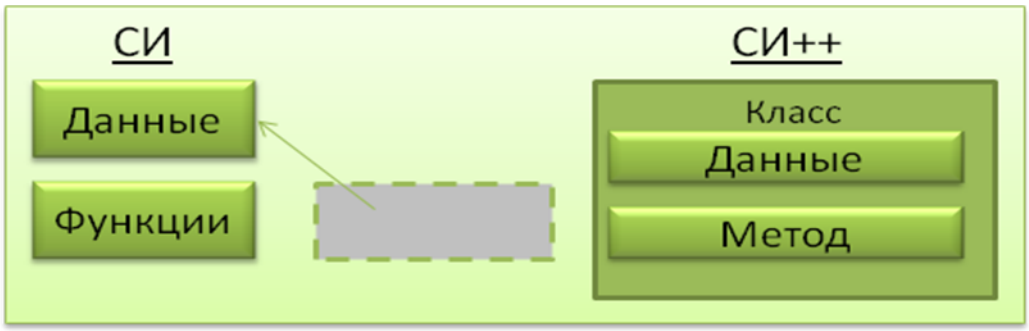


рис.23.4. Данные – то над чем, производится действие. Функции – активные элементы. Данные и функции живут самостоятельно. Если мы обратимся извне, то мы можем поломать структуру.) В объектно-ориентированных структурах, типа Си++ (рис.23.4.) у нас все сохраняется. Так же появляется такая вещь, как класс. Методы – те же самые функции. Классы – те же самые структуры, только в них можно вкладывать функции.

Структуры непрерывного хранения

1. Последовательность (Какие-то элементы приходят один за другим. Мы не знаем сколько их придет). Т.о. приводит к понятию однопроходного алгоритма. (Этим мы занимались к начале I семестра)
   Предназначение:
   а) Инициализация
   б) Обработать очередной элемент
   в) Пропустить очередной элемент
   г) Конец последовательности? (да/нет)
   д) Закрытие

2. Стек (Принцип LIFO – Last Input First Out)

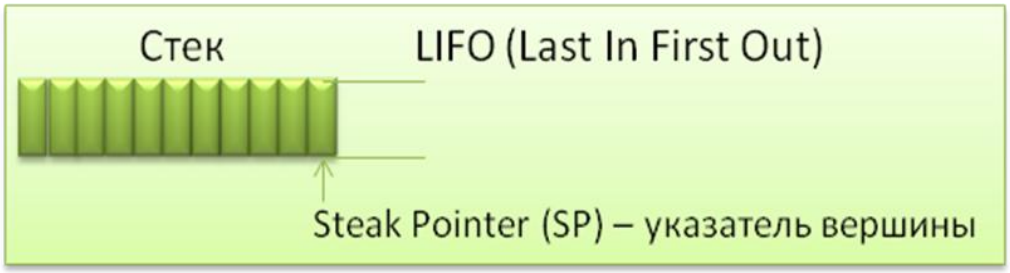


рис.23.5.

а) Инициализация
б) Добавить элемент на вершину
в) Взять элемент с вершины
г) Стек пуст? (да/нет)
д) Удалить вершину
е) Закрыть

Пример:

```
Stack h
typedef Struct {
double *Buffer;
int Depth;
int sp;
} STACK;
```

Процедуры:
STACK *StackInit (int Debth); - создать стэк – знать указатель на него;
int StackPoint (STACK *sptr, double Value); – 0, если все хорошо, отриц. – ошибка;
int StackPop (STACK *Sptr, double *Value); – извлечение из стека;
int StackClose (STACK **Sptr); - завершение. Двойной указатель – чтобы могли изменять адрес.

3. Очередь (Принцип FIFO – First In First Out)

Рис.23.6. Head – голова показывает на первый заполненный слой, который будет обслуживаться. Tail – на первый пустой. (Для симметризации)

а) Инициализация
б) Добавить элемент в конец
в) Взять элемент из начала
г) Удалить элемент в начале очереди
д) Очередь пуста (да/нет)
е) Закрыть

Пример:

```
typedef Struct {
double *Buffer;
int Lenght;
int Head, Tail;
} QUEUE;
```

Ссылочные реализации
4.Список.

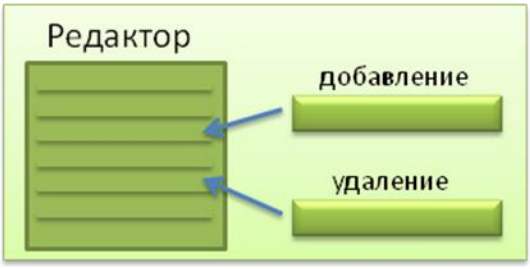


Рис.23.7

Рис.23.8.

Есть однонаправленные(L1) и двунаправленные(L2) списки. В нашем случае L2.
а) Инициализация.
б) Добавление элемент в конец списка.
в) Установить указатель в позицию k.
г) Добавить элемент до/после указателя.
д) Удалить элемент по указателю.
е) Закрыть (Удалить память).

Пример.

```
typedef Struct LIST {
char Str[256];
Struct LIST *Next;;
Struct LIST *Prev;
} LIST;
```

Рис. 23.9

Процедуры:
STACK *StackInit (int Debth); - создать стэк – знать укзатель на него;
int StackPoint (STACK *sptr, double Value); – 0, если все хорошо, отриц. – ошибка;
int StackPop (STACK *Sptr, double *Value); – извлечение из стэка;
int StackClose (STACK **Sptr); - завершение. Двойной указатель – чтобы могли изменять адрес.

Дерево у нас состоится из вершин и ребер. Причем продолжать можем и верх и вниз. Т.е. имея два дерева, мы можем объединить их в одно.

Лекция №24
18.12.2010

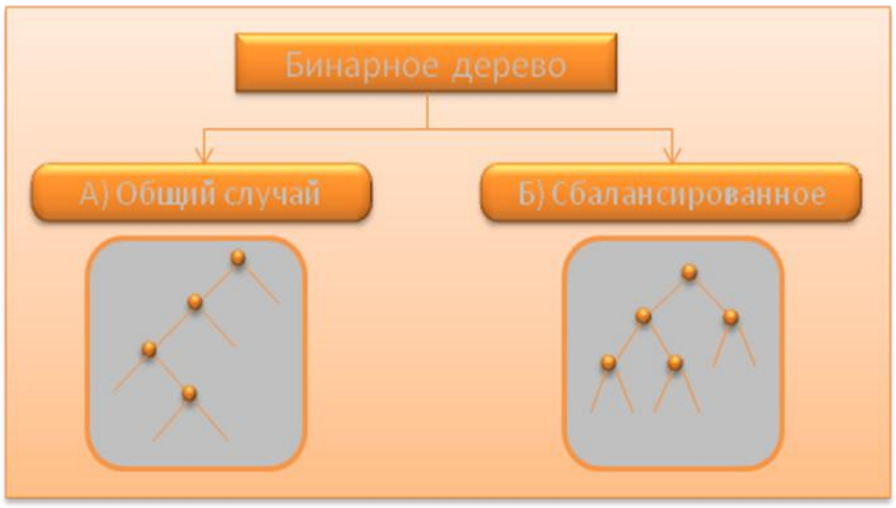
5.Деревья
Деревья выросли из списка, когда количество ссылок больше одной. Две соседние вершины соединяются ребрами. Расстояние между вершинами – все ребра между ними.

Рис.24.1.

Мы будем рассматривать определенные случаи, упорядоченных древ.
Пример.

```
typedef Struct TNODE {
double Value;
Struct TNODE *Prev;
Struct TNODE *Next[k];
} TNODE;
```

Чаще всего мы хотим иметь дело со случаями да/нет, больше/меньше. Т.е. не более двух ответов. Рассмотрим бинарное дерево.



Бинарное дерево (общий случай)
Рис.24.2.а.
Бинарное дерево (сбалансированное, т.е. разница двух вершин не более 1.)
Рис.24.2.б.

Пример:

```
typedef Struct TNODE {
double Value;
Struct TNODE *Prev;
Struct TNODE *Left, Right;
} TNODE;
```

Этот тип является и последним и не последним в каком-то смысле, в зависимости от среды программирования. Рекурсивность и пр. Если язык сам обладает рекурсивной природой, то обратный маршрут он будет запоминать автоматически. Если дерево имеет очень большую глубину, то рекурсивность может переполнять память и становится невыполнимой. Если мы используем рекурсивность древа и языка, то родитель нам не нужен (стираем Struct TNODE *Prev;).

Сейчас мы снова вернемся к общим деревьям и наложим новые условия.
Рассмотрим пример.

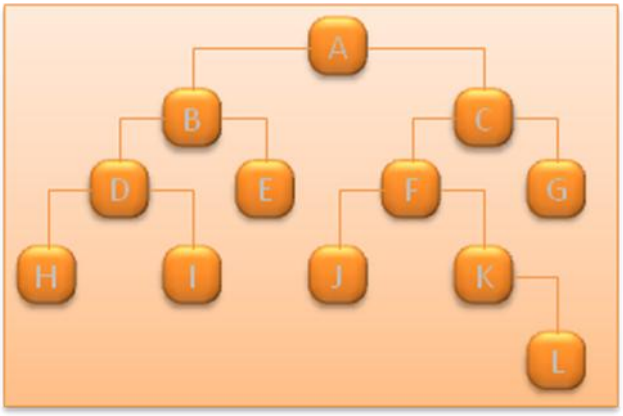


Рис.24.3.

Вообще этот рисунок можно посмотреть в книге [1]; Хотим обходить дерево с поиском максимального значения лежащего в вершина.
Возможные стратегии:
- сверху - вниз
- слева - направо
- снизу – вверх

```
1)   void UpDown (TNODE *Pos, double *Max)
{
 if(!Pos)
   return ;
if (*Max<Pos->Value)
   *Max=Pos->Value;
UpDown (Pos->Left, Max);
UpDown(Pos->Right, Max);
}
```

ABDHIECFJKLG

```
2)   void LeftRight (TNODE *Pos, double *Max)
{
 if(!Pos)
   return ;
LeftRight(Pos->Left, Max)
if (*Max<Pos->Value)
   *Max=Pos->Value;
LeftRight (Pos->Right, Max);
}
```

HDIBEAJFKLCG

Дерево поиска

Pos->Left->Value≤Pos->Value<Pos->Right->Value; (*)

Дерево поиска – бинарное дерево, для каждой вершины которого верно (*);

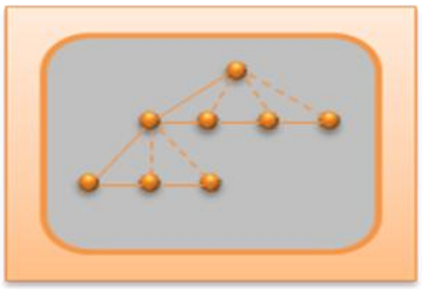


Рис.24.4.

В этом дереве мы явно видим усиление бинарного дерева.

typedef Struct TNODE {

double Value;

Struct TNODE *Down, *Next;

} TNODE;

Мы получили дерево эффективности как бинарное. К нему верны все бинарные алгоритмы, т.к. мы обрубили ветви (пунктиром помечены на рисунке).

6.Множества

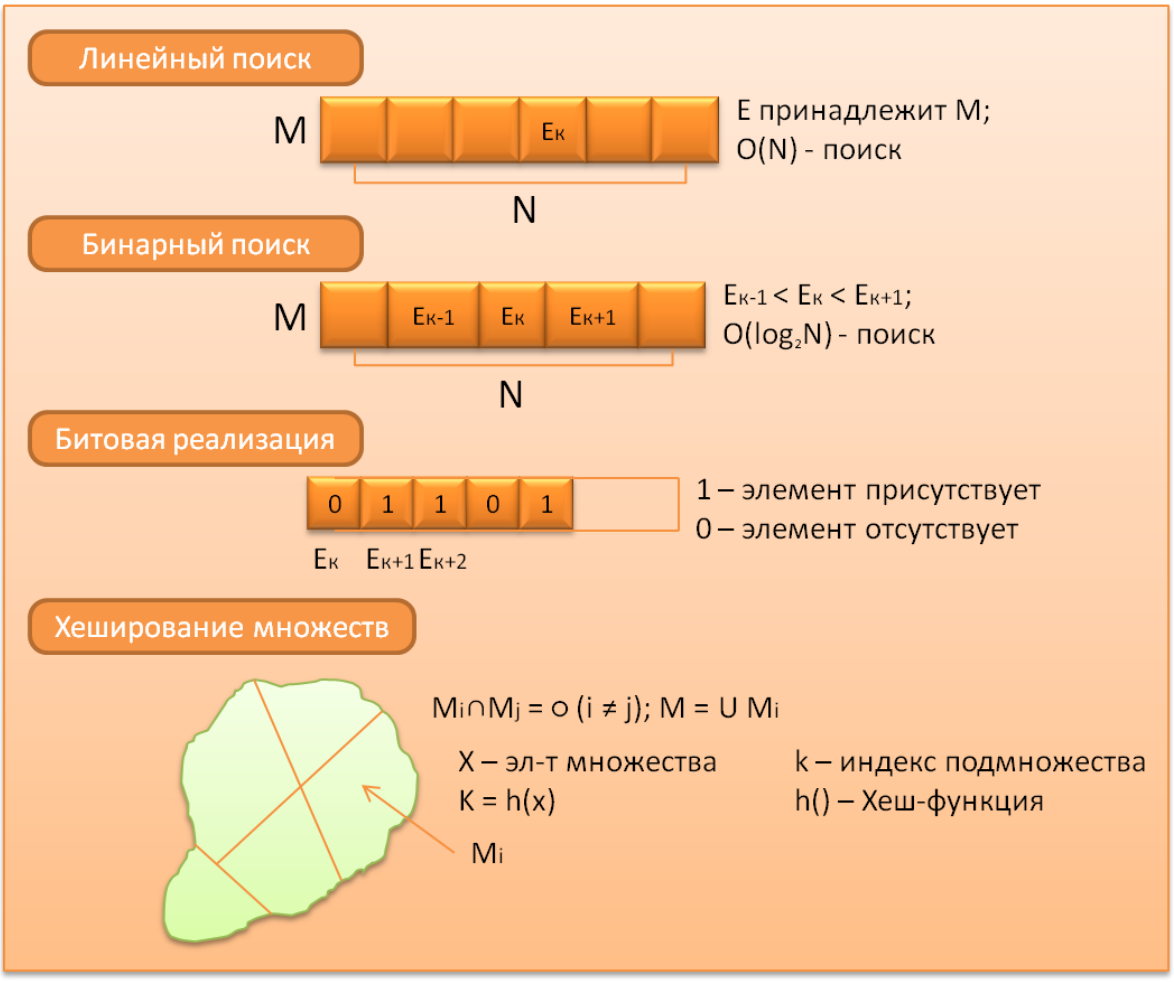
Поиск (основная операция)

1.Линейный поиск

2.Двоичный поиск

3.Битовая реализация

4.Хеширование множеств



Линейный поиск

Рис.24.5а.

Вообще линейный поиск не очень эффективен.

Бинарный поиск

Рис.24.5.б.

Есть затраты на сортировку. Но если отсортировано – то все хорошо, смотри предыдущие разговор. Получим большую эффективность. Область применения сильно лимитирована, как ив прошлом случае.

Битовая реализация

Рис.24.5.в.

Хеширование множеств

Это обособленная часть по факту, хоть и является одним из поисков. Вообще является комбинацией предыдущих, так как мы хотим большую эффективность.

Проблема коллизий

Рис.24.6.

Вместо того, чтобы заполнить память, мы переполнили его часть. Какой же выход из такой ситуации?

Разрешение коллизий

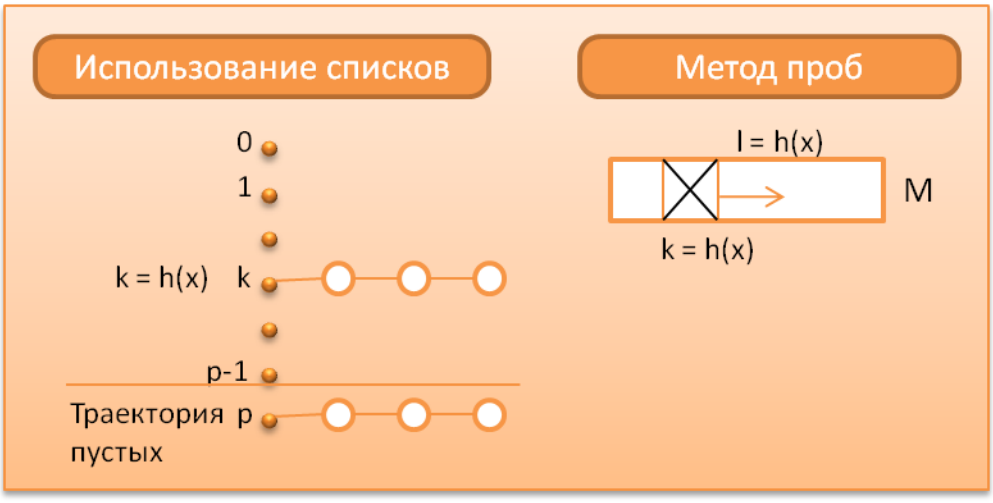


Рис. 24.7

1.Использование списков

2.Метод проб

Использование списков

Имеем некоторые элементы x. Вычисляем функцию хеширование и получаем значение k. k∈

Экзаменационные билеты на 2010 год.

(ходят достоверные слухи, что они же будут и в 2011)

Билет 1

1. Общая архитектура микропроцессорных вычислительных систем. Функциональная схема компьютера. Типы и характеристики микропроцессоров (разрядность, частота синхронизации, организация системы команд). Сравнение CISC и RISC архитектур.
2. Написать программу загрузки из текстового файла последовательности целых чисел, игнорирующую все элементы исходной последовательности, равные среднему арифметическому своих соседей. Напечатать загруженные элементы в порядке их возрастания.

Билет 2

1. Типы и характеристики запоминающих устройств. Организация основной памяти компьютера. Понятие адресного пространства. Реальные и виртуальные адреса. Прямой доступ к основной памяти.
2. Написать программу, отыскивающую во входном файле целых чисел такие, у которых старший байт их внутреннего машинного представления является «зеркальным отражением» младшего байта (например, 01010100.00101010). Напечатать найденные числа в порядке возрастания их абсолютной величины.

Билет 3

1. Многоуровневая организация памяти. Ассоциативная кэш-память. Размещение, поиск и замещение блоков. Стратегии записи. Многоуровневая кэш-память.
2. Выполнить программную реализацию объекта «Стек двухкомпонентных векторов». Максимальная глубина стека задается динамически при инициализации объекта.

Билет 4

1. Организация системной шины персональных компьютеров семейства IBM PC. Взаимодействие основных аппаратных модулей. Управление доступом к основной памяти и подсистеме ввода/вывода. Адресация портов ввода/вывода.
2. Написать программу, загружающую из файла последовательность символьных строк ограниченной длины и располагающую их в элементах динамически создаваемого связного списка. Напечатать загруженные строки в лексико-графическом порядке. Для сравнения строк воспользоваться функцией strcmp().

Билет 5

1. Архитектура микропроцессоров Intel x86. Управление потоком команд. Базирование и индексирование памяти. Управление стеком. Сегментная организация памяти. Формирование исполнительного адреса. Битовые флаги состояния и управления.
2. Написать программу сортировки последовательности целых чисел произвольным методом, принимая за ключ сортировки сумму всех цифр десятичной записи числа. Последовательность загрузить из файла, память для хранения выделить динамически. Напечатать отсортированную последовательность.

Билет 6

1. Представление данных в ЭВМ. Машинная арифметика. Погрешность представления чисел с плавающей точкой и ее влияние на точность вычислений.
2. Числовую матрицу с заданным количеством колонок и заранее неизвестным количеством строк загрузить из файла, выделяя память динамически под каждую ее строку. Написать функцию, печатающую элемент матрицы по его линейному индексу в предположении, что нумерация элементов матрицы ведется по колонкам, начиная с нуля.

Билет 7

1. Организация работы с подпрограммами в режиме реального адреса микропроцессоров Intel x86. Рекурсивные вызовы. Вызовы с параметрами. Механизмы передачи параметров.
2. Написать программу подсчета количества слов в произвольном текстовом файле, принимая за разделители слов любой из символов введенной с клавиатуры цепочки. Определить количество символов в самом длинном слове. Результаты напечатать.

Билет 8

1. Прерывания и программы обработки прерываний. Классификация прерываний. Зарезервированные вектора прерываний режима реального адреса микропроцессоров Intel x86. Последовательность прерывания.
2. Выполнить программную реализацию объекта «Очередь строк фиксированной длины» на базе кольцевого вектора. Максимальная длина очереди задается динамически при инициализации объекта.

Билет 9

1. Обработка внешних аппаратных прерываний. Назначение и работа контроллера прерываний. Последовательность аппаратного прерывания.
2. Написать программу сортировки последовательности целых чисел методом Quick Sort, принимая за ключ сортировки количество единиц в машинном представлении каждого числа. Последовательность загрузить из файла, память для хранения выделить динамически.

Билет 10

1. Работа микропроцессоров Intel x86 в защищенном режиме. Общая схема формирования исполнительного адреса. Кольца защиты. Организация дескрипторных таблиц. Адресное пространство задачи.
2. Написать программу, загружающую из файла последовательность символьных строк ограниченной длины с динамическим выделением памяти для хранения символов каждой строки. Отсортировать строки в порядке убывания количества слов в каждой строке.

Билет 11

1. Работа с подпрограммами в режиме защищенного адреса микропроцессоров Intel x86. Межкольцевые вызовы подпрограмм. Ограничение прав доступа. Управление задачами. Сегмент состояния задачи. Вложенные задачи.
2. Написать программу сортировки последовательности вещественных чисел методом Heap Sort. Последовательность загрузить из файла, память для хранения выделить динамически. Напечатать отсортированную последовательность и количество выполненных операций сравнения ее элементов.

Билет 12

1. Задача управления ресурсами ЭВМ. Операционные системы и их классификация. Иерархия подсистем. Системные вызовы. Примеры реализации системных функций.
2. Написать программу загрузки из текстового файла последовательности целых чисел, игнорирующую элементы исходной последовательности, равные сумме всех предыдущих загруженных элементов. Напечатать загруженные элементы в порядке убывания их абсолютной величины.

Билет 13

1. Понятие процесса. Диаграммы состояний процесса. Параллельное выполнение программ в различных операционных средах. Многопоточные операционные системы.
2. Написать программу загрузки из текстового файла последовательности символьных строк, игнорирующую все строки, содержащие хотя бы одну цепочку символов из заданного с клавиатуры

набора цепочек. Напечатать загруженные строки в порядке возрастания количества символов в каждой строке.

Билет 14

1. Задача управления памятью. Выделение памяти разделами фиксированного и переменного размера. Связное и несвязное распределение памяти. Оверлейные структуры. Системы со свопингом. Виртуальная память. Сегментная и страничная организация виртуальной памяти. Трансляция адресов. Стратегии замещения разделов.
2. Написать программу, загружающую из текстового файла массив записей, каждая из которых содержит одно строковое и одно числовое поле. Записи расположить в элементах динамически создаваемого связного списка. Напечатать загруженные записи в порядке убывания значений числового поля.

Билет 15

1. Взаимодействие процессов. Асинхронно выполняющиеся процессы. Проблема критической секции и семафорные примитивы. Проблема тупика и методы ее решения. Алгоритм банкира.
2. Числовую последовательность неопределенной длины загрузить из текстового файла, выделяя память динамически по мере необходимости блоками фиксированного размера. Распечатать загруженную последовательность в порядке возрастания ее элементов.

Билет 16

1. Модели данных языков программирования, статическая и динамическая память. Управление данными в Си программах. Глобальные и локальные среды. Многомодульные приложения. Классы памяти. Методы структурирования данных. Вызовы функций и механизмы передачи параметров.
2. Написать программу подсчета общего количества и процентного отношения появления каждого символа в текстовом файле. Результат представить в виде таблицы в порядке возрастания частот появления различных символов.

Билет 17

1. Общий подход к абстрагированию и структурированию данных. Понятие объекта и объектно-ориентированные технологии. Последовательности и однопроходные алгоритмы. Моделирование динамических структур данных с последовательным доступом на примере стека и очереди.
2. Написать программу, загружающую из текстового файла последовательность символьных строк ограниченной длины и располагающую их в элементах динамически создаваемого связного списка. Найти в загруженном массиве строк все цепочки символов, являющиеся правильной десятичной записью 16-разрядных двоичных целых чисел без знака.

Билет 18

1. Ссылочные реализации динамических структур данных. Организация списков, операции добавления и исключения элементов. Представление деревьев и графов. Алгоритмы обхода дерева. Деревья поиска. Примеры реализации.
2. Написать программу, загружающую из текстового файла последовательность целых чисел и заменяющую все элементы исходной последовательности с четным количеством единиц во внутреннем машинном представлении на сумму всех ранее прочитанных элементов. Напечатать получившуюся последовательность в порядке возрастания элементов.

Билет 19

1. Реализация множества на базе вектора. Последовательный и двоичный поиск. Битовая реализация множества. Оценка сложности алгоритмов. Хеширование. Методы разрешения коллизий.
2. Написать программу сортировки последовательности целых чисел методом прямого обмена (метод «пузырька»), принимая за ключ сортировки сумму числовых значений старшего и младшего байтов в машинном представлении каждого числа. Последовательность загрузить из файла, память для хранения выделить динамически.

Билет 20

1. Алгоритмы сортировки. Априорная оценка сложности алгоритма. Сравнение эффективности различных алгоритмов сортировки. Особенности реализации.
2. Написать программу загрузки из текстового файла последовательности целых чисел, игнорирующую все элементы исходной последовательности, для которых сумма цифр в их десятичной записи является четным числом. Напечатать загруженные элементы в порядке их убывания.