ОГЛАВЛЕНИЕ

1. ИСТОРИЯ СОЗДАНИЯ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

В 1900г. на затонувшем корабле был обнаружен аналоговый механический компьютер для моделирования движения планет. Градуированные циферблаты могли вращаться независимо друг от друга с пом. приводных механизмов и зубчатых колес. Все зубья заточены под углом 60°. Дифференц. передача состояла из 30 бронзовых шестеренок, 4-х циферблатов и рукоятки.

Чарлз Бэббидж в проекте своей аналитической машины, содержащей арифметическое устройство, память, устройство ввода и устройство печати, предусмотрел управляющие устройства, реализующие управление в зависимости от текущего результата вычислений. х. Бэббидж (сын ч. Бэббиджа) усовершенствовал отдельные части машины так, что смог вычислить таблицу чисел кратных пи.

К. Цузе к 1941 году удалось построить первую действующую вычислительную машину с программным управлением (модель Z3). В 1942 году в лаборатории Bell Telephon была сдана в эксплуатацию вычислительная машина на электромагнитных реле, которую разработал Джордж Штибиц. В 1944 году правительство США приняло в эксплуатацию вычислительную машину Mark I, созданную х. Айкеном. Первая вычислительная машина на электронных лампах ENIAC Дж. Эккерта и Дж. Маучли, проектирование которой велось с 1943 года, начала функционировать в 1946 году. Эти устройства не достигли той универсальности, которой обладал проект К. Цузе.

Основополагающая идея вычислительной машины, управляемой размещенной в ее памяти программой, впервые описана Джоном фон Нейманом и его сотрудниками Дж. Маучли и Дж. Эккертом 30 июня 1945 юда. Она бъта развита Морисом Уилксом, который В мае 1949 юда сдал в ЭКСlШуатацию первую вычислительную машину EDSAC.

В 1944 году Маучли, Эккерт и голдстайн ввели принцип последовательных вычислений, применив одноразрядный сумматор для после довательного суммирования п-разрядных чисел.

В 1948 году С.А. Лебедев (СССР) независимо от Дж. фон Неймана обосновал принцип построения ЭВМ с хранимой в памяти программой.

Первая в СССР цифровая вычислительная машина (ЦВМ) МЭСМ создана в 1951 году под руководством С.А. Лебедева. В 1952 году появилась ЭВМ М--I (руководители И.С. Брук, н.я. Матюхин), а в 1953 году ЭВМ «Стрела» (создатели ю.я. Базилевский, Б.И. Рамеев). В 1955 году в СССР организован первый ВЦ АН СССР (директор А.А. Дородницин).

В 1958 году в СССР создана первая и единственная в мире ЭВМ «Сетунь» (Н.Л. Брусенцов), работающая в троичной системе счисления с симметричным представлением цифр, а также первая ЭВМ, применяющая систему в коде вычетов (и.я. Акушский, д.и. Юдицкий).

2. КЛАССИФИКАЦИЯ ПО

Программное обеспечение (ПО) - это совокупность всех программ и соответствующей документации, обеспечивающая использование ЭВМ в интересах каждого ее пользователя.

По способу исполнения программы делят на: интерпретируемые; компилируемые.

По степени переносимости программы делят на платформозависимые; кроссплатформенные.

По способу распространения и использования программы делят на несвободные (закрытые); открытые; свободные.

По назначению программы делят на: системные; прикладные; инструментальные.

Системное ПО. Комплекс программ, которые обеспечивают управление компонентами компьютерной системы, такими как процессор, оперативная память, устройства ввода-вывода, сетевое оборудование. Выступает как «межслойный интерфейс», с одной стороны которого аппаратура, а с другой — приложения пользователя. Как правило, к системному программному обеспечению относятся: ОС, утилиты, СУБД, широкий класс связующего ПО.

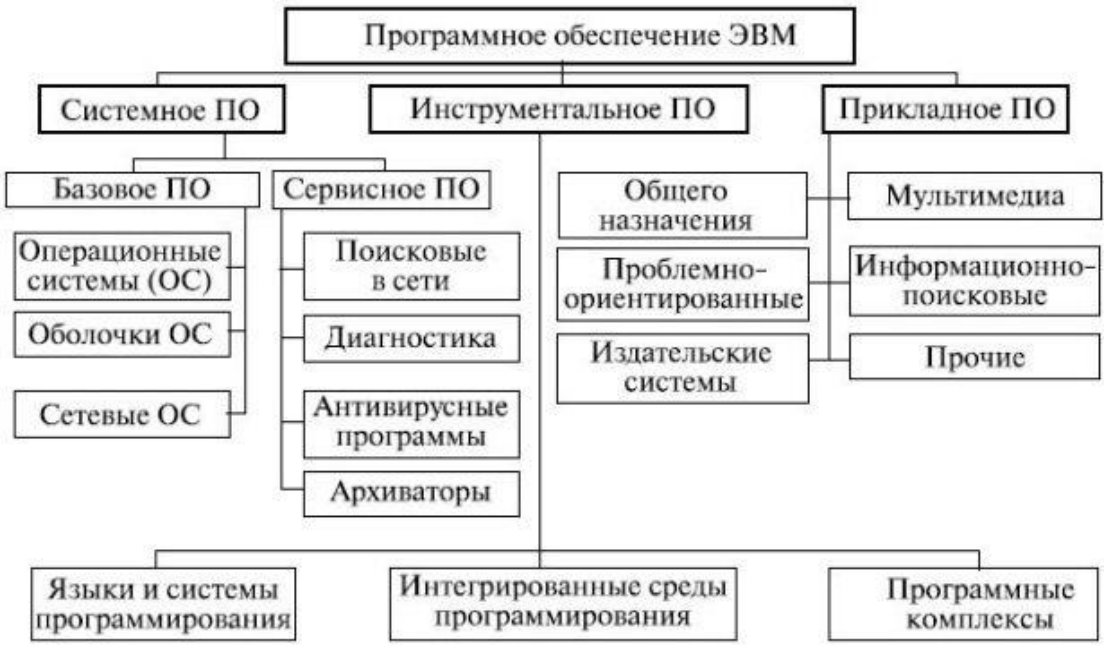
Встроенные программы или firmware — это программы, «зашитые» в цифровые электронные устройства.

Утилиты – вспомогательные компьютерные программы в составе общего программного обеспечения для выполнения специализированных типовых задач, связанных с работой оборудования и операционной системы.

Прикладное ПО — программа, предназначенная для выполнения определенных пользовательских задач и рассчитанная на непосредственное взаимодействие с пользователем.

Инструментальное ПО – ПО, предназначенное для использования в ходе проектирования, разработки и сопровождения программ, в отличие от прикладного и системного программного обеспечения. К ним относятся:

- трансляторы;
- среду разработки программ;
- библиотеки справочных программ (функций, процедур);
- отладчики;
- редакторы связей и др.



3. ДВА ПОДХОДА К ФОРМИРОВАНИЮ ПОНЯТИЯ «АРХИТЕКТУРА КОМПЬЮТЕРА»

Архитектура как набор взаимодействующих компонент

Архитектура ВС определяет основные функциональные возможности системы, сферу применения, режим работы, характеризует параметры ВС, особенности структуры и т. д.

Вычислительные и логические возможности ВС. Они обусловливаются системой команд (СК), характеризующей гибкость программирования, форматами данных и скоростью выполнения операций, определяющих класс задач, наиболее эффективно решаемых на ВС.

К командам управления мы относим команды ввода-вывода данных и команды управления состоянием процессора, памяти и каналов.

Большое влияние на точность выполнения операций оказывают форматы данных. *Аппаратные средства.* Простейшая ВС включает модули пяти типов: центральный процессор, основная память, каналы, контроллеры и внешние устройства.

Процессор (УУ + АЛУ + намять) управляет работой системы и обеспечивает вычисления непосредственно по программе. Выполнение машинных команд, команд ввода-вывода (I/O), обращение к памяти, управление состоянием устройств инициализируются или выполняются с помощью процессора.

Основная память предназначена для хранения команд и данных и обеспечивает адресный доступ к ним от процессора.

Каналы - специальные устройства, управляющие обменом данных с внешними устройствами. Каналы инициируют свою работу с помощью процессора и затем переходят в автономный режим работы.

Контроллеры ввода-вывода служат для подсоединения внешних устройств (ВнУ) к каналам и обеспечивают обмен управляющей информацией с внешними устройствами, присвоение приоритетов и выдачу информации о состоянии ВнУ для канала, т. е. это устройства управления ВнУ.

ВнУ служат для ввода-вывода информации с различных носителей.

Память может быть организована как многоуровневая с различным объемом и временем доступа к ней - сверхоперативная (СОЗУ). оперативная (ОП), внешняя (ВнП), так и одноуровневая, виртуальная. Уровни иерархии памяти взаимосвязаны между собой: все данные одного уровня могут быть найдены на более низком уровне.

Программное обеспечение. является составной частью архитектуры компьютера и существенно влияет на весь вычислительный процесс,

Операционная система (ОС) управляет ресурсами, разрешает конфликтные ситуации, оптимизирует функционирование системы в целом.

Архитектура как интерфейс между уровнями физической системы

Применительно к ВС термин "архитектура" может быть определен как распределение функций, выполняемых системой, по различным уровням и установление интерфейса между этими уровнями.

Архитектура первого уровня определяет, какие функции по обработке данных решаются системой, а какие передаются внешнему миру: пользователю, оператору ЭВМ. администратору баз данных и т. д. Система взаимодействует с внешним миром через два набора интерфейсов: языки (язык программирования, язык оператора терминала, язык управления заданиями, язык общения с базой данных, язык оператора ЭВМ) и *системные программы* (программы редактирования, связи, оптимизации, восстановления и обновления информации, интерпретации, управления и т. д., т. е. программы, созданные разработчиком системы). Оба интерфейса должны быть созданы при разработке архитектуры системы.

Уровни, определяемые интерфейсами внутри программного обеспечения, могут быть представлены как архитектура программного обеспечения.

Таким образом, можно сказать, что архитектура компьютера - это абстрактное представление физической системы с точки зрения программиста.

Процесс разработки архитектуры ЭВМ включает все этапы разработки типовых проектов, анализ требований. предъявляемых к системе; составление спецификаций; изучение известных решений; разработку функциональной схемы; разработку структурной схемы; отладку проекта; оценку проекта.

При анализе требований, например, мы учитываем спектр необходимых языков программирования, способ взаимодействия с окружающей средой, требования к операционной системе, некоторые технико-экономические факторы: количество таких систем, совместимость разрабатываемой системы с существующими аналогичными вычислительными системами и т. д.

При анализе требований и составлении перечня спецификаций параметров системы необходимо разработать критерии, определяющие стоимость, надежность. защиту от несанкционированного доступа, совместимость и т. д. и описать функции системы в соответствии с этими критериями. В спецификациях учитывается и существующий или достижимый технологический уровень для производства системы.

На этапе разработки функциональной схемы необходимо в первую очередь решить вопрос о разграничении функций между аппаратурой и программным обеспечением, а также между другими уровнями архитектуры.При структурной организации все вопросы детализируются и углубляются. В частности, здесь решаются проблемы типов и форматов команд, способов представления данных, способов адресации, семантики языка.

При завершении проекта идет его отладка, согласование взаимодействия отдельных уровней и оптимизация, например, по количеству битов, пересылаемых между процессором и памятью при выполнении данной программы, отысканию эффективного метода кодирования команд и числовой информации и т. д.

Оценка проекта. Простая оценка быстродействия мало что определяет. Здесь необходимо учитывать время, затраченное на разработку программы. Используя известные статистические данные о том. что преобладающее количество разработанных программ выполняется только один раз, простота программирования при оценке архитекторы может оказаться важнее скорости выполнения команд.

4. АРХИТЕКТУРА ФОН НЕЙМАНА: ПРИНЦИПЫ, ПРОБЛЕМЫ И СПОСОБЫ ИХ РЕШЕНИЯ

Архитектурные принципы фон Нейманом формулировались применительно к созданию автоматического устройства для решения ДУ.

Основные характеристики архитектуры фон Неймановского типа следующие:

1) последовательно адресуемая единственная память линейного типа для хранения программ и данных;

2) команды и данные различаются через идентификатор неявным способом лишь при выполнении операций. Принимаемые по умолчанию соглашения типа: операнды операции умножения это данные, а объект, на который указывает команда перехода - это команда, позволяют обращаться с командой как с данными, например, для ее модификации;

3) назначение данных определяется лишь логикой программы, так как в памяти машины набор бит может представлять собой как десятичное число с фиксированной точкой, так и строку символов,

Указанные свойства были исключительно важными для своего времени. Однако появление языков высокого уровня (ЯВУ), новых методов решения, логических способов ускорения операций, более совершенной элементной базы требует наряду с имеющимися возможностями архитектуры и принципиально новых. Среди них требования ЯВУ имеют следующие особенности:

- « память состоит из набора дискретных именуемых переменных. Таким образом, принцип единственной последовательной памяти имеет мало общею с организацией памяти в ЯВУ:
- ЯВУ наряду с линейными данными оперируют и с многомерными: массивами, структурами, списками:
- в ЯВУ четко разграничены операции и данные;
- данные определяют и операции над ними.

Архитектура фон Неймана плохо ориентирована на выполнение программ на ЯВУ. Действительно,

- объем кодов, генерируемых компилятором, из-за несоответствия требуемой ЯВУ и предлагаемой архитектурой организации памяти значительно превосходит необходимый объем для решения запрограммированной задачи;
- примитивность выполняемых операций в объектном коде требует сложной работы компилятора.

5. ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ВС

Разработка архитектуры включает все этапы создания типового проекта:

1) исследование концепции решения
2) анализ требований, предъявляемых к системе
3) составление спецификаций
4) изучение известных решений
5) разработка функциональной схемы
6) разработка структурной схемы
7) автономная отладка компонент
8) интегральная отладка компонент
9) аттестация проекта
10) оценка проекта
11) установка проекта и заказчика
12) обучение персонала
13) сопровождение проекта
14) замена новыми архитектурными решениями

В реальной жизни линейное выполнение этапов не всегда приводит к результату, т.к. на некоторых этапах иногда возникает необходимость вернуться на предыдущие шаги с тем, чтобы изменить, уточнить, доработать, адаптировать проект.

При анализетребоваIBfЙ, например, учитывается спектр необходимых языков программирования, способ взаимодействия с окружающей средой, требования к операционной системе, некоторые техникоэкономические факторы..... предполагаемое количество таких систем (ибо это существенно влияет на соотношение между аппаратным и про[раммным обеспечением), совместимость разрабатываемой системы (по языкам программирования, операционным системам, форматам данных, техническим устройствам) с существующими аналогичными вычислительными системами и т.д.

При анализе требований и составлении перечJIЯ спецификаций пара метров системы необходимо разработать критерии (с их весовыми коэффициентами), определяющие стоимость, надежность, защиту от несанкционированного доступа, совместимость и Т.Д., и описать функции системы в соответствии с этими критериями. В спецификациях учитывается и существующий или достижимый технологический уровень ДШI производства системы. Естественно, что возникающиепротиворечивые требования должны быть разрешены в соответствии с приоритетами или на другой основе.

На этапе разработки функциональной схемы в первую очередь решается вопрос о разграничении ФУНКIIИЙ между аппаратурой и программным обеспечением, а также между другими УРОВJIЯми архитектуры. Здесь же определяются необходимость работы с плавающей или фиксированной точкой, приоритетные прерывания, стековая организа ция памяти и т.д.

При структурной организации все вопросы детализируются и углубляются. В частности, здесь решаются проблемы типов и форматов команд, способов представления данных, способов адресации, семантики языка.

По завершении проекта осуществляется его отладка, согласование взаимодействия отдельных уровней и оптимизация, например, по количеству бит, пересылаемых между процессором и памятью при выполнении данной программы, отыскание эффективного метода кодирования команд и числовой информации и т.д.

При оценке проекта учет только быстродействия (которое иногда понимается как производительность) недостаточен. Здесь необходимо учитывать также время, требуемое на разработку программы. Используя известные статистические сведения о том, что преобладающее количество разработанных программ выполняется только один раз,

простота программирования при оценке архитектуры может оказаться важнее скорости выполнения команд.

Однако скорость выполнения операций достаточно важный параметр. Быстродействие, естественно, зависит как от технологической элементной базы, ВЛИЯющей на скорость обработки данных, так и от архитектуры машины, которая определяет объем выполняемых работ. Часто архитектуры сравниваются по критерию эффективности обработки информации. Для этого, как правило, анализируются СЛе дующие параметры:

S размер программы, определяемый длиной команд, размером косвенных адресов, объемом рабочих областей ДШ ВреМеННого разметения данных;

N p количество бит, передаваемых меЖдУ процессором и памятью машины (пересьтка между регистрами) за время выполнения програмМЬI, характеризующее интерфейс, то есть «полосу пропускания»между процессором и памятью, что в значительной степени определяет предел эффективности выполнения программ. Ясно, что параметр должен учитывать и локальность ссьток, то есть близость расположения в памяти используемых данных;

N R количество бит, передаваемых между внугренними регистрами процессора за время выполнения программы. Здесь учитываются те пересьтки, которые не охватывает параметр N p . Величина N R в значительной степени определяется набором функций, воплощенных в АЛУ и в схемной реализации процессора.

Иногда архитектуры сравнивают лишь по параметрам S и N p , исключая параметр N R , тем самым игнорируя внугреннюю организацию процессора.

Опытная эксплуатация позволяет сопровождать различные компоненты архитектуры, выявлять неточности, модифицировать, совершенствовать и уточнять отдельные аспекты.

Серийное производство позволяет создавать стереотип мышления определенной группы разработчиков и пользователей в области создания архитектур компьютеров.

Иногда в перечень последовательности проектирования включают и этап вывода проекта из эксплуатации, состоящий в полной его замене новой или модернизированной версией.

Во МНогоМ качество реализации проекта зависит от подбора команды разработчиков. Например кодекс разработчиков программных проектов бьт создан объединенной специальной комиссией IEEE Computer SocietyjASM. Этот документ включает следующие принципы:

общественные интересы действия программистов должны соответствовать общественным интересам;

клиент программисты должны максимально учитывать требования работодателя, но при этом соБЛЮl{ать общественные интересы;

продукт программисты должны быть уверены в том, что разрабатыIаемые ими проекты (и их модификации) соответствуют самым высоким профессиональным стандартам;

критицизм программисты должны придерживаться целостности и независимости своих суждений, формируя здоровый профессиональный критицизм МЫШШ1ения;

менеджмент менеджеры, управляющие группами разработчиков, обязаны придерживаться этических норм в процессе разработки и сопровождения продукта;

профессионализм

коллеziальность программисты должны поддерживать своих колЛеf, помогать им в их профессиональной деятельности, в частности в освоении современных методик работы;

самосовершенствование

6. ТИПЫ КОМАНД И ТЕХНИКА АДРЕСАЦИИ

В машинах с регистрами общего назначения метод (или режим) адресации объектов, с которыми манипулирует команда, может задавать константу, регистр или ячейку памяти.

Метод адресации	Пример команды	Смысл команды	Использование команды
Регистровая	Add R4, R3	R4 = R4+R3	Для записи требуемого значения в регистр
Непосредственная или литерная	Add R4, #3	R4 = R4+3	Для задания констант
Базовая со смещением	Add R4, 100(R1)	R4 = R4+M(100+R1)	Для обращения к локальным переменным
Косвенная регистровая	Add R4, (Rl)	R4 = R4+M(R1)	Для обращения по указателю к вычисленном у адресу
Индексная	Add R3, (R1+R2)	R3=R3+M(R1+R2)	Полезна при работе с массивами: R1 - база. R3 - индекс
Прямая или абсолютная	Add Rl. (1000)	R1=R1+M(1000)	Полезна для обращения к статическим данным
Косвенная	Add Rl, @(R3)	R1 - R1+M(M(R3))	Если R3 – адрес указателя p, то выбирается значение по этому
Автоинкрементная	Add Rl, (R2)+	Rl = R1+M(R2) R2 = R2+d	Полезна для прохода в цикле по массиву с шагом: R2 – начало массива. В каждом цикле R2 получает приращение d
Автодекрементная	Add Rl, (R2)-	R2 = R2-d Rl = R1+M(R2)	
Базовая индексная со смещением и масштабированием	Add R1, 100(R2)[R3]	R1=R1+M(100)+R2-+R3*d	

Адресация непосредственных данных и литерных констант обычно рассматривается как один из методов адресации памяти (хотя значения данных, к которым в этом случае производятся обращения, являются частью самой команды и обрабатываются в общем потоке команд).

В табл. 2.2 на примере команды сложения (Add) приведены наиболее употребительные названия методов адресации, хотя при описании архитектуры в документации производители компьютеров и ПО используют разные названия для этих методов. Использование сложных методов адресации позволяет существенно сократить количество команд в программе, но при этом значительно увеличивается сложность аппаратуры.

Команды традиционного машинного уровня можно разделить на несколько типов, которые показаны в след. таблице.

Тип операции	Примеры
Арифметические и логические	Целочисленные арифметические и логические операции: сложение, вычитание, логическое сложение, логическое умножение и т. д.
Пересылки данных	Операции загрузки/записи
Управление потоком команд	Безусловные и условные переходы, вызовы процедур и возвраты
Системные операции	Системные вызовы, команды управления виртуальной памятью и т. д.
Операции с плавающей точкой	Операции сложения, вычитания, умножения и деления над вещественными числами
Десятичные операции	Десятичное сложение, умножение, преобразование форматов и т. д.
Операции над строкам и	Пересылки, сравнения и поиск строк

Тип операнда может задаваться либо кодом операции в команде, либо с помощью тега, который хранится вместе с данными и интерпретируется аппаратурой во время обработки данных.

Обычно тип операнда (целый, вещественный, символ) определяет и его размер. Как правило, целые числа представляются в дополнительном коде. Для задания символов компания IBM использует код EBCDIC, другие компании применяют код ASCII. Для представления вещественных чисел с одинарной и двойной точностью придерживаются стандарта IEEE 754.В ряде процессоров применяют двоично-кодированные десятичные числа, которые представляют в упакованном и неупакованном форматах. Упакованный формат предполагает, что для кодирования цифр 0 - 9 используют 4 разряда и две десятичные цифры упаковываются в каждый байт. В неупакованном формате байт содержит однудесятичную цифру, которая обычно изображается в символьном коде ASCII.

7. ИЕРАРХИЯ ПАМЯТИ: РЕГИСТРОВАЯ, КЭШ, ОПЕРАТИВНАЯ ГЛАВНАЯ И ВСПОМОГАТЕЛЬНАЯ

Память компьютера предназначена для хранения информации. В компьютере имеются два вида памяти: внутренняя и внешняя. Внутренняя память расположена в системном блоке. У компьютера есть три вида внутренней памяти: постоянное запоминающее устройство (ПЗУ) на микросхемах памяти только со считыванием — ROM (Read Only Memory), оперативное запоминающее устройство (ОЗУ), так называемая RAM (Random Access Memory), и кэш-память. Микросхемы памяти относятся к разряду ключевых комплектующих ПК: то есть тех, без которых компьютер не может функционировать. Память современных ПК выполнена на интегральных микросхемах.

ОЗУ используется только для временного хранения программ и данных. ОЗУ - энергозависимая память: при отключении напряжения питания информация, хранящаяся в ней, теряется. Чем больший объем оперативной памяти установлен на компьютере, тем быстрее и комфортнее работать на нем пользователю. ОЗУ строится на микросхемах памяти с произвольным доступом к любой ячейке. ОП бывает либо статической (на триггерах) и называется SRAM (Static RAM), либо динамической DRAM (Dynamic RAM). Быстродействующее запоминающее устройство в компьютерных системах — это память, дающая возможность центральному процессору хранить текущие программы и данные. Процессор компьютера может работать только с данными, которые находятся в оперативной памяти. Данные с диска для обработки считываются в оперативную память.

Память ПЗУ— постоянная память компьютера, или постоянное запоминающее устройство — это устройство памяти в виде набора интегральных схем (чипов), часто использующееся в микропроцессорах. Данные записываются в ПЗУ при производстве компьютеров непосредственно на фабрике. Информацию в постоянной памяти нельзя изменять — она инициализируется только для чтения. Информация в ПЗУ сохраняется даже при отключении электропитания. Такой вид памяти используется для хранения внутри компьютера программ системного тестирования, начальной установки конфигурации Setup и ввода/вывода. Совокупность этих программ называется базовой системой ввода/вывода BIOS (Basic Input/Output System).

Память в персональном компьютере делится на внутреннюю, расположенную на системной плате, и внешнюю. Внутренняя память в свою очередь можно разделить на КЭШ - память и основную память. Регистровая КЭШ-память - высокоскоростная память, являющаяся буфером между оперативной памятью и микропроцессором, позволяющая увеличивать скорость выполнения операций. Регистры КЭШ-памяти недоступны для пользователя.

По принципу записи результатов различают два типа КЭШ-памяти:

КЭШ - память "с обратной записью” - результаты операций прежде, чем записать их в ОП. фиксируются в КЭШ -памяти, а затем контроллер КЭШ - памяти самостоятельно перезаписывает эти данные в ОП;

КЭШ - память "со сквозной записью" - результаты операций одновременно, параллельно записываются и в КЭШ - память, и в ОП.

Следует иметь в виду, что для всех МП может использоваться дополнительная КЭШ-память (КЭШ -память 2-го уровня), размещаемая на материнской плате вне МП, емкость которой может достигать нескольких Мб.

Внешняя память

Внешняя (долговременная) память — это место длительного хранения данных (программ, результатов расчётов, текстов и т.д.), не используемых в данный момент в оперативной памяти компьютера. Внешняя память, в отличие от оперативной, является энергонезависимой. Носители внешней памяти, кроме того, обеспечивают транспортировку данных в тех случаях, когда компьютеры не объединены в сети (локальные или глобальные).

Для работы с внешней памятью необходимо наличие *накопителя* (устройства, обеспечивающего запись и (или) считывание информации) и устройства хранения — *носителя*.

Основные виды накопителей:

- накопители на гибких магнитных дисках (НГМД);
- накопители на жестких магнитных дисках (НЖМД);
- накопители на магнитной ленте (НМЛ);
- накопители CD-ROM, CD-RW, DVD.

Им соответствуют основные виды носителей:

гибкие магнитные диски (*Floppy Disk*) (диаметром 3,5’’ и ёмкостью 1,44 Мб; диаметром 5,25’’ и ёмкостью 1,2 Мб (в настоящее время устарели и практически не используются, выпуск накопителей, предназначенных для дисков диаметром 5,25’’, тоже прекращён)), диски для сменных носителей;

- жёсткие магнитные диски (*Hard Disk*);
- кассеты для стримеров и других НМЛ;
- диски CD-ROM, CD-R, CD-RW, DVD.

8. ОРГАНИЗАЦИЯ КЭШ-ПАМЯТИ.

Кеши — следующий уровень иерархии памяти. Кэши сравнительно небольшие по объёму, но имеют высокую скорость доступа. В кэш-памяти хранятся команды и данные, которые часто используются и требуют малых временных затрат для доступа к ним.

Обычно никакое явное (программное) управление входами кэша невозможно. Данные распределяются и управляются кэшем автоматически.

Кэш-память позволяет организовать работу медленной оперативной памяти как быстродействующей, оптимизируя следующие аспекты:

- Максимизация коэффициента попадания;
- Уменьшение времени доступа;
- Уменьшение штрафа промаха;

Коэффициент попадания в кэш (в %) определяется отношением количества успешных входов к числу промахов.

Принципы создания

Эффективность кэширования базируется на свойстве локализации ссылок во времени и локализации в пространстве. Реализуя принцип *локализации во времени,* каждый элемент (команда, данные), к которому обращается процессор, копируется в кэш, где он хранится до очередного востребования. Поскольку обращения к памяти носят не случайный характер, а происходят в соответствии с выполняемой программой, то при считывании данных из памяти с высокой степенью вероятности можно предположить, что в ближайшем будущем опять произойдёт обращение к этим же данным.

Реализуя принцип *локализации в пространстве,* наряду с текущим элементом в кэш копируется и несколько близлежащих элементов.

В системах, оснащённых кэш-памятью, каждый запрос к оперативной памяти выполняется следующим образом: просматривается содержимое кэш-памяти с целью определения, не находятся ли нужные данные в кэш-памяти. При возникновении промаха контроллер кэш-памяти выбирает подлежащий замещению блок. Как правило, используются две стратегии: случайная (блок выбирается случайно) и LRU (Least-Recently-Used - заменяется блок, не использовавшийся дольше остальных).

Если некоторый блок основной памяти может располагаться в любом месте кэш-памяти, то кэш называется полностью ассоциативным. Если блок основной памяти может располагаться на ограниченном множестве мест в кэш-памяти, то кэш называется множественно-ассоциативным. Обычно множество представляет группу из нескольких блоков в кэше. Если множество состоит из *n* блоков, то такое размещение называется множественно-ассоциативным с *n* каналами.

Принципы размещения блоков определяют основные типы организации кэш-памяти. Если каждый блок основной памяти имеет только одно фиксированное место, на котором он может появится в кэш-памяти, то такая кэщ-память называется кэшем с прямым отображением.

Для записи в кэш-память имеется два способа:

Сквозная запись (write through) - информация записывается сразу в блок КЭШа и в блок более низкого уровня памяти:

Запись с обратным копированием (write back) - информация записывается только в блок кэш-памяти. Модифицированный блок кэш-памяти записывается в основную память только в случае его замещения. Для сокращения частоты копирования блоков при замещении с каждым блоком связывается бит модификации (dirty bit), показывающий, был ли изменён данный блок.

Когда процесс ожидает завершения обращения к памяти при выполнении сквозной записи, то говорят, что он приостанавливается для записи (write stall). Для минимизации приостановок используется буфер записи (write buffer), который позволяет процессору продолжить выполнение других команд во время обновления содержимого памяти. Следует отметить, что приостановки могут возникать и при наличии буфера записи.

9. КОНЦЕПЦИЯ ВИРТУАЛЬНОЙ ПАМЯТИ И ПРЕОБРАЗОВАНИЕ АДРЕСОВ

Каждая часть среды компьютера имеет собственное обозначение: ячейка - адрес, периферийное устройство - номер и т. д. В простейших компьютерах собственные обозначения указывается непосредственно в программе. В более сложных компьютерах программа отделена от среды "аппаратом преобразования собственных обозначений". Рассмотрим один элемент среды - память. Аппарат преобразования адреса (АПА) не находится под прямым управлением программы и связь с ним осуществляется только через процедуры, работающие в управляющем режиме.

Если программисту безразлично существование АПА, то он работает с набором ячеек и периферийных устройств, образующих "виртуальную (математическую, мнимую) среду". Почти всегда виртуальная среда есть переупорядоченное подмножество реальной среды. Каждому виртуальному элементу соответствует реальный элемент, но обратное не всегда верно.

Рассмотрим один из элементов виртуальной среды - виртуальную память (ВП).

Задачи, решаемые виртуальной памятью.

Виртуальный адрес - адрес, по которому ссылаются на ячейку виртуальной памяти. Область виртуальных адресов - это множество всех виртуальных адресов. Использование виртуальной адресации обусловливается следующими обстоятельствами:

Однородность области адресов. Представим себе реальный компьютер без виртуальной памяти. Пусть на нем выполняется параллельно несколько процессов. У каждого процесса будет отдельная локальная среда и каким-то образом распределяемые элементы общей среды. Программисту требуется заранее знать, к каким конкретно частям общей среды его процедура может обращаться. Это затруднительно для пользователей ЭВМ, составляющих свои собственные программы. Отвести наперед фиксированную область среды для каждого процесса невозможно, ибо положение каждой конкретной программы определяется положением всех других программ.

При виртуальной адресации каждый процесс может выполняться в памяти начиная с фиксированной ячейки, имеющей необходимые размеры области ЗУ. Автору безразлично, в каком участке памяти выполняется его программа, так как каждое обращение к виртуальной памяти во время выполнения посредством АПА преобразуется в реальное обращение.

Зашита памяти. Общеизвестно, что основная цель защиты памяти состоит в том, чтобы не дать возможности некорректному процессу испортить часть среды, относящуюся к другому процессу. Особенно это важно при защите сред управляющих процедур. Виртуальная адресация здесь используется следующим образом: при каждой ссылке процессом на память проверяется, принадлежит ли она к области виртуальных адресов, отведенных для данного процесса.

Изменение структуры памяти. При проектировании больших программ структура памяти машины с малой ОП явно усложняет проектируемую программу. Применение виртуальной адресации позволяет преобразовать память на разных ступенях иерархии в "одноуровневую память" с одинаковым доступом ко всем элементам и ее отображение на реальную память.

Страничная организация памяти.

Отображение виртуальной памяти в реальную обычно осуществляется с помощью страничной организации памяти.

Виртуальную намять в системе со страничной организацией памяти делят на ряд "блоков" фиксированной длины, равной 2^k, где k целое натуральное число. Так как первая

ячейка блока N + 1 примыкает к последней ячейке блока N, то программисту факт разбиения ВП на блоки учитывать не требуется.

Оперативная память компьютера делится на "страницы", а вспомогательная - на "сегменты" такого же размера. Виртуальную память пользователя можно разделить на три типа: “активные" блоки, которые содержат программу и данные, используемые в текущий момент; "пассивные" блоки, содержащие программу и данные, которые будут использоваться при выполнении программы; "мнимые" блоки, к которым не обращаются на протяжении выполнения программы.

Аппарат виртуальной адресации должен отображать виртуальную среду на реальную, причем так, чтобы "активные" блоки находились в оперативной памяти, "пассивные" – по возможности в оперативной или вспомогательной, а мнимые – нигде.

Преобразование виртуального адреса в реальный происходит с помощью регистров адресов страниц (РАС). Структура виртуального адреса: | N блока | N строки |

Каждой странице ОП соответствует свой РАС:

| N блока, занимающего страницу | бит записи| бит использования | бит активности |

Аппарат виртуальной адресации отображает виртуальный в реальный следующим образом: виртуальный адрес сравнивается одновременно с содержимым всех РАС. Единственным сравнимым с ним РАС будет тот, который содержит тот же номер блока и "1" в разряде активности. РАС определяет номер страницы, с которой он связан. Для получения реального адреса памяти к номеру страницы данного РАС присоединяется номер строки из виртуального адреса (ВА).

Разряд записи в РАС служит для экономии времени перезаписи "страницы" в ВнП. Когда блок переносится из ВнП в оперативную память в разряд записи пишут "0". Если какая-то строка данной страницы ОП изменяется в процессе обращения к ней, то в пишут "1". И пока в разряде записи "0" эта страница является точной копией соответствующего блока в ВнП.

В бит использования "1" посылается при очередном обращении к данной странице. Через равные промежутки времени содержимое регистра использования сканируется и записывается в опре-ю ячейку памяти, тем самым создавая статистику использования данной страницы. Подобная статистика полезна при замещении страниц в ОП.

Для создания РАС требуется очень дорогая ассоциативная память, поэтому в mainframe-компьютерах число РАС меньше количества страниц ОП. В них каждому блоку ВП отводится в ОП одна последовательная ячейка памяти, указывающая, где хранится данный блок (ОП, ВнП или нигде). Имеется небольшое количество ассоциативных регистров для РАС (обычно 8 или 16).

Полный процесс отображения ВП в реальную выполняется за три этапа.

1. Если происходит обращение к блоку ВП, который должен отображаться на страницу ОП, РАС для которой является одним из 8 (16) ассоциативных регистров, то процесс отображения выполняется согласно схеме, приведенной на рис. 3.

2. Если обращение происходит к блоку ВП, для которого условия п. 1 не выполняются, то номер виртуального блока служит указателем для обращения к таблице блоков (ТБ) в оперативной памяти. Если из ТБ видно, что искомый блок находится в оперативной памяти, то номер страницы также может быть выбран из ТБ. Если же нужный блок находится во вспомогательной памяти или отсутствует вообще, то происходит прерывание, и тогда переходим к управляющей процедуре (см. п. 3).

3. Обработка прерывания.

Основная цель методов оптимизации распределения реальной памяти состоит в минимизации числа обменов между оперативной и вспомогательной памятью.

Основные стратегии замещения страниц, наиболее часто используемые на практике: циклическое изгнание, случайная выборка, наименьшее число обращений с момента последнего прерывания. Результаты экспериментов показали, что ни в одном случае разница в числе обменов, требуемых конкретной задачей при использовании указанных стратегий, не превышала 10 %.

Перенеся концепцию виртуальной памяти на другие компоненты компьютера, мы придем к концепции виртуального компьютера.

10. ФЛЭШ-ПАМЯТЬ

Флэш--памятъ это особый вид памяти, который используется, как правило, для мобильных устройств. Благодаря своим параметрам флэш-память применяется во всех видах мобильных устройств. Она не требует дополнительной энергии для хранения данных (энергия требуется только для записи), допускает перезапись хранимых в ней данных. Флэш-память не содержит механических движущихся частей (как обычные жесткие диски или CO), а поэтому потребляет значительно меньше энергии, что является одним из основных ее преимуществ. В зависимости от типа флэш-памяти возможна перезапись информации от 10 000 до 1 000 000 раз.

Информация, записанная на флэш-память, может храниться длительное время (от 20 до 100 лет) и способна выдерживать значительные механические нагрузки, в 510 раз превышающие предельно допустимые для обычных жестких дисков.

Флэш-память очень компактна. Размер флэш-карты составляет от 20 до 40 мм (по длине или ширине), толщина до 3 мм. Флэш-карта может быть вмонтирована в мобильное устройство, а может быть переносимой и использоваться в нескольких устройствах (например, флэш-карта цифрового фотоаппарата может быть прочитана на обычном компьютере). В настоящее время микросхемы флэш-памяти производят более 50 компаний в мире. В техническом описании любого мобильного устройства сейчас указывается тип используемой флэш-памяти.

В простейшем случае ячейка флэш-памяти хранит 1 бит информации и состоит из 1 полевого транзистора со специальной электрически изолированной областью («плавающим» затвором floating gate), способной хранить заряд многие годы. Наличие или отсутствие заряда кодирует 1 бит информации.

При записи заряд помещается на плавающий затвор одним из 2 способов (зависит от типа ячейки): методом инжекции «горячих» электронов или методом туннелирования электронов. Стирание содержимого ячейки (снятие заряда с «плавающего» затвора) производится методом туннелирования.

Как правило, наличие заряда на транзисторе понимается как логический «0», а его отсутствие как логическая «1». Современная флэш-память обычно изготавливается по 0,13- и 0,18-микронной технологии.

Можно выделить несколько **основных типов** флэш-памяти.

CompactFlash (CF) выпускается двух типов CF type I и CF type II. Этот стандарт остается сегодня наиболее универсальным и перспективным, несмотря на больший по сравнению с другими картами размер (42 x 36 x 4 мм).

SmartMedia дешевая и ультратонкая флэш--карта. Невысокая цена за счет предельно простой конструкции, но не высокая защищенность информации от случайного стирания.

Multimedia Card (MMC) основное достоинство этого типа заключается в миниатюрности и низком энергопотреблении, но при этом достаточно низкая скорость чтения и записи. Используется в мобильных телефонах и других миниатюрных устройствах.

SecureDigital (Sd) ее размер немногим больше MMC, но скорость чтения и емкость значительно выше. В связи с этим карта стоит дороже. SO и MMC обратно совместимы, то есть карты MMC можно вставить и использовать в разъеме для карт SO, но не наоборот.

MemoryStick разработка фирмы Sony Размер 24 x 32 x 1,4 (2,1) мм, довольно высокая защита информации, скорость чтения и записи сопоставимы с SecureOigital (SO), но емкость невысокая.

11. ОПЕРАТИВНАЯ ПАМЯТЬ. ПЗУ. СТРУКТУРА ЗАПИСИ ДАННЫХ

ОЗУ бывает двух типов: статическое и динамическое.

В *статическом* ОЗУ, конструируемом с помощью D-триггеров, информация сохраняется до тех пор, пока к нему подается питание. Доступ к информации осуществляется за несколько нс. В связи с этим статическое ОЗУ используют в качестве кэш-памяти.

Динамическое ОЗУ это набор ячеек, каждая из которых содержит транзистор и конденсатор. Нули и единицы (биты) информации соответствуют заряженным и разряженным конденсаторам. Для предотвращения исчезновения информации (так как электрический заряд быстро исчезает) все конденсаторы должны каждые несколько миллисекунд обновлятся. Процесс обновления обеспечивается внешней по отношению к ОЗУ логической схемой. Динамическое ОЗУ имеет большой объем, хотя время доступа к нему составляет 10ки нс.

Динамическое ОЗУ, требующее один транзистор и один конденсатор на бит (статическое ОЗУ требует не менее шести транзисторов на бит), имеет очень высокую плотность записи. Сочетание основной памяти, реализованной на динамическом ОЗУ с кэш-памятью, реализованной на статическом ОЗУ, позволяет использовать достоинства обоих типов ОЗУ.

Для динамических ОЗУ все еше используется тип памяти FPM (Fast Page Mode) быстрый постраничный режим, представляющий собой матрицу битов. Аппаратная поддержка такого режима обеспечивает поиск данных вначале по адресу строки, затем по адресу столбца.

Второй тип, применяемый для динамического ОЗУ EDO (Extended Date Output) память с расширенными возможностями вывода. Обеспечивает обращение к памяти еще до завершения предыдущего обращения. Такой конвейерный режим не ускоряет доступ к памяти, однако увеличивает пропускную способность вывода.

Очень часто данные следует сохранить даже при выключении питании. Это привело к созданию ПЗУ – постоянных запоминающих устройств, не позволяющих изменять и стирать хранимую в них информацию. Запись в ПЗУ производится во время ero изготовления.

В отличие от обычных ПЗУ, **программируемые** ПЗУ позволяют осушествлять их программирование в условиях эксплуатации, например, пережигая крошечные плавкие перемычки в нужных строках и столбцах и прилагая высокое напряжение к определенному выводу микросхемы.

Существует и стираемое программируемое ПЗУ, которое можно как программировать во время эксплуатации, так и удалять из него информацию. Если на кварцевое окно в данном ПЗУ воздействовать сильным ультрафиолетовым светом в течение 15 минут, то все биты устанавливаются в единицу.

12. УПРАВЛЕНИЕ ПАМЯТЬЮ

Управление памятью, как правило, требует решения двух задач:

- распределение памяти между процессами,
- защита памяти от несанкционированного доступа.

Для решения этих задач рассмотрим различные пространства адресов и их взаимное отображение.

Пространство имен в программе представляет собой совокупность идентификаторов, которые программа использует для обращения к тем или иным областям памяти.

Логическое пространство адресов представляет собой множество числовых значений, которые могут быть использованы для доступа к памяти в машинных командах. Мощность этого множества определяется архитектурой конкретной вычислительной системы.

Физическое пространство адресов ограничивается объемом реальной памяти конкретного компьютера.

Преобразование пространства имен в логическое пространство адресов выполняется на этапе подготовки исполняемого модуля и его загрузки в память перед выполнением.

При компиляции программы отдельным символьным именам ставятся в соответствие логические адреса. Эти адреса имеют вид смещения относительно некоторого базового адреса. Значение базового адреса остается неопределенным. Оставшиеся символьные имена преобразуются в так называемые внешние ссылки, которые должны быть разрешены на последующих этапах обработки программы в частности на этапе линкования. Наконец, на этапе загрузки программ осуществляется настройка базовых адресов. Полученные адреса, тем не менее, остаются логическими.

При преобразовании логических адресов в физические возможны два случая:

1) Пространство физических адресов больше пространства логических. В этом случае для эффективного использования всей физической памяти программы должны использовать специальные механизмы расширения памяти.

2) Пространство логических адресов больше пространства физических. В этом случае возможности преобразования значительно шире и основаны на концепции виртуальной памяти – на диске создается специальный файл подкачки (файл виртуальной памяти), разбитый на страницы. Подлежащая распределению оперативная память также разбивается на страницы того же размера. Логический адрес интерпретируется как «номер виртуальной страницы + смещение в странице». Некоторые из логических страниц загружаются в страницы ОП. ОС ведет специальную таблицу соответствия логических и физических страниц. При любом обращении к логическим адресам выполняется проверка нахождения этой страницы в оперативной памяти. Если страница отсутствует, происходит ее загрузка.

Различают следующие три типа исполняемых модулей в соответствии с механизмом использования памяти:

- простые (линейные);
- оверлейные (с перекрытием);
- динамические.

В простых модулях каждая процедура получает свой блок логических адресов, Не пересекающийся с блоками адресов других процедур.

Альтернативой простой схеме является оверлейная структура. При использовании такой схемы выделяются два или более модулей, которые не обращаются друг к другу, и для них указывается единая точка перекрытия (точка оверлея). Все перекрывающиеся модули загружаются в память начиная с этой точки. Это позволяет уменьшить суммарный объем загруженных модулей, но вынуждает тратить время на подзагрузку перекрывающихся частей.

13. МОДЕЛЬ КОНСИСТЕНТНОСТИ ПАМЯТИ

Предположим, что мы разрешаем многим вычислительным узлам компьютера иметь одновременный доступ к разделяемым данным на чтение/запись. Так как узлы могут писать данные параллельно, то для консистентности(согласованности) данных необходимо контролировать доступ к ним.

Модель консистентности представляет собой некоторый договор между программами и памятью, в котором указывается, что при соблюдении программами определенных правил работа модуля памяти будет корректной, если же требования к программе будут нарушены, то память не гарантирует правильность выполнения операций чтения/записи.

Основные модели консистентности:

Строгая консистентность (операция чтения ячейки памяти с номером X должна возвращать значение, записанное в нее последней операцией. предполагает наличие в системе понятия абсолютного времени для определения наиболее последней операции записи)

Последовательная консистентность (Результат выполнения должен быть тот же, как если бы операторы каждого процесса были расположены в порядке, задаваемом программно, т.е. при параллельном выполнении все процессы должны видеть одну и ту же последовательность записей в память. Результат выполнения проги может не совпадать с предыдущим)

Причинная консистентность (более слабая по сравнению с последовательной, ибо не всегда требует, чтобы все процессоры видели одну и ту же последовательность записей в память. Последовательность записей в память, которые потенциально причинно зависимы, должны одинаково наблюдаться всеми процессами)

*Процессорная консистентность (*требует когерентности памяти. Для каждой переменной X имеется общее согласие относительно порядка, в котором процессоры модифицируют переменную)

Слабая консистентность (Правила: а) доступ к синхронизированным переменным осуществляется с помощью модели послед. Конс. Б) доступ к синхронизированным переменным запрещен или приостановлен до тех пор, пока не выполнены предыдущие операции записи. В) доступ к данным запрещен, пока не выполнены все предыдущие обращения к синхронизированным переменным)

Консистентность по выходу (требования: а) до выполнены обращения к общей переменной должны быть выполнены все предыдущие захваты синхронизированных переменных данным процессором. Б) перед освобождением синхронизированной переменной должны быть закончены все операции чтения/записи, выполнявшиеся процессором ранее. В) реализация операции захвата/освобождения синхр. Переменной должны удовлетворять требованиям процессорной конс)

Консистентность по входу (Требования: а)Процесс не может захватить синхронизационную переменную до того, пока не обновлены все переменные этого процесса, охраняемые захватываемой синхронизационной переменной; б)Процесс не может захватить синхронизационную переменную в монопольном режиме (для модификации охраняемых данных), пока другой процесс, владеющий этой переменной (даже в немонопольном режиме), не освободит ее; в)Если какой-то процесс захватил синхронизационную переменную в монопольном режиме, то ни один процесс не сможет ее захватить даже в немонопольном режиме до тех пор, пока первый процесс не освободит эту переменную и будут обновлены текущие значения охраняемых переменных в процессе, запрашивающем синхронизационную переменную.)

14. ГРАФИЧЕСКИЙ ПРОЦЕССОР

Графический процессор (англ. *graphics processing unit, GPU*) — отдельное устройство персонального компьютера или игровой приставки, выполняющее графический рендеринг. Современные графические процессоры очень эффективно обрабатывают и отображают компьютерную графику, благодаря специализированной конвейерной архитектуре они намного эффективнее в обработке графической информации, чем типичный центральный процессор.

Графический процессор в современных видеоадаптерах применяется в качестве ускорителя трёхмерной графики, однако его можно использовать в некоторых случаях и для вычислений (GPGPU). Отличительными особенностями по сравнению с ЦП являются:
архитектура, максимально нацеленная на увеличение скорости расчёта текстур и сложных графических объектов;
ограниченный набор команд..

Высокая вычислительная мощность GPU объясняется особенностями архитектуры. Если современные CPU содержат несколько ядер (на большинстве современных систем от 2 до 4х, 2011 г.), графический процессор изначально создавался как многоядерная структура, в которой количество ядер измеряется сотнями. Разница в архитектуре обусловливает и разницу в принципах работы. Если архитектура CPU предполагает последовательную обработку информации, то GPU исторически предназначался для обработки компьютерной графики, поэтому рассчитан на массивно параллельные вычисления.[1]

Каждая из этих двух архитектур имеет свои достоинства. CPU лучше работает с последовательными задачами. При большом объеме обрабатываемой информации очевидное преимущество имеет GPU. Условие только одно – в задаче должен наблюдаться параллелизм.

CUDA— программно-аппаратная архитектура параллельных вычислений, которая позволяет существенно увеличить вычислительную производительность благодаря использованию графических процессоров фирмы Nvidia.

CUDA SDK позволяет программистам реализовывать на специальном упрощённом диалекте языка программирования Си алгоритмы, выполнимые на графических процессорах Nvidia, и включать специальные функции в текст программы на Си. Архитектура CUDA даёт разработчику возможность по своему усмотрению организовывать доступ к набору инструкций графического ускорителя и управлять его памятью.

Перечислим **основные характеристики** CUDA:

- унифицированное программно-аппаратное решение для параллельных вычислений на видеочипах Nvidia;
- большой набор поддерживаемых решений, от мобильных до мультичиповых
- стандартный язык программирования Си;

- стандартные библиотеки численного анализа FFT (быстрое преобразование Фурье) и BLAS (линейная алгебра);
- оптимизированный обмен данными между CPU и GPU;
- взаимодействие с графическими API OpenGL и DirectX;
- поддержка 32- и 64-битных операционных систем: Windows XP, Windows Vista, Linux и MacOS X;
- возможность разработки на низком уровне.

Преимущества CUDA перед традиционным подходом к GPGPU вычислениям:

- интерфейс программирования приложений CUDA основан на стандартном языке программирования Си с расширениями, что упрощает процесс изучения и внедрения архитектуры CUDA;
- CUDA обеспечивает доступ к разделяемой между потоками памяти размером в 16 Кб на мультипроцессор, которая может быть использована для организации кэша с широкой полосой пропускания, по сравнению с текстурными выборками;
- более эффективная передача данных между системной и видеопамятью
- отсутствие необходимости в графических API с избыточностью и накладными расходами;
- линейная адресация памяти, и gather и scatter, возможность записи по произвольным адресам;
- аппаратная поддержка целочисленных и битовых операций.

Основные **ограничения** CUDA:

- отсутствие поддержки рекурсии для выполняемых функций;
- минимальная ширина блока в 32 потока;
- закрытая архитектура CUDA, принадлежащая Nvidia.

15. КОМПЬЮТЕРЫ В РЕЖИМЕ УПРАВЛЕНИЯ ТЕХНОЛОГИЧЕСКИМ ПРОЦЕССОМ

Для автоматизации управления сложным производственным или технологическим процессом в контур управления включают компьютер, т. с. управляющую вычислительную машину (УВМ). Наиболее часто в качестве управляющей ЭВМ используют цифровую ЭВМ благодаря следующим ее качествам: наличию больших и высоконадежных ЗУ различного типа; возможности решения на них сложных вычислительных и логических задач; гибкости (за счет программы);надежности и быстродействию;

В общем случае система автоматического управления с УВМ определяет собой замкнутый контур.Здесь

x_1 , x_2 , .., x_n - измеряемые параметры:

•• нерегулируемые (характеристики исходного продукта);

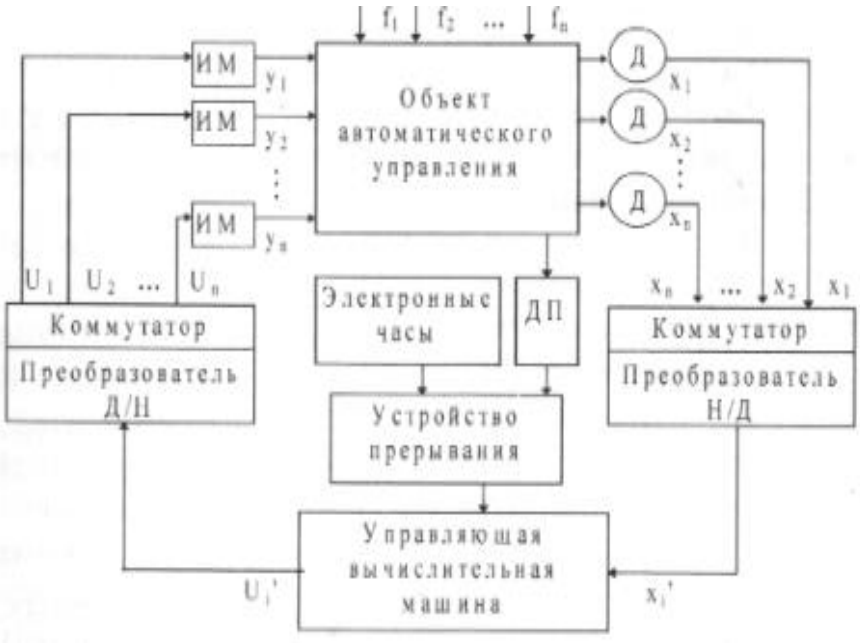
•• выходные параметры, характеристики качества продукции;

•• выходные параметры, по которым непосредственно или путем расчета определяется эффективность производственных процессов (производительность, экономичность), или ограничения, наложенные на условия его протекания;

•y_1 , y_2 , .., y_n - регулируемые параметры, которые могут изменяться исполнительными механизмами (ИМ) - регуляторами и оператором;

f_1 , f_2 , .., f_n нерегулируемые и неизмеряемые параметры (например, химическийсостав сырья).

На вход УВМ от датчика Д (термопар, расходомеров) идет измерительная информация о текущем значении параметров x_1 , x_2 , .., x_n. Согласно алгоритму управления УВМ определяет величину управляющих воздействий U_1 , U_2 , .., U_n которые необходимо приложить к ИМ для изменения регулируемых параметров y_1 , y_2 , .., y_n с тем, чтобы управляющий процесс протекал оптимально. Измерительные датчики вырабатывают непрерывный сигнал (напряжение, ток, угол поворота), а ЦВМ работает в дискретной форме, поэтому 2 раза идет преобразование из непрерывной формы в дискретную (Н/Д) и наоборот (Д/Н).



Для уменьшения оборудования преобразователи Н/Д и Д/Н выполнены многоканальными. Посредством коммутатора преобразователь поочередно подключается к каждому датчику и осуществляется преобразование Н/Д. Поступившее управляющее воздействие U сохраняется в цепи до выработки следующего управляющего воздействия УВМ.

Теперь наибольшее распространение получил синхронный принцип связи УВМ с объектом, при котором процесс управления разбивается на циклы равной продолжительности тактирующими импульсами электронных часов.Цикл начинается с приходом тактирующего импульса на устройство прерывания. В начале каждого цикла проводятся последовательный опрос и преобразование в цифровую форму сигналов датчиков.После выработки управляющих воздействий Ui и преобразовании их в непрерывную форму УВМ останавливается до прихода нового тактирующею импульса или выполняет какую-нибудь вспомогательную работу.

Для установления более тесной связи объекта с УВМ используют асинхронный принцип связи с объектом.

В некоторых системах применяют комбинированный способ - синхронизацию "плюс" датчики аварийного состояния, переводящие УВМ на режиме аварийной работы. В замкнутом контуре (см. рис. 1.12) УВМ прямо воздействует па ИМ, непосредственно управляя производственным процессом. Это режим прямого цифрового управления. УВМ в разомкнутой цепи используется:в системах автоматического программного управления;в системах, где УВМ выполняет функции советчика.

В первом случае уменьшается объем первоначально вводимой информации в УВМ для расчета оптимальных настроек регуляторов. Детальный расчет программы управления с заданной точностью производится самим вычислительным устройством, которое в соответствии с этой программой вырабатывает необходимые управляющие воздействия.В режиме советчика УВМ обрабатывает измерительную информацию е объекта и рассчитывает управляющие воздействия для оптимизации процесса. Эта информация служит рекомендацией оператору, управляющему процессом.

16. CISC- И RISC-АРХИТЕКТУРЫ

CISC-архитектура

Двумя основными архитектурами набора команд, используемыми компьютерной промышленностью на современном этапе развития вычислительной техники, являются архитектуры CISC и RISC. Основоположником CISC-архитектуры - архитектуры с полным набором команд (CISC - Complete Instruction Set Computer) можно считать фирму IBM с ее базовой архитектурой IBM/360, ядро которой используется с 1964 г. и дошло до наших дней, например, в таких современных мейнфреймах, как IBM ES/9000.

Лидером в разработке микропроцессоров с полным набором команд считается компания Intel с микропроцессорами X86 и Pentium. Простота архитектуры RISC-процсссора обеспечивает его компактность, практическое отсутствие проблем с охлаждением кристалла, чего нет в процессорах фирмы Intel, упорно придерживающейся пути развития архитектуры CISC. Формирование стратегии CISC-архитектуры произошло за счет технологической возможности перенесения "центра тяжести" обработки данных с программного уровня системы на аппаратный, так как основной путь повышения эффективности для CISC-компьютера виделся, в первую очередь, в упрощении компиляторов и минимизации исполняемого модуля. На сегодняшний день CISC-процессоры почти монопольно занимают на компьютерном рынке сектор персональных компьютеров, однако RISC-процессорам нет равных в секторе высокопроизводительных серверов и рабочих станций.

CISC-архитектура	**RISC-архитектура**
Многобайтовые команды	Однобайтовые команды
Малое количество регистров	Большое количество регистров
Сложные команды	Простые команды
Одна или менее команд за один цикл процессора	Несколько команд за один цикл процессора
Традиционно одно исполнительное устройство	Несколько исполнительных устройств

Одним из важных преимуществ RISC-архитектуры является высокая скорость арифметических вычислений. Высокая скорость выполнения арифметических операций в сочетании с высокой точностью вычислений обеспечивает RISC-процессорам безусловное лидерство по быстродействию в сравнении с CISC-процессорами.

Другой особенностью RISC-процессоров является комплекс средств, обеспечивающих безостановочную работу арифметических устройств: механизм динамического прогнозирования ветвлений, большое количество оперативных регистров, многоуровневая встроенная кэш-память.

Организация регистровой структуры - основное достоинство и основная проблема RISC. Практически любая реализация RISC-архитектуры использует трехместные операции обработки, в которых результат и два операнда имеют самостоятельную адресацию - Rl : = R2, R3. Это позволяет без существенных затрат времени выбрать операнды из адресуемых оперативных регистров и записать в регистр результат операции. Кроме того, трехместные операции дают компилятору большую гибкость по сравнению с типовыми двухместными операциями формата "регистр - память" архитектуры CISC. В сочетании с быстродействующей арифметикой RISC-операции типа "регистр - регистр" становятся очень мощным средством повышения производительности процессора.

Вместе с тем опора на регистры является ахиллесовой пятой RISC-архитектуры. Проблема в том, что в процессе выполнения задачи RISC-система неоднократно вынуждена обновлять содержимое регистров процессора, причем за минимальное время, чтобы не вызывать длительных простоев арифметического устройства. Для CISC-систем подобной проблемы не существует, поскольку модификация регистров может происходить на фоне обработки команд формата "память - память".

RISC-архитектура

Суть ее состоит в выделении наиболее употребительных операций и создании архитектуры, приспособленной для их быстрой реализации. Это позволило в условиях ограниченных ресурсов разработать компьютеры с высокой пропускной способностью.

4 основных принципа RISC-архитектуры:

1. каждая команда независимо от ее типа выполняется за один машинный цикл, длительность которого должна быть максимально короткой;
2. все команды должны иметь одинаковую длину и использовать минимум адресных форматов, что резко упрощает логику центрального управления процессором;
3. обращение к памяти происходит только при выполнении операций записи и чтения, вся обработка данных осуществляется исключительно в регистровой структуре процессора;
4. система команд должна обеспечивать поддержку языка высокого уровня.

17. КОМПЬЮТЕРЫ СО СТЕКОВОЙ АРХИТЕКТУРОЙ

На сегодняшний день наибольшее распространение получили следующие структуры команд: одноадресные (1А), двухадресные (2А), трехадресные (ЗА), безадресные (БА), команды с большой длиной слова (VLIW - БДС).

Причем операнд может указываться как адресом, так и непосредственно в структуре команды.

В случае БА-команд операнды выбираются и результаты помещаются в стек (магазин, гнездо). Типичными первыми представителями БА-компьютеров являются KDF-9 и МВК "Эльбрус". Их характерной особенностью является наличие стековой памяти.

Стек - это область оперативной памяти, которая используется для временного хранения данных и операций. Доступ к элементам стека осуществляется по принципу FILO (first in, last out) - первым вошел, последним вышел. Кроме того, доступ к элементам стека осуществляется только через его вершину, т е. пользователю "виден'’ лишь тот элемент, который помещен в стек последним.

Рассмотрим функционирование процессора со стековой организацией памяти.

При выполнении различных вычислительных процедур процессор использует либо новые операнды, до сих пор не выбиравшиеся из памяти компьютера, либо операнды, употреблявшиеся в предыдущих операциях. В процессорах с классической структурой обращение к любому операнду (1 А-ЭВМ) требует цикла памяти.

Стековая память представляет собой набор из n регистров, каждый из которых способен хранить одно машинное слово. Одноименные разряды регистров $Р_1$, Р2, ..., $Р_п$ соединены между собой цепями сдвига. Поэтому весь набор регистров может рассматриваться как группа n-разрядных сдвигающих регистров, составленных из одноименных разрядов регистров $Р_1$, Р2, ..., $Р_п$. Информация в стеке может продвигаться между регистрами вверх и вниз.

Движение вниз: ($Р_1$) —> Р2, (Р2) —> Р3...., а $Р_1$ заполняется данными из главной памяти.

Движение вверх: ($Р_n$) —> $Р_{n-1}$ ($Р_{n-1}$) —> $Р_{n-2}$, а $Р_n$ заполняется нулями.

Регистры $Р_1$ и $Р_2$ связаны с АЛУ, образуя два операнда для выполнения операции. Результат операции записывается в $Р_1$. Следовательно, АЛУ выполняет операцию

($Р_1$)®($Р_2$) —>$Р_1$

Одновременно с выполнением арифметической операции (АО) осуществляется продвижение операндов вверх, не затрагивая $Р_1$, т. с. (Р3) -» $Р_2$, ($Р_4$) -> $Р_3$ и т. д.

Таким образом, АО используют подразумеваемые адреса, что уменьшает длину команды. В принципе, в команде достаточно иметь только поле, определяющее код операции. Поэтому компьютеры со стековой памятью называют безадресными. В то же время команды, осуществляющие вызов или

запоминание информации из главной памяти, требуют указания адреса операнда. Поэтому в ЭВМ со стековой памятью используются команды переменной длины. Например, в KDF-9 команды АО - однослоговые, команды обращения к памяти и передач управления - трехслоговые, остальные - двуслоговые.

Команды располагаются в памяти в виде непрерывного массива слогов независимо от границ ячеек памяти. Это позволяет за один цикл обращения к памяти вызвать несколько команд.

Для эффективного использования возможностей такой памяти в ЭВМ вводятся спецкоманды:

- дублирование ~ (P_1) —> P_2, (P_2) —>P_3.... и т. д., а (P_1) остается при этом неизменным;
- реверсирование - (P_1) —> P_2, а (P_2) —> P_1. что удобно для выполнения некоторых операций.

Главное преимущество использования магазинной памяти состоит в том, что при переходе к подпрограммам (ПП) или в случае прерывания нет необходимости в специальных действиях по сохранению содержимого арифметических регистров в памяти. Новая программа может немедленно начать работу. При введении в стековую память новой информации данные, соответствующие предыдущей программе, автоматически продвигаются вниз. Они возвращаются обратно, когда новая программа закончит вычисления.

Наряду с указанными преимуществами стековой памяти отметим также:

- уменьшение количества обращений к памяти;
- упрощение способа обращения к ПП и обработки прерываний.

Недостатки стековой организации памяти:

- большое число регистров с быстрым доступом;
- необходимость в дополнительном оборудовании, чтобы следить за переполнением стековой памяти, ибо число регистров памяти конечно;
- приспособленность главным образом для решения научных задач и в меньшей степени для систем обработки данных или управления технологическими процессами.

18. НЕЙРОКОМПЬЮТЕРЫ

Нейрокомпьютинг - это научное направление, занимающееся разработкой вычислительных систем шестого поколения - нейрокомпьютеров, которые состоят из большого числа параллельно работающих простых вычислительных элементов (нейронов). Элементы связаны между собой, образуя нейронную сеть. Они выполняют единообразные вычислительные действия и не требуют внешнего управления. Большое число параллельно работающих вычислительных элементов обеспечивают высокое быстродействие.

Нейрокомпьютеры позволяют с высокой эффективностью решать целый ряд интеллектуальных задач. Это задачи распознавания образов, адаптивного управления, прогнозирования, диагностики и т.д.

Нейрокомпьютеры отличаются от ЭВМ предыдущих поколений не просто большими возможностями. Принципиально меняется способ использования машины. Место программирования занимает обучение, нейрокомпьютер учится решать задачи.

Обучение - корректировка весов связей, в результате которой каждое входное воздействие приводит к формированию соответствующего выходного сигнала. После обучения сеть может применять полученные навыки к новым входным сигналам. При переходе от программирования к обучению повышается эффективность решения интеллектуальных задач.

Вычисления в нейронных сетях существенно отличаются от традиционных, в силу высокой параллеленности их можно рассматривать как коллективное явление. В нейронной сети нет локальных областей, в которых запоминается конкретная информация. Вся информация запоминается во всей сети.

Отличия нейрокомпьютеров от вычислительных устройств предыдущих поколений:

- параллельная работа очень большого числа простых вычислительных устройств обеспечивает огромное быстродействие;
- нейронная сеть способна к обучению, которое осуществляется путем настройки параметров сети;
- высокая помехо- и отказоустойчивость нейронных сетей;
- простое строение отдельных нейронов позволяет использовать новые физические принципы обработки информации для аппаратных реализаций нейронных сетей.

Нейронные сети находят свое применение в системах распознавания образов, обработки сигналов, предсказания и диагностики, в робототехнических и бортовых системах. Решение слабо зависит от неисправности отдельного нейрона.

Разработки в области нейрокомпьютеров поддерживаются целым рядом международных и национальных программ - от финансовых прогнозов до экспертизы.

Разработки в области нейрокомпьютинга ведутся по следующим направлениям:

- разработка нейроалгоритмов;
- создание специализированного ПО для моделирования нейронных сетей;
- разработка специализированных процессорных плат для имитации нейросетей;
- электронные реализации нейронных сетей;
- оптоэлектронные реализации нейронных сетей.

В основу искусственных нейронных сетей положены следующие черты живых нейронных сетей, позволяющие им хорошо справляться с нерегулярными задачами:

- простой обрабатывающий элемент - нейрон;
- очень большое число нейронов участвует в обработке информации;
- один нейрон связан с большим числом других нейронов (глобальные связи);
- изменяющиеся веса связей между нейронами;
- массированная параллельность обработки информации.

Прототипом для создания нейрона послужил биологический нейрон головного мозга. Биологический нейрон имеет тело, совокупность отростков - дендридов, по которым в нейрон поступают входные сигналы, и отросток - аксон, передающий выходной сигнал нейрона другим клеткам. Точка соединения дендрида и аксона называется синапсом. Упрощенно функционирование нейрона можно представить следующим образом:

1. нейрон получает от дендридов набор (вектор) входных сигналов;
2. в теле нейрона оценивается суммарное значение входных сигналов. Однако входы нейрона неравнозначны. Каждый вход характеризуется некоторым весовым коэффициентом, определяющим важность поступающей по нему информации. Таким образом, нейрон не просто суммирует значения входных сигналов, а вычисляет скалярное произведение вектора входных сигналов и вектора весовых коэффициентов;

3. нейрон формирует выходной сигнал, интенсивность которого зависит от значения вычисленного скалярного произведения. Если оно не превышает некоторого заданного порога, то выходной сигнал не формируется вовсе - нейрон "не срабатывает";
4. выходной сигнал поступает на аксон и передается дендридам других нейронов.

Нейронная сеть представляет собой совокупность большого числа сравнительно простых элементов - нейронов, топология соединений которых зависит от типа сети. Чтобы создать нейронную сеть для решения какой-либо конкретной задачи, мы должны выбрать, каким образом следует соединять нейроны друг с другом, и соответствующим образом подобрать значения весовых параметров на этих связях. Может ли влиять один элемент на другой, зависит от установленных соединений. Вес соединения определяет силу влияния.

Задачи, решаемые на основе нейронных сетей

В литературе встречается значительное число признаков, которыми должна обладать задача, чтобы применение НС было оправдано и НС могла бы ее решить:

- отсутствует алгоритм или не известны принципы решения задач, но накоплено достаточное число примеров;
- проблема характеризуется большими объемами входной информации;
- данные неполны или избыточны, зашумлены, частично противоречивы.

Таким образом, НС хорошо подходят для распознавания образов и решения задач классификации, оптимизации и прогнозирования. Ниже приведен перечень возможных промышленных применений нейронных сетей.

Банки и страховые компании:

- автоматическое считывание чеков и финансовых документов;
- проверка достоверности подписей;
- оценка риска для займов;
- прогнозирование изменений экономических показателей.

Административное обслуживание:

- автоматическое считывание документов;
- автоматическое распознавание штриховых кодов.

Нефтяная и химическая промышленность:

- анализ геологической информации;
- идентификация неисправностей оборудования;
- анализ составов примесей;
- управление процессами.

Военная промышленность и аэронавтика:

- автоматическое пилотирование.

Служба безопасности:

- распознавание лиц, голосов, отпечатков пальцев.

Биомедицинская промышленность:

- анализ рентгенограмм;
- обнаружение отклонений в ЭКГ.

Способы реализации нейронных сетей

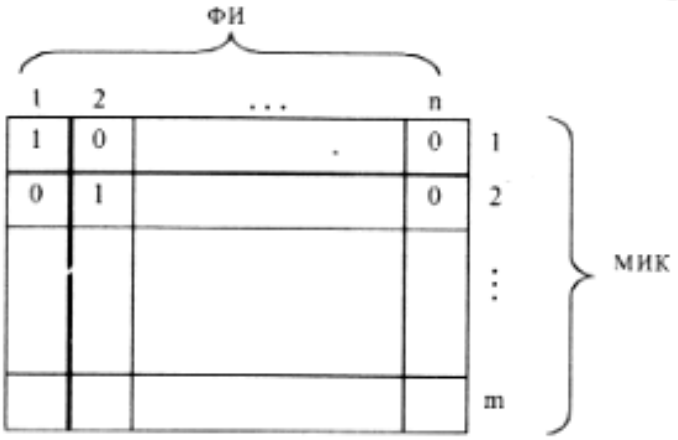
Нейронные сети могут быть реализованы двумя путями: первый - это программная модель НС, второй - аппаратная.

Большинство сегодняшних нейрокомпьютеров представляют собой просто персональный компьютер или рабочую станцию, в состав которых входит дополнительная нейроплата. Такие системы имеют бесспорное право на существование, поскольку их возможностей вполне достаточно для разработки новых алгоритмов и решения большого числа прикладных задач методами нейроматематики.

Однако наибольший интерес представляют специализированные нейрокомпьютеры, непосредственно реализующие принципы НС.

19. ПРОЦЕССОРЫ С МИКРОПРОГРАММНЫМ УПРАВЛЕНИЕМ.

Существуют два вида микропрограммного управления: горизонтальное и вертикальное. При горизонтальном - каждому разряду микрокоманды (МИК) соответствует определенная микрооперация (МИО), выполняемая независимо от содержания других разрядов. Микропрограмма может быть представлена в виде матрицы n x m, где n- число функциональных импульсов (ФИ), m - количество МИК, т. е. строка соответствует одной МИК, а столбец - одной МИО.



Примерные значения разрядов МИК приведены на рис. 2.5.

1	2	3	4	5	6	7	8	9	...

1 - гашение сумматора; 2 - гашение указателя переполнения; 3 - обратный код сумматора; 4-гашение регистра множителя частного; 5 - инвертирование знака; 6 - сдвиг содержимого сумматора влево; 7 - сдвиг содержимого сумматора вправо; 8 - увеличение содержимого сумматора на 1; 9 - чтение из ЗУ в сумматор;...

Наличие "1" в пересечении какой-либо строки и столбца означает посылку ФИ в данную МИК, а наличие "0" - его отсутствие.

Размещение "1" в нескольких разрядах МИК означает выполнение нескольких МИО одновременно. Конечно, возбуждаемые МИО должны быть совместимы.

Пусть, например, разряды 9-разрядной МИК принимают следующие значения: 001001101. Тогда, если заданные разряды соответствуют семантике, указанной на рис. 2.5, то МИО, определяемые разрядами 9, 7 и 6, несовместимы.

Для расширения возможностей МИК иногда используют многотактный принцип исполнения МИК. При этом каждому разряду присваивается номер такта, в котором выполняется соответствующая ему МИО, т. с. здесь все совместимые МИО имеют один номер такта. Все остальные такты нумеруются в порядке их естественного выполнения. Однако универсальную нумерацию МИО в МИК указать затруднительно.

Достоинства горизонтального микропрограммирования:

- возможность одновременного выполнения нескольких МИО;
- простота формирования ФИ (без схем дешифрации).

Недостатки:

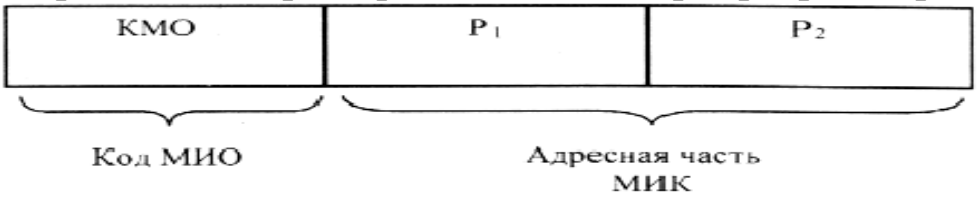
- большая длина МИК, так как число ФИ в современных компьютерах достигает нескольких сот, и соответственно большой объем ЗУ для хранения МИК;
- из-за ограничений совместимости операций, а также из-за последовательного характера выполнения алгоритмов операций лишь небольшая часть разрядов МИК будет содержать "1". В основном матрица будет состоять из нулей. Неэффективное использование ЗУ привело к малому распространению горизонтального микропро1раммирования.

Вертикальное программирование

При вертикальном микропрограммировании каждая МИО определяется не состоянием одного разряда, а двоичным кодом, содержащимся в определенном поле МИК. Микрокоманда несколько напоминает формат обычных команд. Отличие состоит в том, что:

- выполняется более элементарное действие МИО вместо операции;
- адресная часть (в большинстве случаев) определяет не ячейку памяти, а операционный регистр процессора.

Формат МИК при вертикальном микропрограммировании:



Поля P_1 и P_2 в адресной части МИК указывают двоичные номера операционных регистров, содержимое которых участвует в одной операции. Одно из полей является одновременно и адресом результата. Таким образом, реализация арифметической или логической МИО, указанной в данной МИК, может быть выражена формулой

(P_1) ® (P_2) -> P_1 или (P_2) ->P_1 где ® - символ МИО.

Для МИК обращения к памяти поле P_1 указывает регистр, куда принимается информация, а P_2 регистр, содержимое которого является адресом обращения к ЗУ. Указанный формат МИК не единственный.

Каждая МИК выполняет следующие функции:

- указывает выполняемую МИО;
- указывает следующую МИО через задание "следующего адреса";
- задает продолжительность МИК;указывает дополнительные действия - контроль и т. д.

Обычно в слове МИК имеются четыре зоны, соответствующие указанным функциям. Вообще говоря, некоторые из зон могут указываться неявно, например выбор очередной МИК может осуществляться из следующей ячейки, продолжительность МИК может быть определена одинаковой для всех МИК и т.д. Большую часть МИК занимает зона микроопераций.

20. ОСНОВНЫЕ НОВАЦИИ В АРХИТЕКТУРЕ КОМПЬЮТЕРОВ

IBM 701 (1953 год), IBM 704 (1955 год): разряднопараллельная память, разрядно-параллельная арифметика. Первые компьютеры (EDSAC, EDVAC, UNIVAC) имели разряднопоследовательную память, из которой слова считывались последовательно бит за битом. наибольшую популярность получила модель IBM 704, в которой впервые была применена память на ферритовых сердечниках и аппаратное арифметическое устройство (АУ) с плавающей точкой.

IBM 709 (1958 год): независимые процессоры ввода/вывода. Процессоры первых компьютеров сами управляли вводом/выводом. Однако скорость работы самого быстрого внешнего устройства магнитной ленты была в 1000 раз меньше быстродействия процессора, поэтому во время операций ввода/вывода процессор фактически простаивал. В 1958 году к компьютеру IBM 704 присоединили шесть независимых процессоров ввода/вывода, которые могли работать параллельно с основным процессором, а сам компьютер переименовали в IBM 709.

IBM STRETCH (1961 год): опережающий просмотр вперед, расслоение памяти. Компьютер STRETCH имеет две принципиально важные особенности: опережающий просмотр вперед для выборки команд и расслоение памяти на два банка для согласования низкой скорости выборки из памяти и скорости выполнения операций.

ATLAS (1963 год): конвейер команд. Впервые конвейерный принцип выполнения команд был использован в машине ATLAS, разработанной в Манчестерском университете. Выполнение команд разбито на четыре стадии: выборка команды. вычисление адреса операнда, выборка операнда и выполнение операции. Конвейеризация позволила уменьшить время выполнения команд с 6 мкс до 1,6 мкс.

CDC 6600 (1964): независимые функциональные устройства. Фирма Control Data Corporation (CDC) при непосредственном участии одного из ее основателей, Сеймура Крэя (Seymour R. Cray),выпустила компьютер CDC 6600 первый компьютер, в котором исполъзовалось несколько независимых функциональных устройств.

CDC 7600 (1969 год): конвейерные независимые функциональные устройства (ФУ). CDC выпускает компьютер CDC7600 с восемью независимыми конвейерными функциональными устройствами сочетание параллельной и конвейерной обработки.

ILLIAC IV (1974 год): матричные процессоры. Проектировалось 256 процессорных элементов (ПЭ), включающих 4 квадранта по 64 ПЭ, Реально реализована матрица из 64 ПЭ, все элементы которой работали в синхронном режиме, выполняя в каждый момент времени одну и ту же команду, поступившую от устройства управления (УУ); но над своими данным;; ПЭ имел собственное АЛУ с полным набором команд.

CRAУ 1 (1976 год): векторно-конвейерные. Главным новшеством проекта является введение векторных команд, работающих с целыми массивами независимых данных и позволяющих эффективно использовать конвейерные функциональные устройства.

21. ВЫЧИСЛИТЕЛЬНЫЕ ПАРАДИГМЫ. МЕТОДЫ КОММУНИКАЦИЙ

Существует ряд вычислительных парадигм для структуризации работы большого количества потоков или независимых процессов. Рассмотрим наиболее употребительные из них.

1) парадигма SPMD - одна программа, несколько потоков данных. В данном случае все процессы выполняют одну и ту же программу, но над разными наборами данных, то есть производят одни и те же вычисления, но каждый в своем адресном пространстве.

2) парадигма – конвейер. Данные поступают в первый процесс, трансформируются и передаются второму процессу для чтения, и т.д.

3) фазированные вычисления. В этом случае выполняемая работа разделяется на фазы, в качестве которых, например, могут выступать повторения цикла. На каждой фазе одновременно работают несколько процессов, но каждый из них ожидает завершения остальных процессов фазы, после чего начинается новая фаза.

4) парадигма разделяй и властвуй, где каждый начавшийся процесс порождает новые процессы, которым он передает часть работы. Причем порожденные процессы могут в свою очередь породить ряд новых процессов с передачей им части работы.

5) порождающая работа (replicated worker). Организуется центральная очередь, рабочие процессы получают задачи из этой очереди и выполняют их. Если задача порождает новые задачи, они добавляются в центральную очередь. Завершая выполнение текущей задачи, рабочий процесс получает из центральной очереди следующую задачу.

Процессы, выполняющиеся на различных устройствах, могут взаимодействовать с помощью либо общих переменных, либо передачи сообщений.

В первом случае взаимодействующие процессы имеют доступ к общей памяти и взаимодействуют, считывая и записывая информацию в эту память. Один процесс, например, может записывать информацию в ячейку памяти, а второй считывать эту информацию. Отображая одну и ту же страницу в адресное пространство каждого процесса в мультипроцессоре, разделяют переменные между несколькими процессами. Затем общие переменные считываются и записываются с помощью обычных команд LOAD и STORE.

Альтернативный подход взаимодействие через передачу сообщений посредством использования примитивов send и receive. Они реализуются как системные вызовы. Один процесс отправляет сообщение, обозначая другой процесс в качестве пункта назначения. Как только второй процесс применяет примитив receive, сообщение копируется в адресное пространство получателя. Таким способом взаимодействуют процессы в мультикомпьютере, так как они не получают доступа к памяти других процессоров с помощью команд LOAD и STORE. Эти различия полностью меняют модель программирования.

Очень важный вопрос при передаче сообщений количество получателей. Простейший случай «один отправитель один получатель» называют двухточечной передачей сообщений. Отправление сообщения всем процессам – широковещательная передача, а в случае заданного (фиксированного) набора процессов мультнвещание.

22. ОРГАНИЗАЦИЯ СИСТЕМЫ ПРЕРЫВАНИЙ.

Прерывание программы - это свойство ВС при возникновении особых событий временно прекратить выполнение текущей программы и передать управление программе, специально предусмотренной для обработки данного события.

В системе с прерыванием каждое программно-независимое событие (источник прерывания) должно, если оно может повлиять на ход обработки сопровождаться сигналом, говорящим о его возникновении. Назовем эти сигналы *запросами прерывания*. Программы, затребованные запросами прерывания, назовем *прерывающими программами*.

Так как функции по сохранению и восстановлению состояния прерванной программы возлагаются на саму прерывающую программу, то последняя должна состоять из трех частей: подготовительной и восстановительной, обеспечивающих переход к нужной программе, и собственно прерывающей программы.

По окончании работы прерывающей программы переход может быть осуществлен либо к прерванной программе, либо к другой прерывающей программе.

Так как всевозможные запросы на прерывание вырабатываются независимо и асинхронно, то возможны также ситуации:

- приход запросов последовательный;
- одновременный приход нескольких запросов;
- приход запроса во время выполнения прерывающей программы.

Следовательно, должен быть организован порядок, в котором поступившие запросы удовлетворяются. Если в ВС имеются средства для обслуживания запросов в порядке присвоенного им приоритета, то такие системы прерывания называются *приоритетными*.

Система прерываний программ (СПП), как правило, выполняют следующие основные функции:

- организуют вход в прерывающую программу,
- осуществляют приоритетный выбор между запросами прерывания;
- обеспечивают возврат к прерванной программе и программное изменение приоритетов программ.

Параметры эффективности системы прерывании

Для сравнения различных СПП используются чаще всего следующие параметры их функционирования:

время реакции - время между появлением запроса на прерывание и началом выполнения первой команды прерывающей программы (t_p). Так как t_p зависит от приоритета программы, то для характеристики системы используют время реакции для программы с наивысшим приоритетом;

время обслуживания прерывания - разность между полным временем выполнения прерывающей программы ($t_{пр}$) и временем выполнения всех полезных команд ($t_п$), т. е. $t_{обс} = t_З + t_в$

удельный вес прерывающих программ $\eta = t_п / t_{пр}$

глубина прерывания - максимальное число программ, которые могут прерывать друг друга. Возможны случаи с одним запросом, фиксированной глубиной прерывания и неограниченным количеством раз прерываний.

Ясно, что чем больше глубина прерывания, тем лучше можно учесть приоритетное обслуживание. Так, если при глубине прерывания n_0 пришла n_0+1-я программа, когда выполняется n-е прерывание, причем n_0+1-я программа с наивысшим приоритетом, то она будет принята к исполнению только после выполнения n_0 -й программы:

насыщение системы прерывания. Если $t_р$ или $t_п$ запроса настолько велики, что запрос окажется необслуженным к моменту прихода нового запроса от того же устройства, то возникает явление, называемое насыщением системы прерывания. В этом случае факт посылки предыдущего запроса от данного источника будет утрачен. Параметры системы должны быть так согласованы, чтобы насыщение системы не наступило.

Вход в прерывающую программу

Наиболее простыми являются три следующих способа определения допустимого момента прерывания.

Метод помеченного оператора (опорных точек). Суть метода состоит в следующем. В специальные разряды команд, после которых допускается прерывание, записывается определенный знак, разрешающий (например, состояние "1" специального бита команды) или запрещающий (например, состояние "0") прерывание. Тогда вдоль программы можно расставить все опорные точки. Здесь желательно так расставить точки прерывания, чтобы информация, находящаяся в регистрах процессора после выполнения данной команды, дальше не использовалась. Это уменьшает время обслуживания, но увеличивает время реакции.

Покомандный способ. Здесь прерывание допускается после выполнения любой команды. Способ прост в реализации. При этом $t_р$ уменьшается, а $t_{обс}$ увеличивается.

Метод быстрого реагирования. Прерывание допускается по окончании выполнения очередного такта любой команды. При этом $t_р$->min, а $t_{обс}$->max, поскольку надо запомнить и затем восстановить некоторые элементы (например счетчик тактов).

23. КОНВЕЙЕРИЗАЦИЯ. КОНФЛИКТЫ И МЕХАНИЗМЫ ИХ ОБХОДА

Принцип конвейерной обработки основан на разбиении вычислительного процесса на несколько подпроцессов, каждый из которых выполняется в отдельном устройстве, как это имеет место при выполнении сложных операций последовательно по подоперациям на промышленном конвейере, в связи с чем этот метод и называют конвейерным.

1. КОНВЕЙЕРНАЯ ОБРАБОТКА

Конвейерная обработка в общем случае основана на разделении подлежащей исполнению функции на более мелкие части, называемые *ступенями*. Производительность при этом возрастает благодаря тому, что одновременно на разных ступенях конвейера выполняются различные этапы нескольких команд. Такая обработка применяется во всех современных быстродействующих процессорах.

Выполнение типичной команды можно разделить на следующие этапы:

1) выборка команды IF (Instruction Fetch);
2) декодирование команды ID (Instruction Decode);
3) выполнение операции EX (Execute);
4) обращение к памяти MEM (Memory);
5) сохранение (запись) результата WB (Write Back).

Для конвейеризации потока команд мы разбиваем каждую из них на указанные этапы, отведя для выполнения каждого этапа один такт синхронизации, и начинаем в каждом такте выполнение новой команды.

Такая организация функционирования направлена на возможно более полную загрузку всех компонент системы. Однако:

1. не все этапы выполнения команды идентичны по времени,
2. не все команды выполняются до конца (ветвления, циклы и т. д.),
3. имеется информационная зависимость между результатами операций, в связи с чем не все устройства ВС работают на полную мощность.

При реализации конвейерной обработки возникают ситуации, которые препятствуют выполнению очередной команды из потока команд в предназначенном для нее такте. Такие ситуации называются **конфликтными.**

Существуют три класса конфликтов:

1. Структурные, которые возникают из-за конфликтов по ресурсам, когда аппаратные средства не могут поддерживать все возможные комбинации команд в режиме одновременного выполнения с совмещением.

2. Конфликты по данным, возникающие в случае, когда выполнение одной команды зависит от результата выполнения предыдущей команды.

3. Конфликты по управлению, которые возникают при конвейеризации команд переходов и других команд, которые изменяют значение счетчика команд.

Конфликты в конвейере приводят к необходимости приостановки выполнения команд (pipeline stall).

2. КЛАССИФИКАЦИЯ КОНФЛИКТОВ ПО ДАННЫМ

Конфликт по данным возникает всегда, если имеет место зависимость между командами, и они расположены по отношению друг к другу так близко, что совмещение операций, происходящее при конвейеризации, может привести к изменению порядка обращения к операндам.

Известны три возможных конфликта по данным в зависимости от порядка операций чтения и записи. Рассмотрим две команды i и j, при этом i предшествует j. Возможны следующие конфликты:

RAW (чтение после записи) – команда j пытается прочитать операнд-источник данных прежде, чем команда i запишет туда данные. Таким образом, j может некорректно получить старое значение. Это наиболее общий тип конфликтов, способ их преодоления – механизм “обходов”.

WAR (запись после чтения) – j пытается записать результат в приемник прежде, чем он считывается оттуда командой i, так что i может некорректно получить новое значение. Этот тип конфликтов, как правило, не возникает в системах с централизованным управлением потоком команд, обеспечивающих выполнение команд в порядке их поступления, так как последующая запись всегда выполняется позже, чем предшествующее считывание. Особенно часто конфликты такого рода могут возникать в системах, допускающих выполнение команд не в порядке их расположения в программном коде.

WAW (запись после записи) – команда j пытается записать операнд прежде, чем будет записан результат команды i, то есть записи заканчиваются в неверном порядке, оставляя в приемнике значение, записанное командой i, а не j. Этот тип конфликтов присутствует только в конвейерах, которые выполняют запись со многих ступеней (или позволяют команде выполняться даже в случае, когда предыдущая приостановлена).

24. ПРИОРИТЕТНАЯ СИСТЕМА ПРЕРЫВАНИЙ. ШЕСТИУРОВНЕНВАЯ СИСТЕМА ПРЕРЫВАНИЙ.

Аппарат приоритетов предназначен для повышения эффективности использования ресурсов ВС. Так, например, неэффективным является одновременное выполнение двух заданий, каждое из которых требует большой загрузки устройств ввода-вывода и незначительно использует центральный процессор. Приоритет задания может назначаться исходя из:

- времени ею выполнения (короткие» задания с небольшим временем выполнения имеют более высокий приоритет по сравнению с «длинными» зад.аниями, требующими значительных затрат времени);
- объема используемой оперативной памяти (задания, требующие большого объема памяти, не должны иметь одинаковый приоритет);
- интенсивности и объема использования других ресурсов БС;

срочности выполнения:;

- взаимодействия процессов;
- последствий задержКИ обработки процессов и т.д.

При программном распознавании причин прерывания прерывающая программа начинает анализ с запросов на прерывание, имеющих более высокий приоритет. При этом в процессе работы программным путем можно изменить приоритет запросов. При аппаратном распознавании причин прерывания указанные функции возлагаются на специальное оборудование.

Существует два понятия приоритета в прерывании программ.

Предположим, что никаких ограничений на время поступления запросов прерывания не накладывается. При одновременном поступлении нескольких запросов для немедленного удовлетворения может быть принят только один. Этот тип приоритета, определяющий очередность рассмотрения запросов прерывания, называют *приоритетом между запросами прерываний*.

Прерывающие программы, однако, могут иметь относительно текущей программы различную степень важности, и выполняющаяся программа не может быть прервана любым запросом. Чтобы иметь возможность прервать текущую программу принятый СПП запрос должен соответствовать программе с более высоким приоритетом, нежели выполняемая в данный момент. Этот тип приоритета определяет старшинство программ и его обычно называют *приоритетом между прерывающими программами*.

Таким образом, приоритет между запросами прерывания нужен лишь для выбора одного запроса из многих, а приоритет между прерывающими программами определяет фактический порядок выполнения программ.

Наибольшее распространение в компьютерах получили **шесть уровней прерывания**:

Прерывания ввода-вывода, идущие от каналов и периферийных устройств, сигнализируют системе о нормальном (или ненормальном) окончании операции ввода-вывода.

Прерывание при обращении к супервизору позволяет пользователю направлять работу супервизора на реализацию нужных действий (выделить дополнительную область памяти, запустить операцию ввода-вывода и т.п.).

Прерывание по программному сбою возникает в результате различного рода ошибок в программе: переполнение разрядной сетки, нарушение защиты, появление привилегированной команды в состоянии «задача» и т.п.

Внешние прерывания происходят от внешних по отношению к компьютеру объектов оператора путем нажатия определенной кнопки, от датчика времени и т.п.

Прерывание повторного пуска это средство, которое позволяет оператору или некоторому процессору вызвать выполнение требуемой программы.

Прерывание от схем контроля сигнализирует о неисправности оборудования и обеспечивает ее локализацию и исправление.

Каждый уровень прерывания может обслуживать несколько причин. Конкретная причина прерывания внутри уровня определяется программным путем по «коду прерывания) и некоторой дополнительной информации, которая запоминается ю1жды□й раз в оперативной памяти при возникновении прерывания.

В ходе вычислительного процесса в один и тот же момент может возникнуть несколько событий, вызывающих прерывания. Одновременно появившиеся запросы на прерывание удовлетворяются в заранее установленном порядке, в соответствии с их приоритетами. Такая же ситуация наблюдается и с приоритетами между прерывающими про граммами. Чаще всего прерывание от аппаратуры контроля имеет наивысший приоритет.

25. VLIW-АРХИТЕКТУРА.

Архитектура сверхдлинного командного слова (VLIW — Very Long Instruction Word). Вообще, быстродействие VLIW-процессора в большей степени зависит от компилятора, нежели от аппаратуры, поскольку здесь эффект от оптимизации последовательности операций превышает результат, возникающий от повышения частоты.

Архитектура VLIW представляет собой одну из реализаций концепции внутреннего параллелизма в микропроцессорах. Их быстродействие можно повысить двумя способами: увеличив либо тактовую частоту, либо количество операций, выполняемых за один такт. В первом случае требуется применение «быстрых» технологий и таких архитектурных решений, как глубинная конвейеризация (конвейеризация в пределах одного такта, когда в каждый момент времени задействованы все логические блоки кристалла, а не отдельные его части). Для увеличения количества выполняемых за один цикл операций необходимо на одном чипе разместить множество функциональных модулей обработки и обеспечить надежное параллельное исполнение машинных инструкций, что даст возможность включить в работу все модули одновременно. Надежность в таком контексте означает, что результаты вычислений будут правильными.

Планирование порядка вычислений — довольно трудная задача, которую приходится решать при проектировании современного процессора. В суперскалярных архитектурах для распознавания зависимостей между машинными инструкциями применяется специальное довольно сложное аппаратное решение Однако размеры такого аппаратного планировщика при увеличении количества функциональных модулей обработки возрастают в геометрической прогрессии, что, в конце концов, может занять весь кристалл процессора. На самом же деле, текущие реализации VLIW тоже далеко не всегда могут похвастаться 100% заполнением пакетов — реальная загрузка около 7 команд в такте примерно столько же, сколько и лидеры среди RISC-процессоров. При другом подходе можно передать все планирование программному обеспечению, как это делается в конструкциях с VLIW. «Умный» компилятор должен выискать в программе все инструкции, которые являются совершенно независимыми, собрать их вместе в очень длинные строки (длинные инструкции) и затем отправить на одновременное исполнение функциональными модулями, количество которых, как минимум, не меньше, чем количество операций в такой длинной команде.

Логический слой VLIW-процессора

Процессор VLIW, имеющий схему, представленную ниже, может выполнять в предельном случае восемь операций за один такт и работать при меньшей тактовой частоте намного более эффективнее существующих суперскалярных чипов. Добавочные функциональные блоки могут повысить производительность (за счет уменьшения конфликтов при распределении ресурсов), не слишком усложняя чип. Однако такое расширение ограничивается физическими

возможностями: количеством портов чтения/записи, необходимых для обеспечения одновременного доступа функциональных блоков к файлу регистров, и взаимосвязей, число которых геометрически растет при увеличении количества функциональных блоков. К тому же компилятор должен распараллелить программу до необходимого уровня, чтобы обеспечить загрузку каждому блоку — это, думается, самый главный момент, ограничивающий применимость данной архитектуры.

Аппаратная реализация VLIW-процессора очень проста: несколько небольших функциональных модулей (сложения, умножения, ветвления и т.д.). подключенных к шине процессора, и несколько регистров и блоков кэш-памяти. VLIW-архитектура представляет интерес для полупроводниковой промышленности по двум причинам. Первая — теперь на кристалле больше места может быть отведено для блоков обработки, а не скажем, для блока предсказания переходов. Вторая причина — VLIW-процессор может быть высокоскоростным, так как предельная скорость обработки определяется только внутренними особенностями самих функциональных модулей.

Принцип действия VLIW-компилятора.

Такой компилятор упаковывает группы независимых операций в очень длинные слова инструкций таким способом, чтобы обеспечить быстрый запуск и более эффективное их исполнение функциональными модулями. Компилятор сначала обнаруживает все зависимости между данными, а затем определяет, как их развязать. Чаще всего это делается путем переупорядочивания всей программы — разные ее блоки перемещаются с одного места в другое

26. КВАНТОВЫЕ ПРОЦЕССОРЫ

Квантовый компьютер — вычислительное устройство, работающее на основе квантовой механики. Квантовый компьютер принципиально отличается от классических компьютеров, работающих на основе классической механики.

Полноценный квантовый компьютер является пока гипотетическим устройством, сама возможность построения которого связана с серьёзным развитием квантовой теории в области многих частиц и сложных экспериментов; эта работа лежит на переднем крае современной физики. Ограниченные (до 512 кубитов) квантовые компьютеры уже построены.

Сегодняшние компьютеры работают по тому же принципу, что и нормальные машины Тьюринга – с битами, которые находятся в одном из двух состояний: 0 или 1. У квантовых компьютеров таких ограничений нет: информация в них зашифрована в квантовых битах (кубитах), которые могут содержать суперпозиции обоих состояний.

Физическими системами, реализующими кубиты, могут быть атомы, ионы, фотоны или электроны, имеющие два квантовых состояния. Фактически, если сделать элементарные частицы носителями информации, с помощью них можно построить компьютерную память и процессоры нового поколения.

Благодаря суперпозиции кубитов квантовые компьютеры изначально рассчитаны на выполнение параллельных вычислений. Этот параллелизм позволяет квантовым компьютерам выполнять одновременно миллионы вычислений.

30-кубитный квантовый компьютер по мощности будет равен суперкомпьютеру, работающему с производительностью 10 терафлопс (триллион операций в секунду).

Основная проблема считывания информации из квантовых частиц заключается в том, что в процессе измерения они могут изменить свое состояние, причем совершенно непредсказуемым образом. Фактически, если считать информацию с кубита, находящегося в состоянии суперпозиции, получим лишь 0 или 1, но никогда не оба числа одновременно. А это значит, что вместо квантового, мы будем иметь дело с нормальным классическим компьютером.

Чтобы решить эту проблему, ученые должны использовать такие измерения, которые не разрушают квантовую систему. Квантовая запутанность предоставляет потенциальное решение.

В квантовой физике, если приложить внешнюю силу к двум атомам, их можно «запутать» вместе таким образом, что один из атомов будет обладать свойствами другого. Это, в свою очередь, приведет к тому, что, например, измеряя спин одного атома, его «запутанный» близнец сразу примет противоположный спин. Такое свойство квантовых частиц позволяет физикам узнать значение кубита, не измеряя его непосредственно.

В создании компьютера нового поколения выделяют четыре направления, которые отличаются тем, что выступает в роли логических кубитов:

- направление спинов частиц, составляющих основу атома;
- наличие или отсутствие куперовской пары в установленном месте пространства;
- в каком состоянии находится внешний электрон;
- различные состояния фотона.

Основные проблемы:

Необходимость обеспечивать высокую точность измерений.

Внешние воздействия могут разрушить квантовую систему, или внести в ее работу существенные искажения. Следовательно, необходимо изолировать компоненты квантовой системы друг от друга и от внешней среды и т.п.

27. ТОПОЛОГИИ ЛОКАЛЬНЫХ СЕТЕЙ

Топология - это конфигурация соединения элементов ЛВС. Основные топологические решения делят, как правило, на два типа:

- широковещательные, в которых каждый PS (Physical Signalling) передает сигналы, которые могут восприниматься всеми остальными PS. К таким конфигурациям относятся "шина", "дерево" и "звезда с пассивным центром";
- последовательные, где каждый физический подуровень передает информацию только одному из PS. К таким топологиям относятся "кольцо", "цепочка", "звезда с интеллектуальным центром", "снежинка" и "сетка".

Основной тип конфигурации локальных вычислительных сетей (ЛВС) "шина". Коммуникации между территориально распределенными устройствами в ЛВС в чем-то похожи на коммуникации между модулями в компьютерах: высокая скорость передачи данных, частая смена структуры потока, неравномерная загрузка. Основное их отличие состоит в том, что скорость передачи данных в ЛВС может быть ниже, а длительность "взрывной интенсивности" больше. Наличие высокоскоростного общего канала - наиболее характерная особенность всех новых локальных вычислительных сетей.

Из-за неравномерного характера потока данных последние обычно передаются в форме пакетов. Так как одно устройство может получать пакеты от нескольких других устройств, то адрес отправителя - неотъемлемая часть структуры пакета.

Возможность широковещательного обращения реализуется резервированием специального адреса получателя для значения "Всем". Предполагается, что пакет с этим адресом будет обрабатываться всеми устройствами. Эффект от использования шины в ЛВС состоит в том, что внутренние шины компьютера как бы увеличиваются и охватывают целую территорию. Шины обеспечивают процессору досту п к периферийным устройствам, блокам памяти или другим процессорам, находящимся на значительном расстоянии от ЭВМ, но подключенных к шине.

Сетевой контроллер управляет использованием шины, а процессор вызовов (ПВ) управляет интерфейсом с терминалами. ПВ отвечает за установку (создание) виртуального каната через сеть к порту ЭВМ и за освобождение этого канала по окончании сеанса связи с терминалом. Во время сеанса ПВ получает данные от терминала, создает пакеты для ЛВС и направляет их сетевому контроллеру, обеспечивающему передачу пакета по шине.

Шина обычно представляет собой пассивную среду и поэтому обладает очень высокой надежностью. При использовании конфигурации "шина" возникает ряд проблем:

- каждая станция должна успеть распознать свой адрес за время, меньшее, чем время передачи данных;

- любая станция должна обеспечивать достаточную мощность сигнала, посылаемого в шину, чтобы он мог достичь наиболее удаленных станций;
- так как все станции наблюдают весь трафик, шина принципиально не защищаема, поэтому к секретным данным должны быть применены специальные способы защиты.

Конфигурация типа "дерево" образуется путем соединения нескольких шин активными повторителями или пассивными размножителями.

Конфигурация типа "звезда" - это дальнейшее развитие конфигурации "дерево с корнем" с ответвлениям к каждому подключаемому устройству.

Теперь рассмотрим конфигурации последовательного типа. Здесь к передатчикам и приемникам предъявляются более низкие требования, чем в широковещательных конфигурациях.

В конфигурациях типа "кольцо" и "цепочка" для устойчивого функционирования ЛВС требуется постоянная работа всех блоков физической среды PMA (Physical Medium Attachment).

Существуют специальные программы-"сборщики мусора", которые в случае порчи отдельных станций опознают и уничтожают невостребованные пакеты.Конфигурация "кольцо" сильно уязвима в отношении отказов, так как выход из строя какого-либо элемента кабеля останавливает работу всей сети. Некоторым выходом является конфигурация "звездообразное кольцо", все "лучи" которой содержат по две линии. Общение между станциями осуществляется через центральный блок (обычно пассивный), который используют для локализации неисправностей.

Достоинство "звездообразного кольца" - простота управления. Отметим некоторые недостатки:

- значительно увеличивается длина кабеля;
- для каждой вновь подключенной машины необходимо прокладывать свой кабель.

Конфигурация типа "цепочка", как и "кольцо", уязвима в отношении отказов и также требует регенерации сигналов каждой станцией. Передача информации через физическую среду здесь должна осуществляться в двух направлениях.

Конфигурация типа "сетка" применяется в глобальных сетях, поскольку позволяет выбирать наиболее дешевый путь для связывания абонентов. В ЛВС это менее важное достоинство, так как передающая среда не является дорогостоящим ресурсом.

28. АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.

Архитектура программного обеспечения - это первичная организация системы, сформированная ее компонентами, отношениями между компонентами и внешней средой системы, а также принципами, определяющими дизайн и эволюцию системы.

Для успешной реализации проекта объект проектирования должен быть прежде всего адекватно описан, т.е. должны быть построены полные и непротиворечивые модели архитектуры программного обеспечения, обусловливающей совокупность структурных элементов системы и связей между ними, поведение элементов системы в процессе их взаимодействия, а также иерархию подсистем, объединяющих структурные элементы.

В 70-80-х гг. при разработке программного обеспечения достаточно широко применялись структурные методы, базирующиеся на строгих формализованных методах описания программного обеспечения и принимаемых технических решений (в настоящее время такое же распространение получают объектно-ориентированные методы). Эти методы основаны на использовании наглядных графических моделей: для описания архитектуры программного обеспечения в различных аспектах (как статической структуры, так и динамики поведения системы) используются схемы и диаграммы. Наглядность и строгость средств структурного и объекно-ориентированного анализа позволяют разработчикам и будущим пользователям системы с самого начала неформально участвовать в ее создании, обсуждать и закреплять понимание основных технических решений. Однако широкое применение этих методов и следование их рекомендациям при разработке конкретных информационных систем сдерживалось отсутствием адекватных инструментальных средств, поскольку при неавтоматизированной (ручной) разработке все их преимущества практически сведены к нулю. Ручная разработка обычно порождала следующие проблемы: неадекватная спецификация требований, неспособность обнаруживать ошибки в проектных решениях, низкое качество документации, снижающее эксплуатационные характеристики, затяжной цикл и неудовлетворительные результаты тестирования.

Перечисленные проблемы породили потребность в программно-технологических средствах специальною класса - CASE-средствах, реализующих CASE-технологию создания и сопровождения программного обеспечения информационных систем. Термин CASE (Computer Aided Software Engineering) имеет весьма широкое толкование.

В середине 90-х, на волне распространения клиент-серверного подхода и начала его трансформации в “многозвенный клиент-сервер", призванный обеспечить централизованное развертывание и управление общей (для клиентских приложений) бизнес-логикой, вопросы организации архитектуры программного обеспечения стали складываться в самостоятельную и достаточно обширную дисциплину. В результате, сформировалась точка зрения на архитектуру не только в приложении к конкретной программной системе, но и

развился взгляд на архитектуру, как на приложение общих (generic) принципов организации программных компонент. В итоге, уже на сегодняшний день, на фоне такого развития понимания архитектуры, накоплен целый комплекс подходов и созданы (и продолжают создаваться и развиваться!) различные архитектурные “фреймворки”, то есть систематизированные комплексы методов, пракгик и инструментов, призванные в той или иной степени формализовать имеющийся в индустрии опыт.

Базовые принципы структурного подхода:

- принцип "разделяй и властвуй";
- принцип иерархического упорядочения - принцип организации составных частей системы в иерархические древовидные структуры с добавлением новых деталей на каждом уровне.

Выделение двух базовых принципов не означает, что остальные принципы являются второстепенными, поскольку игнорирование любого из них может привести к непредсказуемым последствиям (в том числе и к провалу всего проста). Основными из этих принципов являются:

- принцип абстрагирования - выделение существенных аспектов системы и отвлечение от несущественных;
- принцип непротиворечивости - обоснованность и согласованность элементов системы;
- принцип структурирования данных - данные должны бьпь структурированы и иерархически организованы.

Принципиальное различие между структурным и объектно-ориентированным подходом заключается в способе декомпозиции системы. Объектно-ориентированный подход использует объектную декомпозицию, при этом статическая структура системы описывается в терминах объектов и связей между ними, а поведение системы описывается в терминах обмена сообщениями между объектами. Каждый объект системы обладает своим собственным поведением, моделирующим поведение объекта реального мира.

Концептуальной основой объектно-ориентиронанного подхода является объектная модель. Основными ее элементами являются: абстрагирование (abstraction);инкапсуляция (encapsulation);модульность (modularity);иерархия (hierarchy).

Кроме основных имеются еще три дополнительных элемента, не являющихся в отличие от основных строго обязательными: типизация (typing);параллелизм (concurrency);устойчивость (persistence).

29. КОДИРОВАНИЕ ДАННЫХ С СИММЕТРИЧНЫМ ПРЕДСТАВЛЕНИЕМ ЦИФР.

В известных позиционных системах кодирования данных (СКД) для изображения положительных и отрицательных чисел используются знаки "+" и "–". В позиционных сокращенных системах кодирования, где разряды числа наряду с положительными могут принимать и отрицательные значения, знак числа явно не указывается. Особое место среди систем такого рода занимает троичная СКД. В ней введены цифры 0, 1, $\bar{1}$ для обозначения чисел 0, 1, –1. Таким образом, базисные числа расположены симметрично относительно нуля. СКД является позиционной, так как значение каждой цифры в записи числа в 3 раза больше значения той же цифры в соседней позиции. Число А в этой системе записывается в виде $A= a_n\ a_{n-1}\ ...\ a_1\ a_0\ a_{-1}\ a_{-2}\ ...$,где каждое a_i может принимать значения $\{0, 1, \bar{1}\}$. Это сокращенная запись полинома: $A=a_n3^n+ a_{n-1}3^{n-1}+\ldots+ a_13+a_0+ a_{-1}3^{-1}+\ldots$
где 3^n,..-веса разрядов.

Примеры представления чисел.

$-1=\bar{1}_3$	$1=1_3$	$-10=\bar{1}0\bar{1}_3$
$-2=\bar{1}1_3$	$2=1\bar{1}_3$	$10=101_3$
$-3=\bar{1}0_3$	$3=10_3$	$0=0_3$
$-4=\bar{1}\bar{1}_3$	$4=11_3$	$1/3=0,(3)=0,1_3$
$-5=\bar{1}11_3$	$5=1\bar{1}\bar{1}_3$	$32/81=0,11\bar{1}\bar{1}_3$

Замечание. Знак числа определяется знаком старшей значащей цифры троичного изображения числа.

Таблица сложения

a \ b	-1	0	+1
–1	$\bar{1}1$	$\bar{1}$	0
0	$\bar{1}$	0	1
+1	0	1	$1\bar{1}$

Примеры сложения чисел:

а)

35	~	1	1	0	$\bar{1}$
+15	~	1	$\bar{1}$	$\bar{1}$	0
50	~	$1\bar{1}$	0	$\bar{1}$	$\bar{1}$

б)

35	~	1	1	0	$\bar{1}$
-15	~	$\bar{1}$	1	1	0
20	~	1	$\bar{1}$	1	$\bar{1}$

Таблица умножения

a \ b	$\bar{1}$	0	1
$\bar{1}$	1	0	$\bar{1}$
0	0	0	0
1	$\bar{1}$	0	1

Пример умножения чисел:

			1	$\bar{1}$	$\bar{1}$	0	~	15
			1	$\bar{1}$	$\bar{1}$	1	~	16
			1	$\bar{1}$	$\bar{1}$	0		
		$\bar{1}$	1	1	0			
	$\bar{1}$	1	1	0				
1	$\bar{1}$	$\bar{1}$	0					
0	1	0	0	0	$\bar{1}$	0	~	240

Деление.

Если промежуточное делимое (полученное из остатка путем сноса цифры из делимого) содержит столько же разрядов, сколько делитель, то не-зависимо от того, больше это промежуточное делимое, чем делитель, или нет, в частное пишут "+1", если первые цифры делимого и делителя совпадают, или цифру "", если нет. Если вслед за этим получается остаток, содержащий столько же разрядов, сколько и делитель, то повторяют указанные действия, записывая новую цифру частного в тот же разряд.

Замечание. Если есть остаток, т. е. нацело деление не осуществляется, то в частном ставим "," и продолжаем операцию деления дальше.

Преимущества системы:

- нет знакового разряда, и знак числа явно не указывается;
- отрицательные числа получаются из равных им по абсолютной велчине положительных простой инверсией знаков всех разрядов числа.
  Из недостатков можно отметить появление нескольких значений для одного и того же разряда частного.

30. КОДИРОВАНИЕ ДАННЫХ В СИСТЕМАХ С ОТРИЦАТЕЛЬНЫМ ОСНОВАНИЕМ.

Известны компьютеры, работающие в СКД с отрицательными основаниями. Целесообразность введения отрицательного основания обусловливается тем, что знак числа органически включается в представление числа, в связи с чем специально его отображать не надо. Известно, что каждое целое число A может быть представлено в виде

$A = \sum_{i=0}^{m} C_i n^i$. где при n < –1 имеет место 0<=Ci < –n–1.

Рассмотрим наиболее привычный случай СКД при n = –2. Здесь Ci {0; 1}. Предположим, что m = 5. Тогда таблица представления целых чисел от –10 до 10 будет иметь следующий вид (табл. 1).

Представление целых чисел

Веса	$(-2)^4$	$(-2)^3$	$(-2)^2$	$(-2)^1$	$(-2)^0$	Веса	$(-2)^4$	$(-2)^3$	$(-2)^2$	$(-2)^1$	$(-2)^0$
Числа	16	-8	4	-2	1	Числа	16	-8	4	-2	1
0	0	0	0	0	0	-10	0	1	0	1	0
1	0	0	0	0	1	-9	0	1	0	1	1
2	0	0	1	1	0	-8	0	1	0	0	0
3	0	0	1	1	1	-7	0	1	0	0	1
4	0	0	1	0	0	-6	0	1	1	1	0
5	0	0	1	0	1	-5	0	1	1	1	1
6	1	1	0	1	0	-4	0	1	1	0	0
7	1	1	0	1	1	-3	0	1	1	0	1
8	1	1	0	0	0	-2	0	0	0	1	0
9	1	1	0	0	1	-1	0	0	0	1	1
10	1	1	1	1	0						

Из табл. 5.1 видно, что знак числа определяется местоположением первой значащей цифры: если старшая значащая цифра стоит в четном разряде, число отрицательное, если в нечетном – положительное.

1. Сложение. Пусть α_i, β_i – разряды слагаемых; P_{i-1} – перенос из (i-1)-го разряда на i-й; P_i – перенос из i-го в (i+1)-й. Тогда справедливо соотношение $\gamma_i + P_i(-2) = \alpha_i + \beta_i + P_{i-1}$.

Возможные значения разрядов:

$P_i \in \{0; 1; \overline{1}\}$, $\gamma_i \in \{0; 1\}$, $\alpha_i, \beta_i \in \{0; 1\}$.

Пример. Пусть $\gamma_i + P_i(-2) = 2$, тогда $\gamma_i = 0$, $P_i = -1$;

$\gamma_i + P_i(-2) = -1$, тогда $\gamma_i = 1$, $P_i = 1$.

Таблица сложения

α_i	0	0	1	1		0	0	1	1		0	0	1	1
β_i	0	1	0	1		0	1	0	1		0	1	0	1
P_{i-1}	0	0	0	0		$\overline{1}$	$\overline{1}$	$\overline{1}$	$\overline{1}$		1	1	1	1
γ_i	0	1	1	0		1	0	0	1		1	0	0	1
P_i	0	0	0	$\overline{1}$		1	0	0	0		0	$\overline{1}$	$\overline{1}$	$\overline{1}$

2. Вычитание. Так как a-b = a+b-2b = a+b+b(-2), причем b(-2) – это число b, сдвинутое на один разряд влево, то алгоритм вычитания может быть сформулирован так. Для того чтобы из a вычесть b, необходимо к a прибавить b и прибавить b, сдвинутое на один разряд влево.

3. Умножение Операция умножения осуществляется посредством последовательных сложений и сдвигов. Деление см. вопрос 35.

4. Деление. Пусть требуется разделить число υ на число α и получить частное ω, то есть $\upsilon : \alpha = \omega$.

Введем следующие обозначения:

α_k — это α, сдвинутое на k разрядов влево;

$\upsilon_{i+1} = \upsilon_i - \alpha_k$, на каждом шаге алгоритма индекс у υ_i увеличивается на 1;

$N(\upsilon)$ — номер разряда старшей значащей цифры.

Разрядность частного ω равна $k + 1$, а веса его разрядов будут иметь вид:

$$(-2)^k, (-2)^{k-1}, \ldots, (-2)^0.$$

Алгоритм деления.

1. Сдвигаем α на k разрядов влево до тех пор, пока старшая значащая цифра у α не станет под старшей значащей цифрой υ, таким образом:

$\alpha \rightarrow \alpha_k$, то есть α преобразуется в α_k;

$\upsilon = \upsilon_0$, то есть υ присваивается начальное обозначение υ_0.

2. Вычитаем $\upsilon_0 - \alpha_k$, то есть $\upsilon_1 = \upsilon_0 + \alpha_k + (-2)\,\alpha_k$.

3. Если $N(\upsilon_{i+1}) = N(\alpha_k)$, то записываем «1» в разряд $(-2)^k$ частного и еще раз вычитаем α_k.

4. Если $N(\upsilon_{i+1}) < N(\alpha_k)$, то записываем «1» в разряд $(-2)^k$ частного и α_k меняем на α_{k-1}, то есть сдвигаем α_k на 1 разряд вправо и вычитаем α_{k-1}.

5. Если $N(\upsilon_{i+1}) > N(\alpha_k)$, то записываем «0» в разряд $(-2)^k$ частного, υ_{i+1} заменяем на υ_i, α_k меняем на α_{k-1} и производим вычитание.

6. Процесс завершается, если $\upsilon_{i+1} = 0$.

Замечание 1. Результат «0» или «1» записывается в разряд частного с весом $(-2)^k$, где k — индекс у α.

Замечание 2. Возможен случай, когда в один и тот же разряд частного записывается несколько значений. Формирование частного потребует дополнительно одну операцию сложения.

Замечание 3. Сравнение очередного υ_i с нулем (п. 6 алгоритма) осуществляется на каждом шаге после выполнения одного из пунктов 3, 4 или 5.

31. КОДИРОВАНИЕ ДАННЫХ С ПОМОЩЬЮ ВЫЧЕТОВ.

Пусть $P_1, P_2, ..., P_n$ — целые числа; $P_i > 1$, $(P_i, P_j) = 1$, $i \neq j$; $M = \prod_{i=1}^{n} P_i$;

x_i – наименьшие неотрицательные решения системы сравнений

$A \equiv x_i \pmod{P_i}$, $i = 1, 2, ..., n$; $A \subset [0,M)$.

Тогда кортеж (x1, x2, ..., xn) будем называть кодом числа A в системе в коде вычетов (СКВ) при заданных основаниях P1, P2, ..., Pn. Записывают это обычно так: A . (x1, x2, ..., xn). Компоненты xi называют разрядами числа A в СКВ. Идейными корнями данная система восходит к так называемой китайской теореме об остатках. Среди особенностей СКВ исследователи отмечали следующие:

. каждый разряд числа несет информацию обо всем числе, откуда следует независимость разрядов друг от друга;

. отсутствие переносов между разрядами при выполнении арифметических операций упрощает их выполнение;

. малоразрядность компонент числа, в связи с чем арифметические операции превращаются в однотактные, иногда просто в выборку результата из таблицы.

Арифметические операции

Рассмотрим алгоритмы арифметических операций. Пусть необходимо выполнить операцию $A \otimes B = C$, где $A \sim (x_1,x_2,...,x_n)$; $B \sim (y_1,y_2,...,y_n)$; $C \sim (\gamma_1,\gamma_2,...,\gamma_n)$.

1. Если $\otimes$ – сложение, то

$\gamma_i = |x_i + y_i|_{P_i}$, т.е. $\gamma_i = x_i + y_i - l_i P_i$,

где

$$l_i = \begin{cases} 1, \text{если } x_i + y_i \geq P_i, \\ 0, \text{если } x_i + y_i < P_i, \end{cases} \quad i = \overline{1,n}.$$

2. Если $\otimes$ – вычитание, то

$\gamma_i = |x_i - y_i|_{P_i}$, т. е. $\gamma_i = x_i - y_i + l_i P_i$,

где

$$l_i = \begin{cases} 1, \text{если } x_i < y_i \\ 0, \text{если } x_i \geq y_i \end{cases}$$

3. Если $\otimes$ – умножение, то

$\gamma_i = |x_i y_i|_{P_i}$, т. е. $\gamma_i = x_i y_i - l_i P_i$,

где $l_i = [\frac{x_i y_i}{P_i}]_{p_i}$; [x] – наибольшее целое, не больше x.

4. Если $\otimes$ – деление, то

$$\gamma_i = \left|\frac{x_i + k_i P_i}{y_i}\right|_{p_i}, \text{ где } k_i = 0, 1, ..., P_i - 1$$

Иногда операцию деления A/B заменяют операцией умножения A на мультипликативную инверсию от B по модулю M, т. е. выполняют умножение:

$(x_1, x_2, ..., x_n) \cdot (y_1', y_2', ..., y_n')$,

где $(y_1', y_2', ..., y_n')$ – мультипликативная инверсия от B по модулю M. Здесь $y_i' = \left|\frac{1}{y_i}\right|_{P_i}$, т. е. y_i' – решение уравнения $|y_i' y_i|_{P_i} = 1$ или сравнения $y_i' y_i \equiv 1 (\mathrm{mod}\ P_i)$. Мультипликативная инверсия существует, если $(y_i, P_i) = 1$, для всех $i = \overline{1,n}$.

Пример. Пусть $P_1 = 2$, $P_2 = 3$, $P_3 = 5$. $M = 30$. Тогда числу $A = 7$ будет в данной СКВ соответствовать код (1, 1, 2), а числу $3 \sim (1, 0, 3)$.

Следует отметить, что кодирование данных в СКВ в силу вышеуказанных свойств позволяет существенно уменьшить время выполнения основных арифметических операций.

Восстановление позиционного значения числа A по его коду $(x_1, x_2, ..., x_n)$ в СКВ осуществляется из соотношения

$$A_{\text{пос}} = \sum_{i=1}^{n} x_i B_i - r_A M,$$

где B_i – ортогональные базисы; r_A – ранг числа A.

Ортогональные базисы представляют собой целые числа, удовлетворяющие соотношению

$$B_i \equiv \begin{cases} 1(\mathrm{mod} P_i) \\ 0(\mathrm{mod} P_j) \end{cases}, \ i \neq j, i, j = 1,2,...n,$$

и ищутся среди чисел вида

$$m_i = \prod_{j=i}^{n} P_j, \text{ где } m_i \subset \{1, 2, ..., P_i - 1\}.$$

Исследования показали, что полученные алгоритмы выполнения немодульных операций в рамках классической СКВ близки к предельным. В связи с этим рассмотрим некоторые обобщения классической СКВ, которые позволили бы усовершенствовать алгоритмы основных операций в плане уменьшения их вычислительной сложности, что дало бы возможность расширить класс задач, решаемых на компьютерах с базовой системой СКВ.

Итак, целое неотрицательное число A будем представлять в виде

$$(\lambda_1 x_1 + (1 - \lambda_1)(x_1 - P_1), \ldots, \lambda_n x_n + (1 - \lambda_n)(x_n - P_n)), \qquad (1)$$

где $\lambda_i \in \{0, 1\}$, $x_i = |A|_{p_i}$. Значения λ_i следует выбирать таким образом, чтобы форма (1) для любого целого числа A из области определения имела нулевой ранг. Назовем форму (1) формой нулевого ранга (ФНР).

Определение 1. Систему в коде вычетов, для которой любое целое $A \in [0, M)$ допускает ФНР, назовем безранговой СКВ (БСКВ).

Справедлива следующая теорема 1.

32. КОДИРОВАНИЕ ЧИСЛОВОЙ, ТЕКСТОВОЙ, ГРАФИЧЕСКОЙ, ЗВУКОВОЙ ИНФОРМАЦИИ.

Инф. Как правило представляет собой целые действительные числа. Для их представления есть 2 формы: с фикс. точкой и плав. точкой. Целые числа преобразуются в двоичную форму путем последовательного деления при записи в обратном порядке.

Кроме чисел мы кодируем текстовую инф. Основная операция, проводимая над отдельными символами – это сравнение символов, для этой операции важна уникальность кода и его длины. Для кодирования текста исп. различные таблицы перекодировки. Институт стандартизации США ввел в действие систему кодирования инф ASCII. Аналогичные были и в др странах (ссср- кои-7).

Кодирование граф. инф. в основном явл. его разбиением на дискретные элементы (дискретизация). Чем детальнее разбит рис. тем он точнее. Основными способами представления графики для ее храния и обработки явл. растровые и векторные изображения, векторные – совокупность элементарные геометрич. фигуры (дуги, отрезки). Для каждой линии указываются двоичные коды типа линии, толщины и цвета. Растровые изображения- совокупность точек полученных в рез. дискретизации в соответствии с матричным принципом.

Пиксель – элементарная единица изображения, цвет и яркость к-й можно задавать независимо от остального изображения . Чем пиксели плотнее тем качество круче.

Кодирование звуковой информации. Для компьютерной обработки аналогового сигнала преобраз. последовательность двоичных чисел, а для этого необходимо его разбить на отдельные сигналы и оцифровать.

Цифровой звук – аналоговый звуковой сигнал, представленный посредством дискретных численных значений его амплитуды.

Оцифровка звука включает 2 процесса:

1. Выборка сигнала по времени (дискретизация);
2. Квантование по амплитуде.

Методы кодирования звука основаны на том, что теоретически любой сложный звук можно разложить на послед. гармонич. сигналов разных частот, каждый из кот. представ. собой синусоиду, называемую спектром исходного сигнала.

Задачей кодирования явл представление звука в форме аналог или цифр сигнала. Процесс дискретизации по времени – это процесс получения знач сигнала, кот преобразует с опред временным шагом.

Рассказать про сжатие с потерями и без потерь.

33. ВЫЧИСЛЕНИЯ С ЧИСЛАМИ КОНЕЧНОЙ ТОЧНОСТИ.

Когда люди выполняют какие-либо арифметические действия, их не волнует вопрос, сколько десятичных разрядов занимает то или иное число. Проблема нехватки бумаги для записи числа никогда не возникает.

С компьютерами дело обстоит иначе. В большинстве компьютеров количество доступной памяти для хранения чисел фиксировано и зависит от того, когда был выпущен этот компьютер. Если приложить усилия, программист сможет представлять числа в два, три и более раза большие, чем позволяет объем памяти, но это не меняет природы данной проблемы. Память компьютера ограничена, поэтому мы можем иметь дело только с такими числами, которые можно представить в фиксированном количестве разрядов. Такие числа называются числами конечной точности.

Рассмотрим ряд положительных целых чисел, которые можно записать тремя десятичными разрядами без десятичной точки и без знака. В этот ряд входит ровно 1000 чисел: ООО, 001, 002, 003,.., 999. При таком ограничении невозможно выразить определенные типы чисел. Сюда входят:

- числа больше 999;
- отрицательные числа;
- дроби;
- иррациональные числа;
- комплексные числа.

Одно из свойств набора всех целых чисел - замкнутость по отношению к операциям сложения, вычитания и умножения. Другими словами, для каждой пары целых чисел i и j числа i + j, i - j и i * j - тоже целые числа. Ряд целых чисел не замкнут относительно деления, поскольку существуют такие значения i и j, для которых i/j не выражается в виде целого числа (например, 7/2 или 1/0).

Числа конечной точности не замкнуты относительно всех четырех операций. Вот примеры операций над трехразрядными десятичными числами:

♦ Слишком большое число: 600 + 600 = 1200.

♦ Отрицательное число: 003 - 005 = -2.

♦ Слишком большое число: 050 x 050 - 2500.

♦ Не целое число: 007 / 002 = 3,5.

Отклонения можно разделить на два класса: операции, результат которых больше самого большого числа ряда (ошибка переполнения) или меньше самого маленького числа ряда (ошибка потери значимости), и операции, результат которых не является слишком маленьким или слишком большим, а просто не является членом ряда. Из четырех приведенных примеров первые три относятся к первому классу, а четвертый - ко второму классу.

Поскольку объем памяти компьютера ограничен и компьютер должен выполнять арифметические действия над числами конечной точности, с точки зрения классической математики результаты определенных вычислений оказываются неправильными. Ошибка в данном случае - это только следствие

конечной природы представления чисел в вычислительном устройстве. Некоторые компьютеры имеют встроенную аппаратную поддержку для обнаружения ошибок переполнения.

Алгебра чисел конечной точности отличается от обычной алгебры. В качестве примера рассмотрим ассоциативный закон a + (b - c) = (a + b) - c.

Вычислим обе части выражения для a = 700, b = 400, c = 300. В левой части сначала вычислим значение (b - c). Оно равно 100. Затем прибавим это число к a и получим 800. Чтобы вычислить правую часть, сначала вычислим (a + b). Для 3-разрядных целых чисел получится переполнение. Результат будет зависеть от компьютера, но он окажется неравным 1100. Вычитание 300 из какого-то числа, отличного от 1100, не даст в результате 800. Ассоциативный закон не имеет силы. Важна очередность выполнения операций.

Другой пример - дистрибутивный закон:

a*(b-c) = a*b-a*c.

Подсчитаем обе части выражения для a = 5, b = 210, c = 195. В левой части 5 x 15 = 75. В правой части 75 не получается, поскольку результат выполнения операции ax b выходит за пределы ряда.

Исходя из этих примеров, кто-то может сделать вывод, что компьютеры совершенно непригодны для выполнения арифметических действий. Вывод, естественно, неверен, но эти примеры наглядно показывают, как важно понимать механизм работы компьютера и знать о его ограничениях.

34. ПОМЕХОЗАЩИТНЫЕ КОДЫ. КОД ХЭММИНГА

Наиболее известные из самоконтролирующихся и самокорректирующихся кодов – коды Хемминга. Построены они применительно к двоичной системе счисления.

Для построения самокорректирующегося кода достаточно иметь один контрольный разряд (код с проверкой на четность). Но при этом мы не получаем никаких указаний о том, в каком именно разряде произошла ошибка, и, следовательно, не имеем возможности исправить ее. Остаются незамеченными ошибки, возникшие в четном числе разрядов.

Коды Хемминга имеют бóльшую относительную избыточность, чем коды с проверкой на четность, и предназначены либо для исправления одиночных ошибок (при d = 3), либо для исправления одиночных и обнаружения без исправления двойных ошибок (d = 4). В таком коде n-значное число имеет m информационных и k контрольных разрядов. Каждый из контрольных разрядов является знаком четности для определенной группы информационных знаков слова. При декодировании производится k групповых проверок на четность. В результате каждой проверки в соответствующий разряд регистра ошибки записывается 0, если проверка была успешной, или 1, если была обнаружена нечетность.

Группы для проверки образуются таким образом, что в регистре ошибки после окончания проверки получается K-разрядное двоичное число, показывающее номер позиции ошибочного двоичного разряда. Изменение этого разряда – исправление ошибки.

Первоначально эти коды предложены Хеммингом в таком виде, при котором контрольные знаки занимают особые позиции: позиция i-го знака имеет номер 2^{i-1}. При этом каждый контрольный знак входит лишь в одну проверку на четность.

Рассмотрим код Хемминга, предназначенный для исправления одиночных ошибок, т. е. код с минимальным кодовым расстоянием d = 3.

Ошибка возможна и в одной из n позиций. Следовательно, число контрольных знаков, а значит, и число разрядов регистра ошибок должно удовлетворять условию

$$k \geq \log_2(n + 1). \quad (1)$$

Отсюда

$$m \leq n - \log_2(n + 1). \quad (2)$$

Значения m и k для некоторых коротких кодов, вычисленные по формулам (1) и (2) даны в табл. 11.1.

Таблица 11.1. Значения m и n

n	3	4	5	6	7	8	9	10	11	12
m	1	1	2	3	4	4	5	6	7	8
k	2	3	3	3	3	4	4	4	4	4

Чтобы число в регистре ошибок (РОШ) указывало номер позиции ошибочного разряда, группы для проверки выбираются по правилу:

I гр. : все нечетные позиции, включая и позиции контрольного разряда, т. е. позиции, в первом младшем разряде которых стоит 1.

II гр. : все позиции, номера которых в двоичном представлении имеют 1 во втором разряде справа (например, 2, 3, 6, 7, 10) и т. д.

III гр. : разряды, имеющие "1" в третьем разряде справа, и т. д.

Примечание: каждый контрольный знак входит только в одну проверяемую группу.

Пример 1. Пусть k = 5 (табл. 11.2).

Таблица 11.2. Формирование контрольных групп

Номер проверки	Позиция контрольного знака	Проверяемые позиции
1	1	1, 3, 5, 7, 9, 11, 13, ...
2	2	2, 3, 6, 7, 10, 11, ...
3	4	4, 5, 6, 7, 12, 13, ...
4	8	8, 9, 10, 11, 12, 13, ...

5	16	16, 17, 18, 19, 20, 21

Пример 2. Рассмотрим семизначный код Хемминга, служащий для изображения чисел от 0 до 9 (табл. 11.3).

Таблица 11.3. **Семизначный код Хемминга**

Десятичное число	Простой двоичный код				Код Хемминга						
								к		К	к
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1	1	1
2	0	0	1	0	0	0	1	1	0	0	1
3	0	0	1	1	0	0	1	1	1	1	0
4	0	1	0	0	0	1	0	1	0	1	0
5	0	1	0	1	0	1	0	1	1	0	1
6	0	1	1	0	0	1	1	0	0	1	1
7	0	1	1	1	0	1	1	0	1	0	0
8	1	0	0	0	1	0	0	1	0	1	1
9	1	0	0	1	1	0	0	1	1	0	0

Пусть передан код числа 6 в виде "0 1 1 0 0 1 1", а приняли в виде "0 1 0 0 0 1 1". Проверочные группы:

I проверка : разряды 1, 3, 5, 7 – дает 1 в младший разряд РОШ.

II проверка : разряды 2, 3, 6, 7 – дает 0 во второй разряд РОШ.

III проверка: разряды 4, 5, 6, 7 – дает 1 в третий разряд РОШ.

Содержимое РОШ "101", значит ошибка в пятой позиции.

Примечание. В каждый из контрольных разрядов при построении кода Хемминга посылается такое значение, чтобы общее число единиц в его контрольной сумме было четным. РОШ заполняется начиная с младшего разряда.

Вывод. Рост кодового расстояния позволяет увеличить корректирующую способность кода. В то время как d = 2 у кода с проверкой на четность позволяет обнаруживать одиночную ошибку, код Хемминга с d = 3 ис-правлет ее.

35. АЛГОРИТМ ДЕЛЕНИЯ В СИСТЕМЕ С ОТРИЦАТЕЛЬНЫМ ОСНОВАНИЕМ

4. Деление. Пусть требуется разделить число υ на число α и получить частное ω, то есть $\upsilon : \alpha = \omega$.

Введем следующие обозначения:

α_k — это α, сдвинутое на k разрядов влево;

$\upsilon_{i+1} = \upsilon_i - \alpha_k$, на каждом шаге алгоритма индекс у υ_i увеличивается на 1;

$N(\upsilon)$ — номер разряда старшей значащей цифры.

Разрядность частного ω равна $k + 1$, а веса его разрядов будут иметь вид:

$$(-2)^k, (-2)^{k-1}, \ldots, (-2)^0.$$

Алгоритм деления.

1. Сдвигаем α на k разрядов влево до тех пор, пока старшая значащая цифра у α не станет под старшей значащей цифрой υ, таким образом:

$\alpha \rightarrow \alpha_k$, то есть α преобразуется в α_k;

$\upsilon = \upsilon_0$, то есть υ присваивается начальное обозначение υ_0.

2. Вычитаем $\upsilon_0 - \alpha_k$, то есть $\upsilon_1 = \upsilon_0 + \alpha_k + (-2)\,\alpha_k$.

3. Если $N(\upsilon_{i+1}) = N(\alpha_k)$, то записываем «1» в разряд $(-2)^k$ частного и еще раз вычитаем α_k.

4. Если $N(\upsilon_{i+1}) < N(\alpha_k)$, то записываем «1» в разряд $(-2)^k$ частного и α_k меняем на α_{k-1}, то есть сдвигаем α_k на 1 разряд вправо и вычитаем α_{k-1}.

5. Если $N(\upsilon_{i+1}) > N(\alpha_k)$, то записываем «0» в разряд $(-2)^k$ частного, υ_{i+1} заменяем на υ_i, α_k меняем на α_{k-1} и производим вычитание.

6. Процесс завершается, если $\upsilon_{i+1} = 0$.

Замечание 1. Результат «0» или «1» записывается в разряд частного с весом $(-2)^k$, где k — индекс у α.

Замечание 2. Возможен случай, когда в один и тот же разряд частного записывается несколько значений. Формирование частного потребует дополнительно одну операцию сложения.

Замечание 3. Сравнение очередного υ_i с нулем (п. 6 алгоритма) осуществляется на каждом шаге после выполнения одного из пунктов 3, 4 или 5.

Пример 1. Пусть необходимо число 14 разделить на число 2, то есть $\upsilon_0 = 14$, $\alpha = 2$.

Тогда $14 \sim 10010_{(-2)}$, $2 \sim 110_{(-2)}$.

Для выравнивания старших значащих цифр делимого и делителя сдвигаем α на два разряда влево, то есть $\alpha \rightarrow \alpha_2 = 11000$.

Структура частного ω:

Веса разрядов	$(-2)^2$	$(-2)^1$	$(-2)^0$	
Частное	+1	+1	+1	
	+1			
	110	1	1	$= 7$

Процесс деления:

	1 0 0 1 0	υ_0	
	1 1 0 0 0	α_2	
Записываем	1 1 0 0 0		
«1» во второй разряд	← 0 1 1 0 1 0	υ_1	$N(\upsilon_1) = N(\alpha_2)$
			$V_1 = 0$? нет
	1 1 0 0 0	α_2	
Записываем	1 1 0 0 0		
«1» во второй разряд	← 0 0 0 0 1 0	υ_2	$N(\upsilon_2) < N(\alpha_2)$
			$V_2 = 0$? нет
	1 1 0 0	α_1	
Записываем	1 1 0 0		$N(\upsilon_3) < N(\alpha_1)$
«1» в первый разряд	← 0 0 0 1 1 0	υ_3	$V_3 = 0$? нет
	1 1 0	α_0	
Записываем	1 1 0		
«1» в нулевой разряд	← 0 0 0 0 0 0	υ_4	$N(\upsilon_4) < N(\alpha_0)$
			$V_4 = 0$? да
			Stop

Процесс деления завершен, так как $\upsilon_4 = 0$.

36. ПРОЦЕСС. ГРАФИЧЕСКОЕ ПРЕДСТАВЛЕНИЕ. ФОРМАТЫ ПРОЦЕССОВ В ПАМЯТИ.

Процесс есть тройка (Q,f, g). где Q - множество состояний процесса;

f - функция действия f:Q->Q; gcQ - начальное состояние процесса.

Действия, реализуемые процессом, будем рассматривать как результат выполнения некоторой программы на реальном процессоре.

Свойства процесса:

- процесс не является закрытой системой и может взаимодействовать с другими процессами, воспринимая или изменяя часть среды, кот.он с ними разделяет;
- каждый процесс существует лишь временно. Имеется и «главный» процесс, выполняемый вручную при включении компьютера и начинающий всю цепочку процессов;
- в любой момент процесс может быть описан его состоянием. Все параметры (переменные), характеризующие текущее состояние процесса, объединяются в «вектор состояний» или «слово состояний», которые позволяют возобновить как процесс после его прерывания, так и локальную среду

Графическое представление.

Чаще всего процесс представляется в виде некоторого графа, включающего кружки – вершины, соединенные стрелками-дугами. По традиции вершины означают состояния процесса, а дуги переходы между ними, при этом корневая вершина соответствует начальному состоянию процесса. Дуги помечаются именем (номером) события, соответствующего данному переходу

Рассмотрим структуру записей, описывающих соответствуюшие процессы, в памяти компьютера. Любой процесс, управляемый хранящейся в памяти программой, имеет два важных параметра:

- адрес, по которому выбирается следующая команда;

- адрес страницы (блока) данных, который следует прибавлять к каждой ссылке на данные во время обращения.

Периферийный процесс характеризуется типом устройства и адресом данных, участвуюших в обмене. Кроме того, процесс может быть «свободным» или «занятым», пока не выполнится определенное условие.

37. УПРАВЛЕНИЕ ПРОЦЕССАМИ В МНОГОПРОЦЕССОРНЫХ КОМПЬЮТЕРАХ

Рассмотрим проекты ВС, включающие три компоненты: среду, процессоры и управление. Отметим, что ЦП управляется последовательностью команд, вызываемых откуда-то из среды, а периферийный процессор действует в соответствии с фиксированной, встроенной последовательностью команд.

ВС 1. Предположим, что имеется достаточное (бесконечное) количество процессоров различного типа, чтобы обслуживать любой родившийся процесс.

Управление будет состоять из ЦП, управляемого процедурой, расположенной в памяти. Управляющий процессор соединен линиями связи с другими процессорами, запуская их и получая всю информацию об их состоянии. Обычное состояние ЦП - ожидание какого-нибудь события. При возбуждении управления оно определяет причину возбуждения, обрабатывает ее и, если возможно, запускает этот и/или другой процесс. Закончив обслуживание какого-то устройства, ЦП, прежде чем перейти в режим ожидания, проверяет, пуста ли очередь на обслуживание.

Важной особенностью таких структур является необходимость защиты информации от несанкционированного чтения-записи.

ВС 2. Снимем наше предположение о бесконечном количестве процессоров различных типов. Теперь процессы будут выстраиваться в очередь из-за отсутствия свободного процессора. При освобождении очередного процессора управляющий процессор (УП) должен решать, кому его предоставить. Если снабдить УП механизмом прерывания всех процессоров, то УП после сигнала об освободившемся процессоре может перераспределить работу всех процессоров заново. Этот проект оптимальнее предыдущего.

ВС 3. На третьем этапе из-за того, что УП часто простаивает, мы можем функции УП распределять между всеми ЦП. Центральные процессоры, работая в обычном режиме, выполняют какой-то процесс и могут быть прерваны ЦП. работающим в режиме управления. Последний не может быть прерван. Работой всех ЦП в управляющем режиме руководит одна и та же управляющая процедура.

ВС 4. На четвертом этапе предположим, что в управляющем режиме могут работать одновременно несколько ЦП. В этом случае следует принять меры защиты информации при операциях с таблицами состояния процессов.

Возбуждение управляющей процедуры (УП) может происходить одним из следующих способов:

- если процесс, выполняющийся в ЦП, выдает сигнал связи, то этот же ЦП переключается в управляющий режим и выполняет соответствующие действия:
- освободившийся от управления ЦП передает себя какому-нибудь процессу;
- прерывание от периферийного устройства переводит один из ЦП в режим управления. Просмотр всех ЦП происходит поочередно, и если все они находятся в режиме управления, прерывание помещается в очередь.

В целях упрощения реализации можно все прерывания адресовать одному ЦП. Однако при этом увеличивается время обработки каждого события.

*Управление процессами в однопроцессорном компьютере

Так как большинство существующих ВС имеет один центральный процессор, то этот случай изучим подробнее.

Рассмотрим абстрактную ВС с ОП, одним ЦП, устройством ввода и устройством печати.

Пусть в исходном состоянии все периферийные устройства (ПУ) находятся в состоянии покоя, а ЦП выполняет процесс $P_{i,}$ управляемый последовательностью команд. Если в момент t_i этот процесс выдает код, возбуждающий процесс в ПУ, то ЦП переходит в управляющий режим и посылает сигнал по линии связи $A_{1,}$ A_2 или A_3. Затем ЦП возвращается к выполнению своего процесса. По окончании работы одно из ПУ посылает сигнал прерывания по линии $B_{1,}$ устанавливая тем самым i-й разряд регистра прерываний в соответствующее состояние. ЦП переходит в управляющий режим, анализирует состояние регистра прерываний для опознания ПУ, вызвавшего прерывания, и выполняет соответствующие действия. До перехода в режим управления ЦП запоминает состояние прерываемого процесса. Если во время обработки прерывания приходит новое прерывание, обработка предыдущего прерывания будет доведена до конца.

38. ИНФОРМАЦИОННЫЕ МОДЕЛИ: МУЛЬТИПРОЦЕССОРЫ И МУЛЬТИКОМПЬЮТЕРЫ.

Предложены и реализ-ны 2 инф модели взаимодейств-я: мультипроцессоры и мультикомп-ры.

Мультипроцессоры. Если все проц-ры разделяют общую физич память, то такая с-ма наз-ся мультипроцес-м. Мультипроц модель распростр-ся и на ПО. Все проц-сы вместе работающие на мультипр-ре могут разделять одно вирутуал адресной простр-во, отображ-ое в общую память. Любой процесс может считывать слово из памяти или записывать слово в память с пом-ю команд LOAD и STORE. Два процесса могут обмен-ся инф-ей, если один из них будет просто записывать данные в память, а другой считывать эти данные. Благодаря такой возм-ти взаимод-я двух и более процессов мультипроцессоры весьма популярны.

Примеры мультипроцессоров: Sun Enterprise 10000, Sequent NUMA-Q, SGI Origin 2000 и HP/Convex Exemplar.

Выделяют 3 вида мультипроц-в в зав-ти от механизма реализации памяти совместного исп-я:

1)UMA(Uniform Memory Access)с однор доступом к памяти,

2) NUMA – с неоднор доступом к памяти,

3) COMA (Cash Only ..)с доступом только к кеш-памяти.

UMA обесп одно и то же время доступа к слову в незав-ти от его распол-я внутри модуля и расстояния модуля от процессора. Эта арх-ра с предсказуемой производит-ю.

В NUMA доступ к данным существенно зависит от расстояния процессора до модуля с необх-ми данными. NUMA м б как с кеш-памятью (CC-NUMA), так и без неё(NC-NUMA). Наличие кеш-памяти несколько сглаживает разницу во времени доступа к инф-ии в разных модулях.Однако если объём кеш-памяти сущ-но меньше объёма требуемых данных, произв-ть мультипроц-ра снижается.

В арх-ре COMA память каждого проц-ра исп-ся как кеш-память.В ней всё физич адресное простр-во делится на строки, которые мигрируют по сис-ме по мере необх-ти. Такая организация повышает частоту обращения кеш-памяти, повышая тем самым произв-ть с-мы.

Мультикомпьютеры. Параллельную арх-ру при к-й каждый процессор имеет свою собств память, доступную только этому процессору, наз-т муьтикомп-ром (с-мой с распределённой памятью). Мультикомп-ры обычно явл-ся с-мой со слабой связью. Ключевое отличие мультикомп-ра от мультипроц-ра сост в том, что каждый процессор в мультикомп-ре имеет свою собств уник память, к кот этот процессор может обращаться, выполняя команды LOAD и STORE. След-но. Мультипроц-ры имеют одно физич адрес пр-во, разделяемое всеми процессорами, а мультикомп-ры содержат отдельное физич адр простр-во для каждого процессора.

Так как процессоры в мультикомп-ре не могут взаимодействовать друг с другом путём чтения из общей памяти и записи в общую память, необходим другой механизм взаимодействия. Здесь процессоры рассылают друг другу сообщения, используя сеть межсоединений.

При отсутствии памяти совместного исп-я в аппаратном обеспечении располагается опр структура ПО. В мультикомп-ре невозможно иметь одно вирт адр пр-во. В мультикомп-ре для взаимодействия между процессорами часто исп-ся примитивы SEND и RECEIVE. Поэтому ПО мультикомп-ра имеет более сложную стр-ру, чем программное обесп-е мультипроц-ра. При этом основной проблемой становится правильное разделение данных и разумное их размещение. В мультипроц-ре размещение частей не влияет на правильность вып-я задачи. Хотя может повлиять на произв-ть.

Т.о.,мультикомп-р программировать сложнее, чем мультипроцессор. А мультипроц-ры сложно строить, но легко программировать. Поэтому стали предприниматься попытки создания гибридных с-м, которые отн-но легко конструировать и легко прогр-ть. Это привело к различной реализации совместной памяти. Практически все исследования в области арх-р с паралл обработкой напр на создание гибр форм, кот сочет в себе преимущ-во обеих арх-р. Важно получить такую с-му, к-я была бы расширяема.

Первый подход к решению такой задачи основан на том, что соврем комп с-мы не монолитны, а состоят из ряда уровней. Это даёт возм-ть реализовать общую память на любом из нескольких уровней.

Второй подход: исп-ть аппар обесп-е мультикомп-ра и опер с-му, кот моделирует разделённую память, обеспечивая единое вирт пр-во, разбитое на страницы. При таком подходе(DSM) каждая страница расположена в одном из блоков памяти. Каждая машина сод-т свою собств вирт память и собст таблицы страниц.

3ий подход:реализовать общую раздел-ю память на уровне ПО. При таком подходе абстракцию раздел памяти создаёт язык прогр-я и эта абстракция реализ-ся компилятором.

39. МЕТРИКА АППАРАТНОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Проектирование систем параллельного действия (СПД) ставит целью достичь максимальноrо быстродействия при минимальных затратах. Рассмотрим вопросы производительности, связанные с проектированием архитектур СПД.

ОСНОВНЫМИ параметрами аппаратноrо обеспечения метрики спд являются параметры функционирования системы межсоединений время ожидания и пр опускная способность.

Полное время ожидания состоит из времени на отправку пакета и времени получения ответа. При посылке пакета в память это время чтения и записи в память, при пересылке «процессор-процессор» это время процессорной связи для пакетов минимальноrо размера (как правило, одноrо слова).

При **коммуrации каналов** время ожидания состоит из времени установки и времени передачи. для установки схемы передачи следует выслать пробный пакет с целью резервирования требуемых ресурсов и передачи подтверждения. Затем rотовый пакет может быть передан достаточно быстро. Если общее время установки Tu, объем пакета p бит, а пропускная способность b бит/с, то время ожидания в одну сторону будет Tu + p/b. При дуплексной схеме минимальное время ожидания для передачи пакета объемом p бит и получения ответа при такой же пропускной способности составит T и + 2 p/Ь секунд.

Пакетная коммуrация Не требует посылки пробноrо пакета в пункт назначения, но для нее необходимо некоторое время Tk на коммуrацию пакета. Torда время передачи пакета в одну сторону будет Tk + p/Ь, причем за это время пакет дойдет только до nepBoro коммуттора, проход через который потребует задержки Td, после чеrо пакет переходит в очередной KOMMyraTop. Td включает время обработки пакета и время освобождения выходноrо порта. Если KOMMyraToroB n, то общее время ожидания в одну сторону будет Tk + n (P/Ь + T d) + p/b, при этом последнее слаrаемое отражает время копирования пакета из по.... следнеrо KOMMyraTopa в пункт назначения.

В случае **коммутации без буферизации пакетов** (червоточина) время передачи пакета в наилучшем случае будет приближаться к Tk+ p/Ь, ибо здесь отсyrcTByeT как посылка пробных пакетов для установки схемы, так и задержка промежуrочноrо хранения. Задержкой на распространение сиrнала можно пренебречь, поскольку она незначительна.

Очень важное значение имеет возможность быстрой передачи больших объемов информации. Это свойство отражает **суммарная пропускная способность**, определяемая суммированием пропускной способности всех каналов связи.

Наибольшее ускорение возможно при решении N-объектных зацач. Например, задачу инвертирования матриц нельзя ускорить более чем в пять раз вне зависимости от числа процессоров. И для этоrо имеется ряд причин.

Во-первых, все проrраммы содержат последовательную часть считывание данных, инициализация, интеrрирование результата. Пусть линейная часть проrраммы есть f, тоrда параллельная будет 1-f. Если на однопроцессорном компьютере время решения задачи T, то общее время решения зацачи не может быть лучше fT + (1 -f T)/n. Только при f = 0 возможно линейное ускорение. Это ЯRЛение называют законом Амдала.

ВО-ВТОрЫХ, невозможность линейноrо ускорения связана со временем ожидания в коммуникациях, втретьих, оказывает влияние оrраниченная пропускная способность, в-четвертых неэффективные алrоритмы, в-пятых непроизводительные зацержки для запуска большоrо количества процессоров. для достижения высокой (требуемой) производительности можно просто увеличивать количество процессоров даже при их неэффективном использовании. Если увеличение количества процессоров ведет к ПОВЫшению производительности системы, то систему называют расширяемой.

40. АЛГОРИТМЫ ВЫБОРА МАРШРУТОВ ДЛЯ ДОСТАВКИ СООБЩЕНИЙ.

В любой сети межсоединений с размерностью один и выше можно выбирать, по какому пути передавать пакеты от одного узла к другому. Часто существует множество возможных маршрутов. Правило, определяющее, какую последовательность узлов должен пройти пакет при движении от исходного пункта к пункту назначения, называется **алгоритмом выбора маршрута.**

Хорошие алгоритмы выбора маршрута необходимы, поскольку часто свободными оказываются несколько путей. Хороший алгоритм поможет равномерно распределить нагрузку по каналам связи, чтобы полностью использовать имеющуюся в наличии пропускную способность. Кроме того, алгоритм выбора маршрута помогает избегать взаимоблокировки в сети межсоединений. Взаимоблокировка возникает в том случае, если при одновременной передаче нескольких пакетов ресурсы затребованы таким образом, что ни один из пакетов не может продвигаться дальше и все они блокируются навечно.

Алгоритмы выбора маршрута можно разделить на две категории: **маршрутизация от источника** и **распределенная маршрутизация.**

При маршрутизации от источника источник определяет весь путь по сети заранее. Этот путь выражается списком из номеров портов, которые нужно будет использовать в каждом коммутаторе по пути к пункту назначения. Если путь проходит через к коммутаторов, то первые к байтов в каждом пакете будут содержать к номеров выходных портов, 1 байт на каждый порт. Когда пакет доходит до коммутатора, первый байт отсекается и используется для определения выходного порта Оставшаяся часть пакета затем направляется в соответствующий порт. После каждого транзитного участка пакет становится на 1 байт короче, показывая новый номер порта, который нужно выбрать в следующий раз.

При распределенной маршрутизации каждый коммутатор сам решает, в какой порт отправить каждый приходящий пакет. Если выбор одинаков для каждого пакета, направленного к одному и тому же конечному пункту, то маршрутизация является **статической**. Если коммутатор при выборе принимает во внимание текущий трафик, то маршрутизация является **адаптивной.**

Популярным алгоритмом маршрутизации, который применяется для прямоугольных решеток с любым числом измерений и в котором никогда не возникает тупиковых ситуаций, является **пространственная маршрутизация.** В соответствии с этим алгоритмом пакет сначала перемещается вдоль оси *x* до нужной координаты, а затем вдоль оси *y* до нужной координаты и т. д. (в зависимости от количества измерений). Например, чтобы перейти из (3,7, 5) в (6,9, 8), пакет сначала должен переместиться из точки x=3 в точку x=6 через (4, 7, 5), (3, 7, 5) и (6,7, 5). Затем он должен переместиться по оси *y* через (6, 8, 5) и (6, 9, 5). Наконец, он должен переместиться по оси z в (6,9,6), (6,9,7) и (6,9,8). Такой алгоритм предотвращает тупиковые ситуации.

41. МЕТОДЫ СИНХРОНИЗАЦИИ ПРОЦЕССОВ.

Процессы, выполняемые в мультипрограммном режиме, можно рассматривать как набор последовательных слабосвязанных процессов, которые действуют почти независимо друг от друга, лишь изредка используя общие ресурсы. Взаимосвязь между такими процессами устанавливается с помощью различных сообщений и так называемого механизма синхронизации, который позволяет согласовывать и координировать работу процессов.

Хотя каждый процесс, выполняемый в мультипрограммном режиме, имеет доступ к общим ресурсам, существует некоторая область, которую в фиксированный момент времени может использовать лишь один процесс. Нарушение этого условия приведет к неизвестному порядку обработки процессов. Назовем такую область критической. При использовании критической области возникают различные проблемы, среди которых можно выделить проблемы состязания (гонок) и тупиков (клинчей).

Условие состязания возникает, когда процессы настолько связаны между собой, что порядок их выполнения влияет на результат операции.

Условие тупиков появляется, если два взаимосвязанных процесса блокируют друг друга при обращении к критической области.

В системах, допускающих перераспределение любых ресурсов в произвольной последовательности, но имеющей и не освобожденные ресурсы, время от времени должны возникать тупиковые ситуации.

Пример. Пусть программе А нужен ресурс R1. Она запрашивает его и получает. Программе В нужен ресурс R2. Она запрашивает его и получает . Далее, пусть программа А, не отпуская R1, запрашивает R2, а программа В, не отпуская R2, запрашивает R1. Налицо типичный клинч, если только один из ресурсов R1 или R2 не может быть освобожден до момента завершения обработки соответствующего процесса.

Простейший прием разрешения подобной проблемы - стандартные операции типа WAIT (ЖДАТЬ) и SIGNAI, (ОПОВЕСТИТЬ). Операция WAIT позволяет: временно заблокировать процесс, а SIGNAL информирует систему о необходимости разблокирования процесса, задержанного из-за невыполнения условия. Следует отметить такие приемы, как БЛОКИРОВКА ПАМЯТИ (для реализации взаимного исключения одному процессу разрешается выполнить операцию над памятью, а другому ждать, пока первый не завершит работу); ПРОВЕРКА и УСТАНОВКА (аппаратная операция, к которой обращаются с двумя параметрами: ЛОКАЛЬНЫЙ И ОБЩИЙ). Наиболее эффективным и простым средством синхронизации процессов, исключающим состояние “активного" ожидания, является семафор.

42.УРОВНИ ПАРАЛЛЕЛИЗМА. НАПРАВЛЕНИЕ ИССЛЕДОВАНИЙ В ОБЛАСТИ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ.

Говорят, что задача обладает естественным параллелизмом если она сводится к операциям над многомерными векторами или над матрицами либо над другими аналогичными объектами.

Параллелизм множества объектов представляет собой частный случай естественного параллелизма. Задача состоит в обработке информации о различных, но однотипных объектах по одной и той же или почти по одной и той же программе. Здесь сравнительно малый вес занимают так называемые интегральные операции.

Параллелизм независимых ветвей -- количество независимых частей (участков, ветвей) задачи, которые при наличии в ВС соответствующих средств могут выполняться параллельно (одновременно одна с другой).

Ветвь программы У не зависит от ветви Х, если:

- между ними нет функциональных связей, т.е. ни одна из входных переменных ветви У не является выходной переменной ветви Х либо какой-нибудь ветви, зависящей от Х;
- между ними нет связи по рабочим полям памяти;
- они должны выполняться по разным программам;
- независимы по управлению, то есть условие выполнения ветви У не должно зависеть от признаков вырабатываемых при выполнении ветви Х или ветви, от нее зависящей.

Параллелизм смежных операций. Суть его состоит в следующем.

Если подготовка исходных данных и условий исполнения i-й операции заканчивается при выполнении (i- k)-й операции (где k = 2, 3, 4 ...), то i-ю операцию можно совместить с (i-k+1)-й, (i-k+2)-й, ..., (i-1)-й.

В последние годы интерес к параллельным вычислениям концентрируется на осмыслении общих закономерностей разработки параллельных алгоритмов. Работы в указанной области условно можно разбить на три основных направления.

1. Разработка моделей параллельных вычислений и установление соотношений между ними, определение класса задач, допускающих эффективное распараллеливание (класс NC), и класса задач, наиболее трудных для распараллеливания (P-полных задач).

2. Разработка общих методов построения параллельных алгоритмов и распараллеливание последовательных алгоритмов. Исследование сложности параллельных алгоритмов в различных областях применений.

3. Создание методов и средств отображения параллельных алгоритмов на реальные параллельные архитектуры.

43. ЯЗЫКИ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ.

Р-язык

Одним из первых ЯПП был матричный Р-язык. Программа, записанная на этом языке, представляет собой матрицу, элементами которой являются операторы, среди которых имеются операторы настройки, производящие изменение коммутаторов элементарных машин (ЭМ) либо параметров, управляющих структурой ЭМ; операторы обмена информацией между ЭМ и "пустые" операторы, пропускающие выполнение одного шага вычислений. Каждый столбец матрицы включает независимые операторы, которые можно выполнять параллельно. Процесс выполнения программы сводится к последовательному прохождению всех столбцов матрицы.

Основной принцип последовательного программирования - явное указание порядка выполнения операций - в Р-языке остается, однако здесь вводится двухкоординатная система записи, где одна координата указывает последовательность выполнения операций во времени, другая - распределение операций между ветвями вычислений. Р-язык, однако, не позволяет сохранить природную асинхронность задач и требует учитывать структуру конкретной вычислительной системы.

ЯПФ-язык

Более удобным в этом плане является язык ярусно-параллельных форм (ЯПФ). Программа на ЯПФ представляет собой мультиграф, задаваемый, например, в матричной форме. Вершинами мультиграфа являются операторы, мультиграф не имеет контуров. Дуги данного мультиграфа бывают двух типов: информационные и управляющие. При выполнении программы на ЯПФ первоначально выделяется нулевой ярус (множество операторов, не имеющих непосредственных предшественников по информационным дугам) и инициируется выполнение операторов из этого множества. После их выполнения убираются все информационные дуги, выходящие из нулевого яруса, и в оставшемся графе таким же образом выделяется первый ярус.

Процесс продолжается от яруса к ярусу. После выполнения управляющего оператора, из которого выходят управляющие дуги, происходит выбор одного из операторов, в которые заходят эти дуги. Процесс обработки ЯПФ-программы завершается, когда выполнены все операторы из яруса с наибольшим номером. Однако представление программы в ЯПФ требует "развертки" циклических участков программы, что не позволяет использовать ее как язык практического параллельного программирования.

К-язык

В качестве языка параллельного программирования был разработан так называемый К-язык. Суть языка: задается множество элементарных операторов и множество порождающих правил построения алгоритмов, образующие некоторое исчисление. Запись вывода в этом исчислении и является К-программой. Имеются три типа основных порождающих правил: суперпозиция, дизъюнкция, рекурренция. Правило суперпозиции означает, что результаты

выполнения операторов b_1, b_2,.... b_k служат аргументами для некоторого оператора B. Порядок выполнения операторов b_1, b_2,.... b_k безразличен, что и порождает параллелизм в их выполнении. Правило дизъюнкции задает ветвление в К- программе, а правило рекурренции - цикличность. К-язык относится к языкам асинхронного типа.

Язык OCCAM

Для многотранспьютерных систем английскими учеными был создан специальный язык параллельного программирования OCCAM. В отличие от ЯПП, например, Concurent Fortran или Concurent Pascal, он не построен путем расширения известных последовательных ЯП.

Основным понятием языка является процесс (примитивный, составной, именованный), напоминающий процедуры и подпрограммы обычных языков программирования. Простейшим примитивным процессом является процесс SKIP. Если именованный процесс имеет список формальных параметров, то они следуют в описании процесса после его имени с указанием ключевых слов VALUE, VAR и CHAN, обозначающих их природу: соответственно значение, переменная или канал.

Процессы взаимодействуют с помощью каналов и могут выполняться как последовательно, так и параллельно.

Последовательный процесс задается перечислением за ключевым словом SEG всех его компонент, которые должны выполняться последовательно.

Для описания параллельного процесса после ключевою слова PAR задаются необходимые составляющие его процессы. Выполнение параллельной композиции (PAR- процесса) состоит в выполнении каждой его компоненты до полного их завершения.

Выбор процесса для исполнения в зависимости от готовности других процессов задается через ALT-процесс.

В языке OCCAM существует только один вид структурированных данных - одномерный массив.

44. ПРЕОБРАЗОВАНИЕ ПОСЛЕДОВАТЕЛЬНЫХ ПРОГРАММ В ПОСЛЕДОВАТЕЛЬНО-ПАРАЛЛЕЛЬНЫЕ

Задача распараллеливания. В любом полном рассмотрении вопросов параллельного программирования должны быть представлены способы получения параллельной программы из обычной алгоритмической записи, которая несет в себе след чисто последовательной логики рассуждений, так как особые вычисления, проводимые вручную и автоматически, выполнялись последовательно. Однако оказывается, что любой реальный процесс при детальном его рассмотрении выполняется параллельно.

Как правило, основное время выполнения программ на обычных языках связано с реализацией циклов. Поэтому важной, задачей является распараллеливание циклов.

Рождается актуарная проблема - преобразование последовательных программ в последовательно-параллельные. Но так как эта работа необычайно трудоемка, ее следует автоматизировать. Решение указанной проблемы позволяет, во-первых, высвободить большие трудовые ресурсы, во-вторых, использовать накопленный за долгие годы развития ЭВМ информационно-программный багаж без ручного перепрограммирования и, в-третьих, осуществить переход на многопроцессорные системы с меньшими затратами труда и времени.

Распараллеливание программ можно осуществлять как на уровне отдельных задач, так и на уровне отдельных процедур, операторов, операций и микроопераций. Целесообразность преобразования на указанных уровнях должна решаться в каждом отдельном случае в зависимости от структуры ВС, типа программы и цели, которая славится при ее решении.

Рассмотрим общую схему преобразования последовательных программ в последовательно-параллельные. Организовать параллельные вычисления можно с помощью некоторой формальной процедуры, выполняемой автоматически над каждой программой, состоящей из последовательности операторов обычного ЯП. Выявление параллелизма заключается в разбиении всех операторов программ на два класса: тех, которые могут выполняться параллельно, и тех, которые должны быть завершены прежде, чем следующие последовательности операторов начнут выполняться.

Такой подход к распараллеливанию основывается на представлении программы в виде ориентированного графа G, множество вершин которого V = {v1, v2,..., vn} соответствует либо отдельным операторам программы, либо совокупности этих операторов (типа подпрограмм в языке ФОРТРАН либо процедур в языке АЛГОЛ или ПАСКАЛЬ). Множество направленных дуг U = {u1, u2, .., um} соответствует возможным переходам между операторами программы.

Построим схему алгоритма распараллеливания программ, используя некоторые сведения из теории графов.

Схема алгоритма

1)Представление исходной программы в виде графа

2)Сведение циклического графа к ациклическому.

3)Наложение информационных связей, заданных между операторами программы, на связи возможных переходов.

4) *Распределение вершин графа по уравнениям готовности.*

5) Формирование ветвей решения.

В результате указанных действий каждый уровень разбивается на несколько подуровней и производится их согласование. Операторы исходной программы, окончательно распределенные по уровням готовности, объединяются в ветви решения, что и является окончательным результатом рассматриваемой процедуры распараллеливания.

45. ПЛАНИРОВАНИЕ В МУЛЬТИСИСТЕМАХ.

Все задания (процессы), находящиеся в системе с мультипрограммированием, конкурируют из-за процессорного времени. Кроме процессов пользователя, имеются и системные процессы, для выполнения которых нужно процессорное время. Имеются, как правило, две очереди процессов:

- **очередь готовых процессов** - активные процессы, ожидающие кванта времени ЦП (Ог);
- **очередь заблокированных процессов**, ожидающих выполнения какого-нибудь события (Об).

Каждый процесс P_i может находиться в одном из состояний: **готовом, заблокированном** или **выполняемом**. Если pi принадл Ог и ему выделен квант t_{Pi} процессорного времени, то он будет выполняться, пока не произойдет одно из событий:

- истечет его квант времени;
- процесс будет заблокирован;
- процесс будет вытеснен более приоритетным процессом;
- процесс будет завершен.

Если произошло одно из указанных событий, процесс переводится либо в очередь Ог, либо в очередь Об, либо оставляет систему. По окончании блокировки процессы переводятся из очереди Об в Ог. Алгоритм планирования для такой системы включает в себя способ выбора готового процесса из заданной совокупности процессов и способ вычисления величины кванта t_{pi} для него.

Каждому p_i из Ог поставлен в соответствие приоритет - целое число, учитывающее важность процесса p_i, занимаемый им объем памяти, срочность выполнения и объем операций I/O, а также внешний приоритет, назначаемый пользователем.

Планирование по наивысшему приоритету

В этом методе HPF (highest priority first) процессор предоставляется тому процессу, который имеет наивысший приоритет. И если не допускается вытеснение процесса, то он выполняется до тех пор, пока не выполнится или не будет заблокирован. Если вытеснение разрешено, то поступивший процесс с более высоким приоритетом прервет выполняющийся процесс и ему будет передано управление на выполнение. Вытесненный процесс перейдет в очередь готовых процессов. Если процессор освободился, то ему из Or назначается процесс с наивысшим приоритетом.

Если очередь От рассортирована по приоритетам, то выбрать первый элемент просто. Если же такой сортировки нет (а при динамическом изменении приоритетов ее всегда нет), необходимо по истечении некоторого времени T проводить сортировку всей очереди Or или его некоторой части. Здесь возможны различные модификации.

Метод круговорота (карусель)

Простой круговорот (round robin. RR) не использует никакой информации о приоритетах. Порядок обслуживания процессов следующий: первый пришедший процесс получает квант времени t_0 и встает в очередь па обслуживание, если только он себя не заблокирует. Поэтому при наличии N процессов в системе для обслуживания будем считать, что скорость работы процессора равна V/N, где V - истинное быстродействие процессора. Если to -> бесконечность, то стратегия SRR (simple round robin) сводится к стратегии FIFO (first in first out). С уменьшением t_0 улучшается обслуживание коротких процессов. Но t_0 не должно быть слишком мало: если оно становится сравнимым по порядку со временем переключения с одного процесса на другой, то задержка в выполнении процессов становится заметной.

Очереди с обратной связью

Основной алгоритм FB (feedback) очередей с обратной связью использует n очередей, каждая из которых обслуживается в порядке поступления. Новый процесс поступает в первую очередь, затем после получения кванта времени он переходит в очередь со следующим номером и так далее после очередного кванта времени. Время процессора планируется таким образом, что он обслуживает непустую очередь с наименьшим номером. В методе FB каждый, вновь поступающий на обслуживание процесс получает (в неявном виде безусловно и в соответствии со стратегией обслуживания) высокий приоритет и выполняется подряд в течение такого количества квантов времени, пока не придет следующий процесс. Но если приход нового процесса задерживается, то текущий процесс не может проработать большее количество квантов, чем предыдущий процесс.

Многоуровневое планирование

Метод многоуровневого планирования сохраняет всю информацию о процессах, выполняет перевычисления приоритета и т. д., но делает это не очень часто.

В основе этого метода лежит следующий принцип - операции, которые встречаются часто, должны требовать меньше времени, чем те, которые встречаются редко. С этой целью все операции в зависимости от частоты выполнения разбиваются на уровни.

46. КЛАССИФИКАЦИЯ КОМПЬЮТЕРОВ

В настоящее время существует классификация многопроцессорных систем по Флинну, базирующаяся на порядке поступления потоков команд и данных на обработку. Их структуры образуют следующие четыре класса:

1) **одиночный поток команд – одиночный поток данных** (SISD). Сюда относятся типичные машины последовательного действия;

2) **одиночный поток команд – множественный поток данных** (SIMD). Это матричные и ассоциативные структуры;

3) **множественный поток команд – одиночный поток данных** (MISD). В данный класс включаются конвейерные или магистральные структуры;

4) **множественный поток команд – множественный поток данных** (MIMD). К этому классу принадлежат структуры, содержащие несколько процессоров, одновременно выполняющих различные фрагменты одной и той же программы.

Классификация Флинна, как видно из приведенного, базируется лишь на множестве потоков команд и данных для обработки и не является классификацией внутренней организации (архитектуры) компьютера. В отличие от Флинна **систематика Шора** основана на формировании компьютера из различных компонент. Выделяется шесть типов компьютеров со сквозной нумерацией.

Ml - одно устройство управления (УУ), обработчик (УО), память команд (ПК) и память данных (ПД). Устройство обработки может содержать множество функциональных устройств как векторного, так и скалярного типа. В связи с этим к классу М1 относят конвейерную векторную (CRAY--I) и конвейерную скалярную (СОС 7600) вычислительные системы.

М2 – здесь чтение разрядов данных из ПД осуществляется (в отличие от М1) из одноименных разрядов всех слов памяти. Примером могут служить системы DAP и STARAN.

М3 - считывание и обработка данных осуществляется в двух измерениях: по вертикальным и горизонтальным срезам памяти. Примером может служить ортогональная машина Шумана и OMEN-60.

М4 - включает многократно дублирующие ПД и УО, образующие процессорные элементы (ПЭ), а команды для исполнения поступают из единственного УУ. Примером этого класса может служить система PEPE без связи между отдельными ПЭ.

М5 - это тип М4 со связями между каждым ПЭ и его ближайшим соседом. К этому классу относится система ILLlAC IV.

М6 - этот тип представляет собой матрицу с функциональной логикой. Сюда можно отнести различные ассоциативные ансамбли запоминающих устройств и процессоров.

47. МАТРИЧНЫЕ КОМПЬЮТЕРЫ. АРХ-РА ТИПА ГИПЕРКУБ.

Матричный процессор (array processor) состоит из большого числа сходных процессоров, которые выполняют одну и ту же последовательность команд применительно к разным наборам данных. Первым в мире таким процессором был ILLIAC IV (Университет Иллинойс). Первоначально предполагалось сконструировать машину, состоящую из четырех квадрантов, каждый из которых содержал матрицу размером 8 x 8 из блоков процессор/память. Для каждого квадранта имелся один блок контроля. Он рассылал команды, которые выполнялись всеми процессорами одновременно, при этом каждый процессор использовал собственные данные из собственной памяти (загрузка данных происходила при инициализации). Это решение, значительно отличающееся от стандартной фон-неймановской машины, иногда называют архитектурой SIMD (Single Instruction-stream Multiple Data-stream — один поток команд с несколькими потоками данных). Из-за очень высокой стоимости был построен только один такой квадрант, но он мог выполнять 50 млн операций с плавающей точкой в секунду. Если бы при создании машины использовалось четыре квадранта, она могла бы выполнять 1 млрд операций с плавающей точкой в секунду.

АРХИТЕКТУРА ТИПА ГИПЕРКУБ.

Гиперкуб представляет собой архитектуру с большим числом процессоров (2^n), связанных в систему таким образом, что каждый из них можно представить одной из вершин n-мерного двоичного гиперкуба. Архитектура отличается регулярностью и простотой. Каждый процессор связан только с n другими соседними, и при этом переход между любой парой процессоров – не более чем через n ребер.

Гиперкубы размерностью 0, 1, 2, 3, 4 показаны на рис. 1-4. Ясно, что двоичный n-мерный куб регулярным образом связывает 2n процессоров, представляемых вершинами, через n×2^(n-1) связей, представляемых ребрами, причем каждая вершина связана с n соседними. Максимальный кратчайший путь между вершинами имеет всего log2n ребер.

Примером компьютера, использующего архитектуру гиперкуба, может служить построенная фирмой Thinking Machine параллельная ЭВМ с общим управлением Connection Machine-2, включающая 65536 процессорных элементов.

Можно получить различные схемы связи, убирая те или иные ребра. Например преобразовать архитектуру четырехмерного гиперкуба в трехмерную сетку. Следует отметить, что трехмерная сетка не имеет внутренних вершин. Угловые вершины связаны ребрами с тремя соседними вершинами, а остальные – с четырьмя вершинами.

Архитектура типа гиперкуб широко используется в компьютерах, предназначенных для задач обработки текстов. В частности, используя их, решают задачи генерирования знаний и подготовки аннотаций к текстам статей и книг.

48. ВС ПАРАЛЛЕЛЬНОГО ДЕЙСТВИЯ (ВКЛЮЧАЯ ЭЛЬБРУС).

При разработке многопроцессорного вычислительного комплекса (МВК) Эльбрус ставились следующие задачи:

-повысить эффективность использования оборудования;

-обеспечить возможность достижения предельной производительности;

-создать высоконадежные резервируемые структуры, обладающие возможностью постепенного наращивания производительности с учетом адаптации к решаемым задачам.

В состав семейства МВК Эльбрус входит система Эльбрус--I производительностью от 1,5 миллиона операций в секунду до 10 миллионов операций в секунду и Эльбрус2 с суммарным быстродействием 120 миллионов операций в секунду.

Структура вычислительного комплекса МВК характеризуется следующими особенностями. Все процессоры комплекса имеют одну и ту же систему команд и одинаковую по функциям операционную систему ЕОС (Единую ОС).

Основными модулями системы являются:

-центральные процессоры (ЦП) в количестве от 1 до 10;

-модули оперативной памяти (от 4 до 32) объемом от 576 Кбайт до 4608 Кбайт;

-модули процессоров ввода-вывода (ПВВ) (от 1 до 4);

-модули процессоров передачи данных (ППД) (от 1 до 16);

-модули управления накопителями на магнитных барабанах и дисках, образующие систему управления массовой памятью.

Оперативная память для всех процессоров системы доступна через коммутатор, на который возлагаются функции замены неисправных блоков резервирования (рис. 9.3). Аппаратный контроль охватывает не только работу процессоров, но и работу по обмену информацией на всех уровнях. Система команд центрального процессора базируется на принципе магазинного обращения к памяти с аппаратной реализацией стека. Внутренний язык машины подобен польской инверсной записи (ПОЛИЗ). В вершине стека могут находиться не только сами операнды, но и ссылки на них, а также ссылки на процедуры вычисления операндов. По принципу построения СК для ЦП МВК Эльбрус близка к СК KDF--9 и ЭВМ фирмы Burroughs. Однако МВК Эльбрус имеет более развитый аппарат описания типов данных, их защиты, способов распределения памяти, а также развитый аппарат дескрипторов. Каждый объект данных в памяти снабжен дополнительным управляющим разрядом (TeroM), в котором содержится информация о типе данных (целое, вещественное, набор, дескриптор, адрес, метка, формат) и различные управляющие признаки, включая признаки защиты по чтению и записи. Все это позволяет строить чистые (иногда их называют реентерабельные, повторно входимые) процедуры, не имеющие явных ссылок на адреса объектов в лотческой или физической памяти. Это очень важно, ибо позволяет одно и то же тело процедуры использовать разными

процессорами над разными данными. Аппарат дескрипторов и косвенных ссылок позволяет разным программам обращаться к общим данным.

Многие функции синхронизации процессов реализованы аппаратно(используется аппарат семафоров). Модуль ПВВ (процессор ввода/вывода) это, по сути, специальная ЭВМ с локальной паWlТЬЮ и доступом к основной оперативной памяти. Он служит для управления связью системы с внешними устройствами. В состав ПВВ входят блоки быстрых каналов, состоящие из 4 селекторных каналов, каждый из которых может обслужить до 64 быстрых абонентов, и блоки стандартных каналов, каждый из которых в свою очередь содержит 16 каналов, обслуживающих до 256 внешних абонентов. Стандартный канал обеспечивает мультиплексное обслуживание медленных абонентов. Кроме того, в состав ПВВ входит блок сопряжения с процессорами передачи данных (до 4 каналов). Основное назначение ПВВ - освободить ЦП от функций организации очередей обмена, реакций на прерывание по 1/0, а также от оптимизации обслуживания очередей запросов на обмен. Логическая память разбивается на страницы длиной в 512 слов. В основу программного обеспечения положен принцип достижения высокой эффективности вычислений при помощи гибкости и адаптируемости МВК. Здесь МВК и его ПО как бы адаптируются к решаемым задачам, а не наоборот, как сделано во многих системах, когда алгоритм решаемой задачи модифицируется, чтобы его можно было реализовать в рамках жесткой ВС. МВК Эльбрус может выполнять программы пользователей БЭСМ6 благодаря спецпроцессору, реализующему систему команд БЭСМ 6. Модули ЦП, ОП и процессоров 1/0 связаны между собой при помощи центрального коммутатора (К). Процессоры приема/передачи данных, промежуточная и внешняя память, а также устройства 1/0 под ключаются к центральной части системы через процессоры 1/0. Модули системы работают параллельно и независимо друг от друга, ресурсы системы динамически распределяются ЕОС, которая обеспечивает мультипрограммный режим пакетной обработки, режим раз деления времени и терминальную обработку.Для повышения быстродействия аппаратно реализованы базовые конструкции и общепринятые механизмы воплощения ЯП, лежащие в основе систем программирования.

49. СУПЕРКОМПЬЮТЕРЫ.

Суперкомпью́тер — специализированная вычислительная машина, значительно превосходящая по своим техническим параметрам и скорости вычислений большинство существующих в мире компьютеров.

В общеупотребительный лексикон термин «суперкомпьютер» вошёл благодаря распространённости компьютерных систем Сеймура Крэя, таких как, CDC 6600, CDC 7600, Cray-1, Cray-2, Cray-3 (*англ.*) и Cray-4 (*англ.*). Сеймур Крэй разрабатывал вычислительные машины, которые по сути становились основными вычислительными средствами правительственных, промышленных и академических научно-технических проектов США .

На сегодняшний день суперкомпьютеры являются уникальными системами, создаваемыми «традиционными» игроками компьютерного рынка, такими как IBM, Hewlett-Packard, NEC и другими, которые приобрели множество ранних компаний, вместе с их опытом и технологиями. Компания Cray по-прежнему занимает достойное место в ряду производителей суперкомпьютерной техники.

В настоящее время суперкомпьютерами принято называть компьютеры с огромной вычислительной мощностью («числодробилки» или «числогрызы»). Такие машины используются для работы с приложениями, требующими наиболее интенсивных вычислений (например, прогнозирование погодно-климатических условий, моделирование ядерных испытаний и т. п.), что в том числе отличает их от серверов и мэйнфреймов (англ. *mainframe*) — компьютеров с высокой общей производительностью, призванных решать типовые задачи.

Cray-1 – векторный суперкомпьютер, который позволяет осуществлять параллельную обработку. Tantie-2 (Китай) – на тесте Linpach производительность – 33,86 PF (петафлокс); теоретическая (пиковая) – 54,9 PF. Построен на микропроцессорах Intel Xeon (12 ядер 2,2 ГГц). Общее число ядер 3120000. ОС Kylin Linux.

Titan. Разработан на базе Cray (модиф. Jaguar). 18688 различных узлов, построенных на микропроцессорах AMD Opteron (16 ядер), включает графические процессоры Nvidia Tesla. Технология CUDA. Ускорители Nvidia обеспечивают ~80-85% общей производительности СК. Используется для решения задач квантовой химии, физики высоких энергий, генетики.

Sequoia. Построен на технологии Blue Gene. Способен достигать производительности 20 PF. ОС – Линукс.

K computer (Japan) – 10PF

Ju Queen (Германия) – 5.9 PF

СКИФ (Беларусь + Россия) ОС Линукс

Суперкомпьютеры используются во всех сферах, где для решения задачи применяется численное моделирование; там, где требуется огромный объём сложных вычислений, обработка большого количества данных в реальном времени, или решение задачи может быть найдено простым перебором множества значений множества исходных параметров.

50. КОММУНИКАЦИОННЫЕ ТЕХНОЛОГИИ.

Коммутационные среды обычно включают адаптеры вычислителей и коммутаторы, устанавливающие соединения между ними. Различают коммутаторы простые и составные, компонуемые из простых. Простые коммутаторы имеют малую временную задержку при установлении полнодоступных соединений, но в силу физических ограничений могут быть применены только для систем с малым числом вычислителей. Если количество входов и выходов большое, то создаются составные коммутаторы, состоящие из простых путем объединения их в многокаскадные схемы линиями «точка--точка). В настоящее время коммутаторы, как правило, используют в качестве базовой одну из следующих схем:

-коммутационная матрица,

-разделяемая многовходовая память,

-общая шина.

Иногда эти способы взаимодействуют в одном коммутаторе.

Коммутаторы вычислительных систем

Коммутаторы на основе коммутационной матрицы обеспечивают основной и самый быстрый способ взаимодействия портов процессоров. Такой способ взаимодействия был реализован в первом промышленном коммутаторе. Однако реализация матрицы возможна лишь для ограниченного числа портов. Известно, что сложность схемы возрастает пропорционально квадрату количества портов коммутатора.

Коммуникационная среда SCI

Масштабируемый когерентный интерфейс (ScaIable Coherent Interface) принят как стандарт ANSI/IEEE Std 15961992. Для обозначения этого стандарта общепринято сокращение SCI. SCI предусматривает реализацию когерентности посредством стандартно организованной кэш памяти, размещаемой в узле SCI. Эта кэш-память располагается в интерфейсе между вычислительным модулем (ВМ) и узлом и вписывается в механизм реализации когерентности ВМ следующим образом:

-- если на предыдущих уровнях иерархии памяти ВМ обнаруживается отсутствие необходимых данных, то производится их поиск В кэшпамяти узла;

-- при нахождении данных, если последние состоятельны, их копия перемешается внутрь иерархии памяти ВМ. Если же данные находятся в состоянии модификации, запись осуществляется после ее завершения;

-- при отсутствии данных вырабатывается сигнал промаха, который активизирует адаптер интерфейса на выполнение действий по доставке данных с удаленных блоков памяти.

Коммуникационная среда MYRINET

Логический протокол Myrinet был развит в Caltech Submicron Systems Architecture Project. Эта среда стандартизует формат пакета, способ апресации ВМ, набор управляющих символов протокола передачи пакетов.

Коммуникационная среда образуется адаптерами шина компьютера - линк сети и коммуrаторами линков сети. Каждый линк содержит пару однонаправленных каналов, образуя дуплексный канал.

Возможные структуры вычислительных систем, построенных с использованием адаптеров и коммутаторов Myrinet, показаны на рис. 10.22.Линки передают пакеты заданной структуры, состоящие из заголовка данных пакета и концевика. Поток передачи по линку может быть остановлен получателем на заданный промежуrок времени, например на время подrотовки буфера приема. Если передающий адаптер заблокирован получателем на время большее, чем требует пересылка 222 байт, то этот адаптер посылает получателю сиrнал сброса. Плата адаптера встраивается в шину ВМ и подсоединяется к питанию компьютера. Допускается подключение произвольноrо числа адаптеров к шине в пределах ее коммуникационных возможностей. Адаптер содержит 128 Кбайт памяти, используемых как для хранения пакетов, так и для управляющей программы адаптера. Эта прогрaмма исполняется процессором LANai адаптера. Микропроцессор LANai реализован как заказная СБИС фирмой MYRICOM,распространяющей сеть Myrinet.