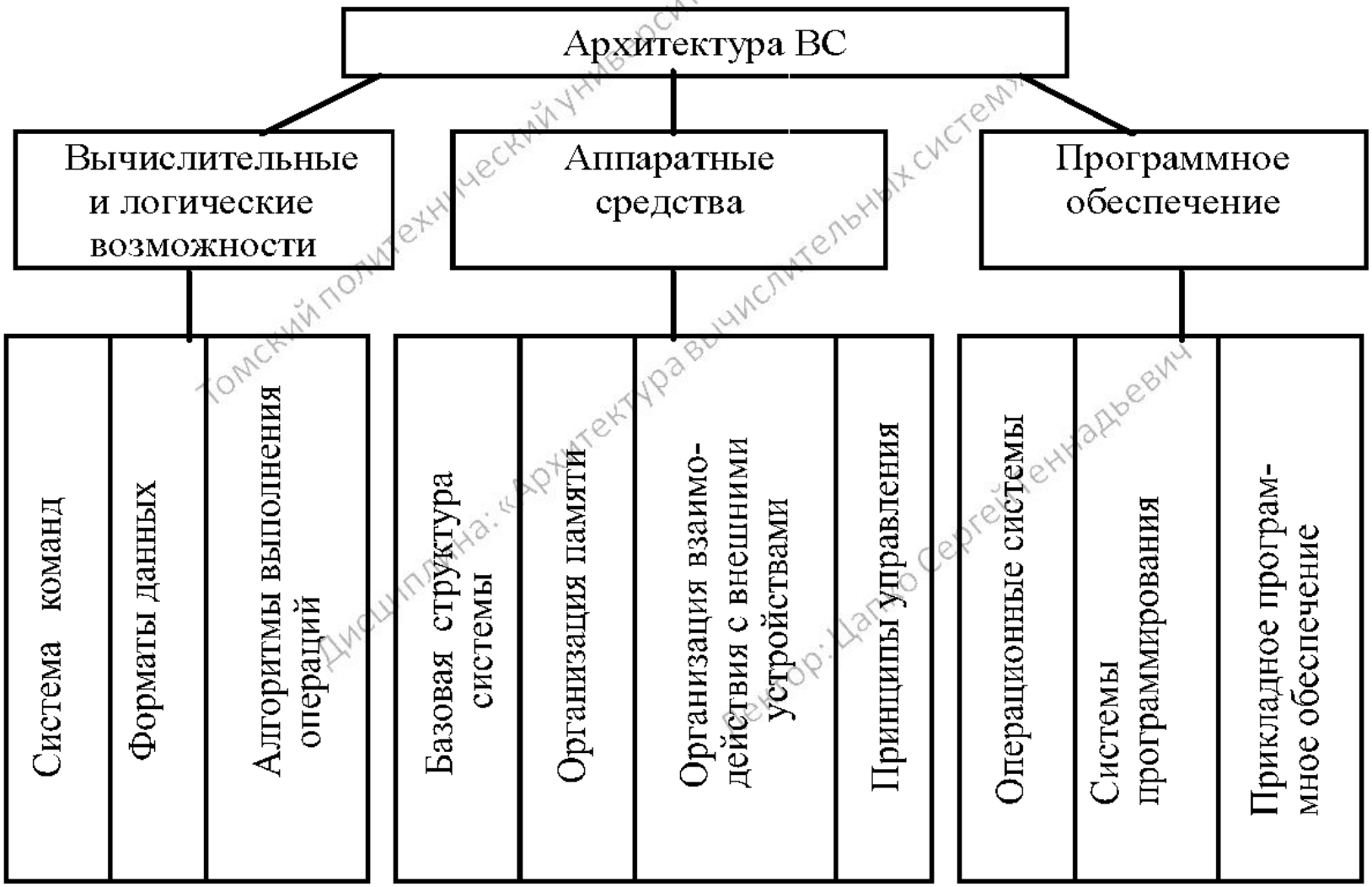
Архитектура вычислительных систем

(презентационные материалы к лекционным занятиям)

Лектор: Цапко Сергей Геннадьевич

Функциональные возможности ВС



Составные части понятия «архитектура»

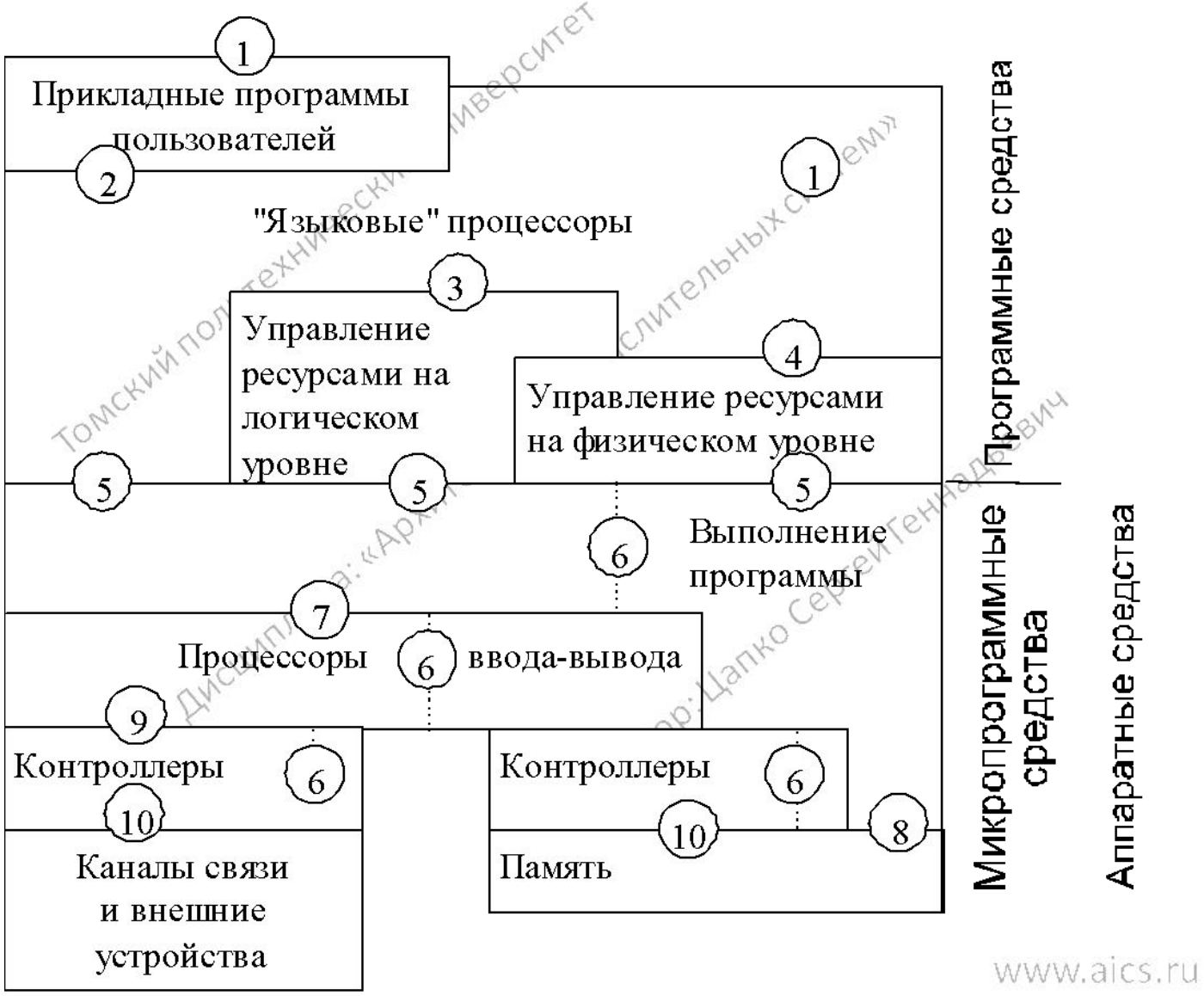
- *Вычислительные и логические возможности ВС.* Они обусловливаются системой команд (СК), характеризующей гибкость программирования, форматами данных и скоростью выполнения операций, определяющих класс задач, наиболее эффективно решаемых на ВС.

Классификация системы команд по назначению

Составные части понятия «архитектура»

- *Аппаратные средства.* Простейшая ВС включает модули пяти типов: центральный процессор, основная память, каналы, контроллеры и внешние устройства.
- *Программное обеспечение.* Оно является составной частью архитектуры компьютера и существенно влияет на весь вычислительный процесс, в частности позволяет эффективно эксплуатировать аппаратные средства системы.

Многоуровневая организация архитектуры ВС



Этапы разработки типовых проектов

(характерны для процесса разработки архитектуры ЭВМ)

- анализ требований, предъявляемых к системе;
- составление спецификаций;
- изучение известных решений;
- разработка функциональной схемы;
- разработка структурной схемы;
- отладка проекта;
- оценка проекта.

Конструкции языков программирования, вызывающие семантический разрыв

- Массивы (реализация принципа организации данных в виде массива возлагается на компилятор)
- Структуры (это тип организации данных в виде наборов разнородных элементов данных, что как правило не поддерживается ПЭВМ).
- Строки (допустимые операции: слияние, выделение заданной части строки, поиск строки по заданной подстроке, определение длины текущей строки, проверка присутствия одной строки в другой строке. Подобные возможности в системе команд многих процессоров)
- Процедуры (При вызове процедуры требуется сохранить состояние текущей процедуры, динамически назначить память для локальных переменных вызванной процедуры, передать параметры и инициализировать выполнение вызванной процедуры. Как правило, эти действия возлагаются на компилятор)

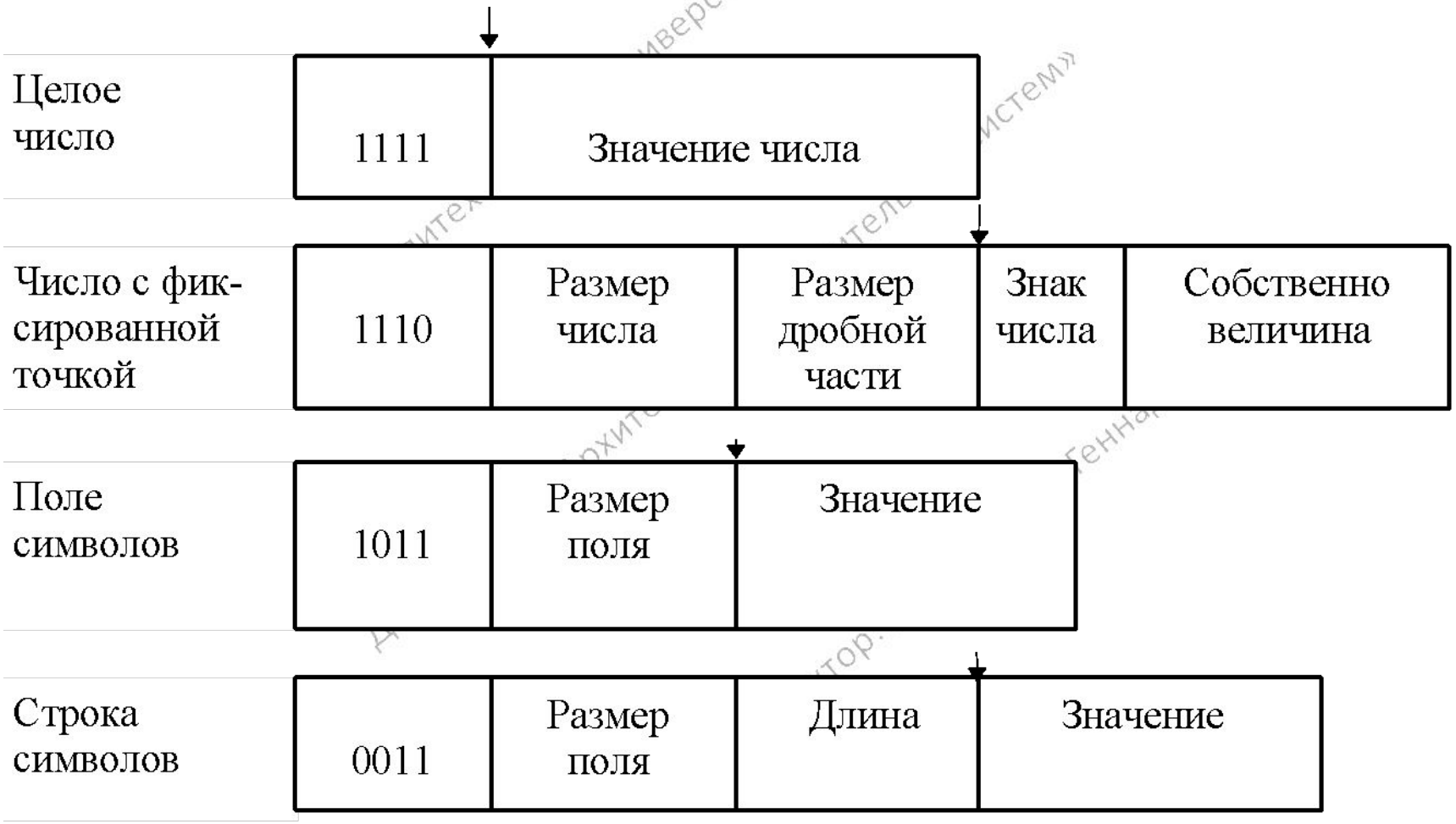
Основные характеристики архитектуры фон Неймановского типа

- последовательно адресуемая единственная память линейного типа для хранения программ и данных;
- 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;
- 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.

Требования ЯВУ к архитектуре ЭВМ

- память состоит из набора дискретных именуемых переменных.
- 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.

Примеры типов ячеек при теговой организации

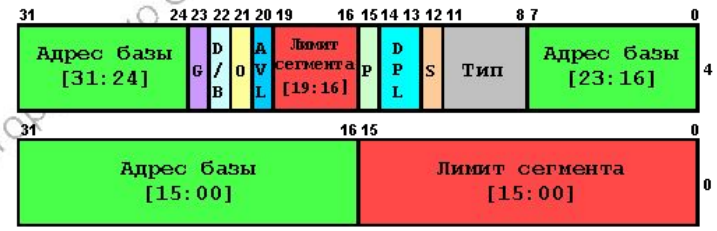


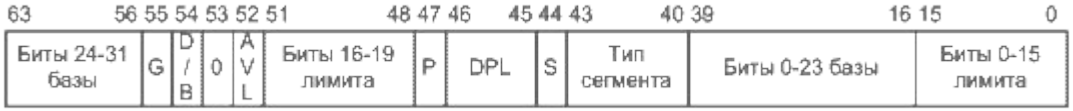
Пример дескриптора

- Основное отличие тегов и дескрипторов состоит в следующем: дескрипторы создают дополнительный уровень адресации, что требует увеличения затрат на формирование адреса, т. е. дескрипторы – это часть команды (программы), а теги – это часть данных.

101	P	I	R	W	Длина	Адрес

Здесь первые три бита содержат тег. Если значение его 101, то данное слово дескриптор. Бит P указывает, находятся данные в основной памяти или во вспомогательной; I указывает, одиночный ли элемент описывает данный дескриптор или весь массив; R идентифицирует непрерывную или разрывную область памяти; W означает, что разрешено только чтение данных.





Области санкционированного доступа

Временное расширение домена

Достоинства:

- улучшается отладка программ. Сфера действия любой ошибки ограничивается размерами домена, в котором она произошла, что увеличивает вероятность ее обнаружения;
- повышается надежность защиты программ. Информация, принадлежащая одной секции, защищается от воздействия других секций.

Одноуровневая память

Достоинства:

- сравнительно низкая стоимость программного обеспечения;
- независимость адресации от принципа организации памяти.

Трудности реализации:

- создание встроенного в архитектуру ЭВМ механизма иерархии ЗУ;
- восстановление памяти;
- переносимость объектов на другие системы с традиционной организацией архитектуры.

Достоинства виртуальной памяти

- Однородность области адресов

  каждый процесс может выполняться в памяти начиная с фиксированной (обычно нулевой) ячейки, имеющей необходимые размеры области ЗУ. Каждое обращение к виртуальной памяти во время выполнения посредством АПА преобразуется в реальное обращение.
- Защита памяти

  при каждой ссылке процессом на память проверяется, принадлежит ли она к области виртуальных адресов, отведенных для данного процесса.
- Изменение структуры памяти

  Применение виртуальной адресации позволяет преобразовать память на разных ступенях иерархии в "одноуровневую память" с одинаковым доступом ко всем элементам.

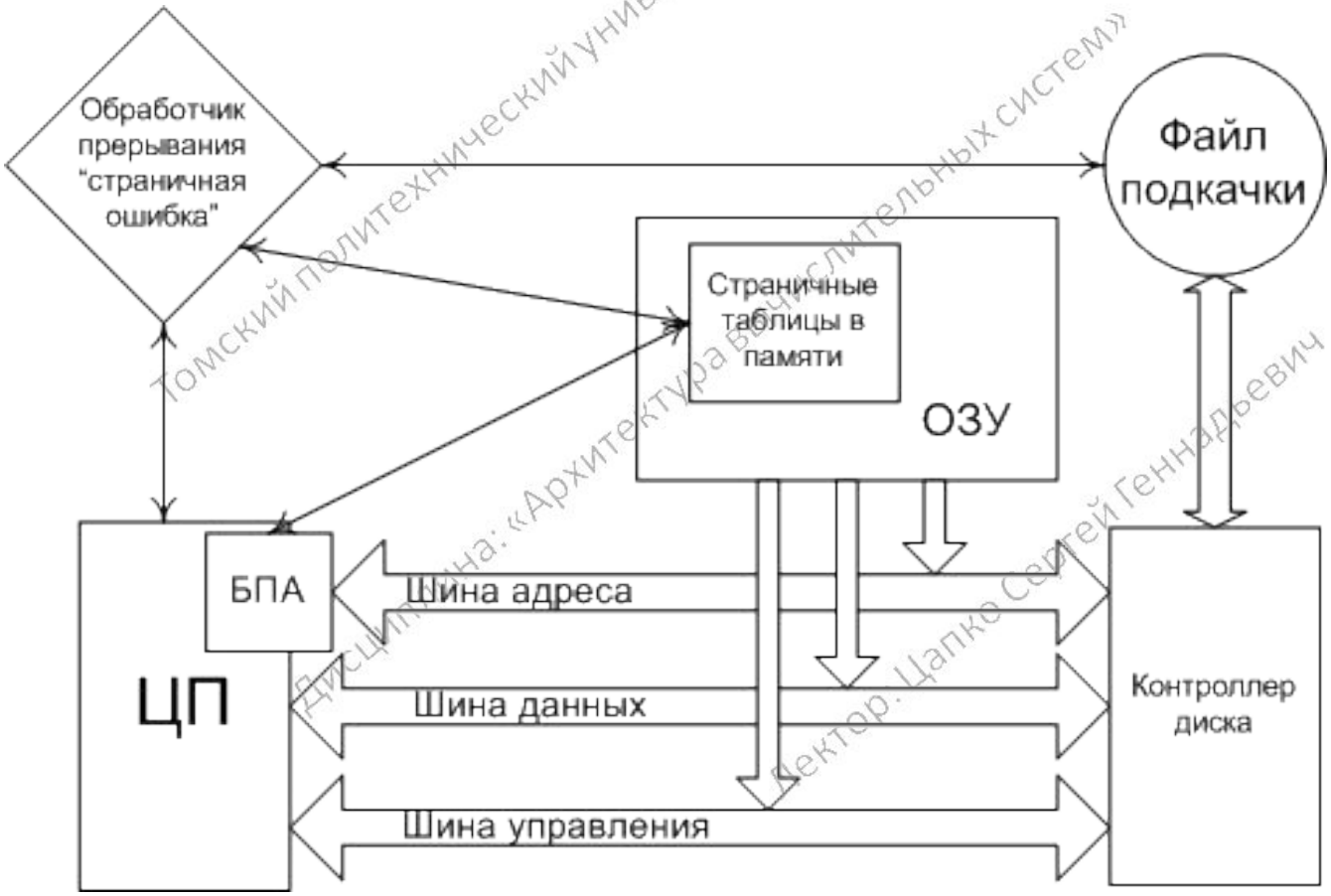
Виртуальная память

Виртуальную память пользователя можно разделить на три типа:

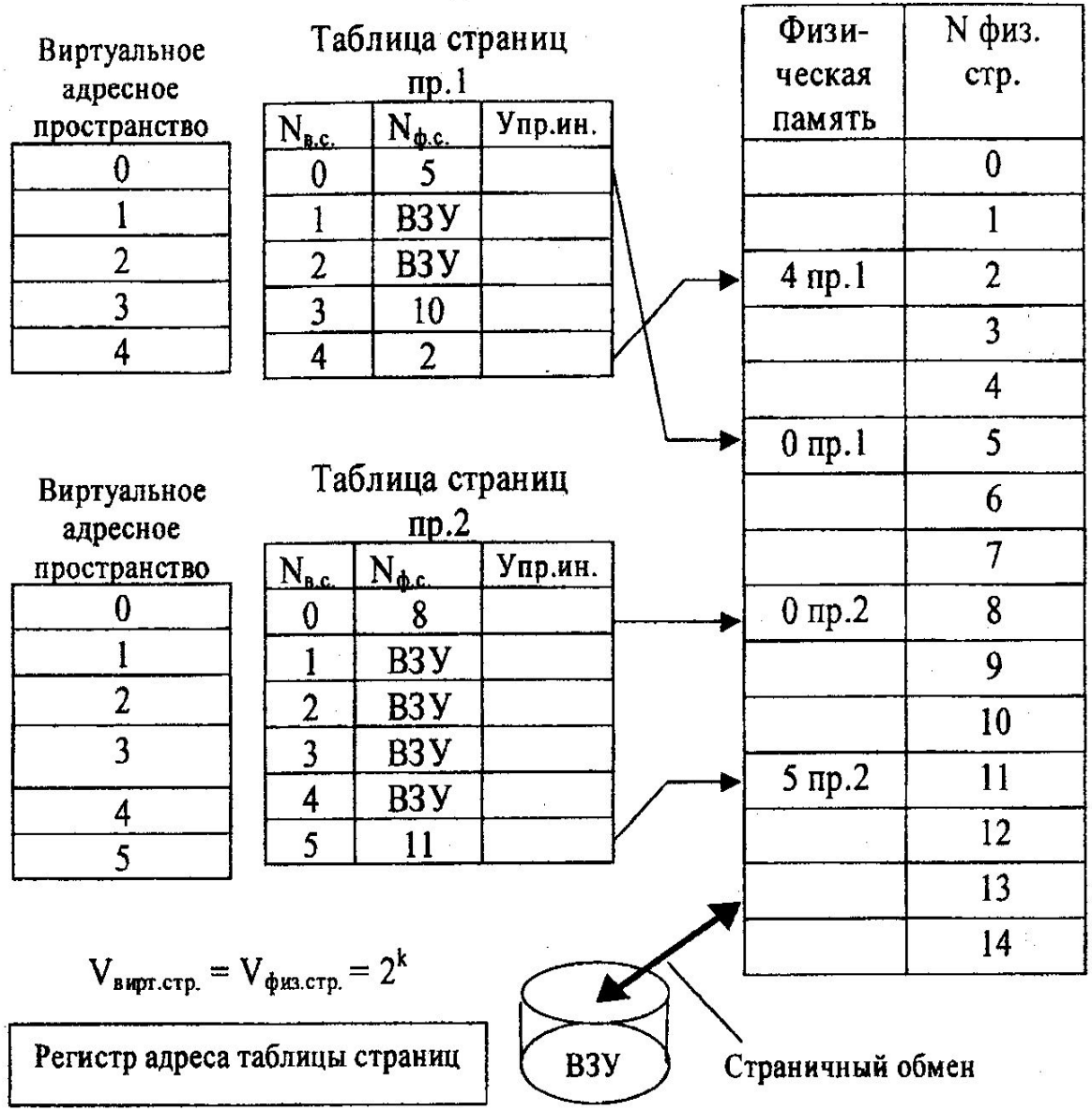
- "активные" блоки, которые содержат программу и данные, используемые в текущий момент;
- "пассивные" блоки, содержащие программу и данные, которые будут использоваться при выполнении программы;
- "мнимые" блоки, к которым не обращаются на протяжении выполнения программы.

Структура регистра адреса страницы

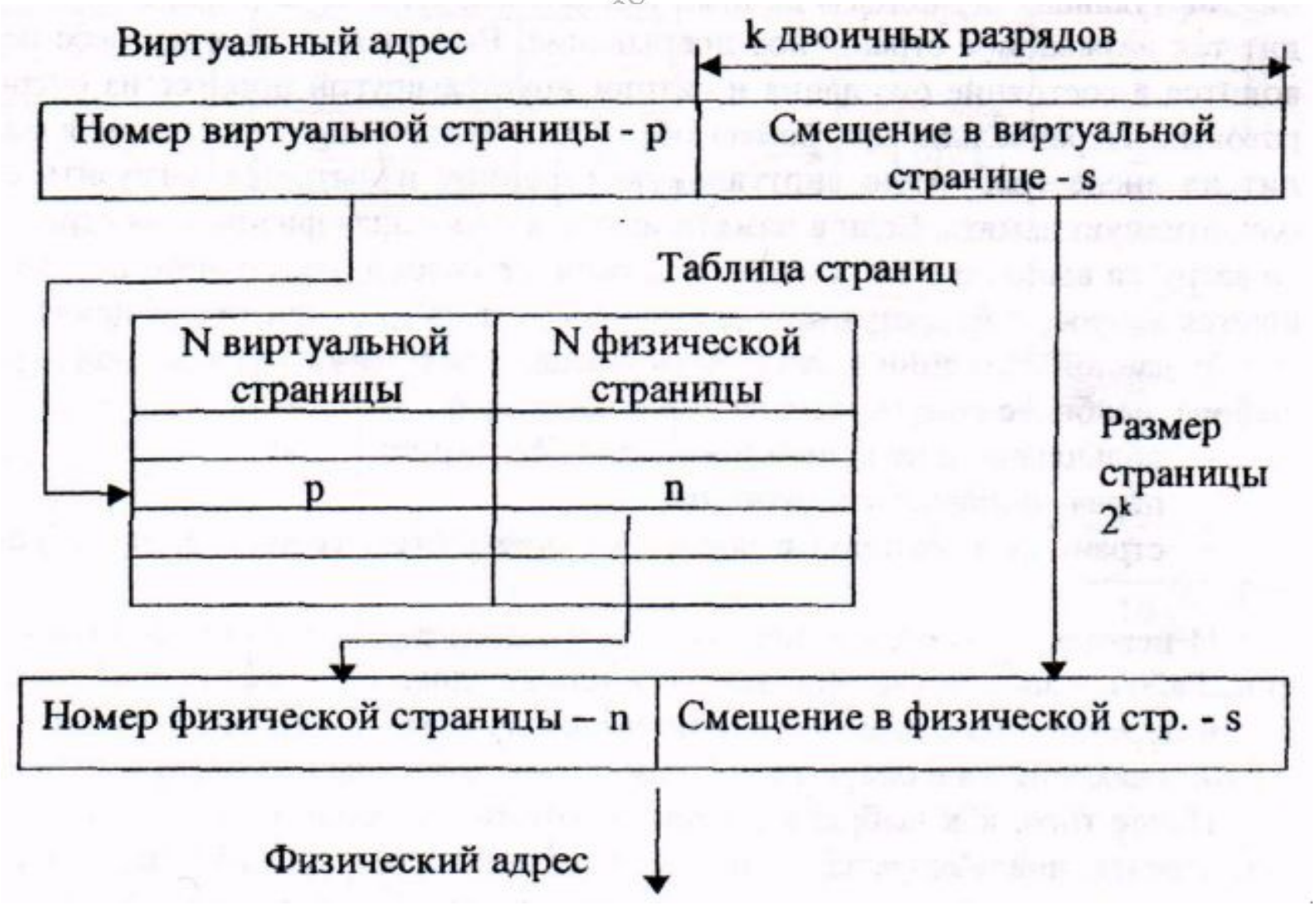
Функционирование виртуальной памяти



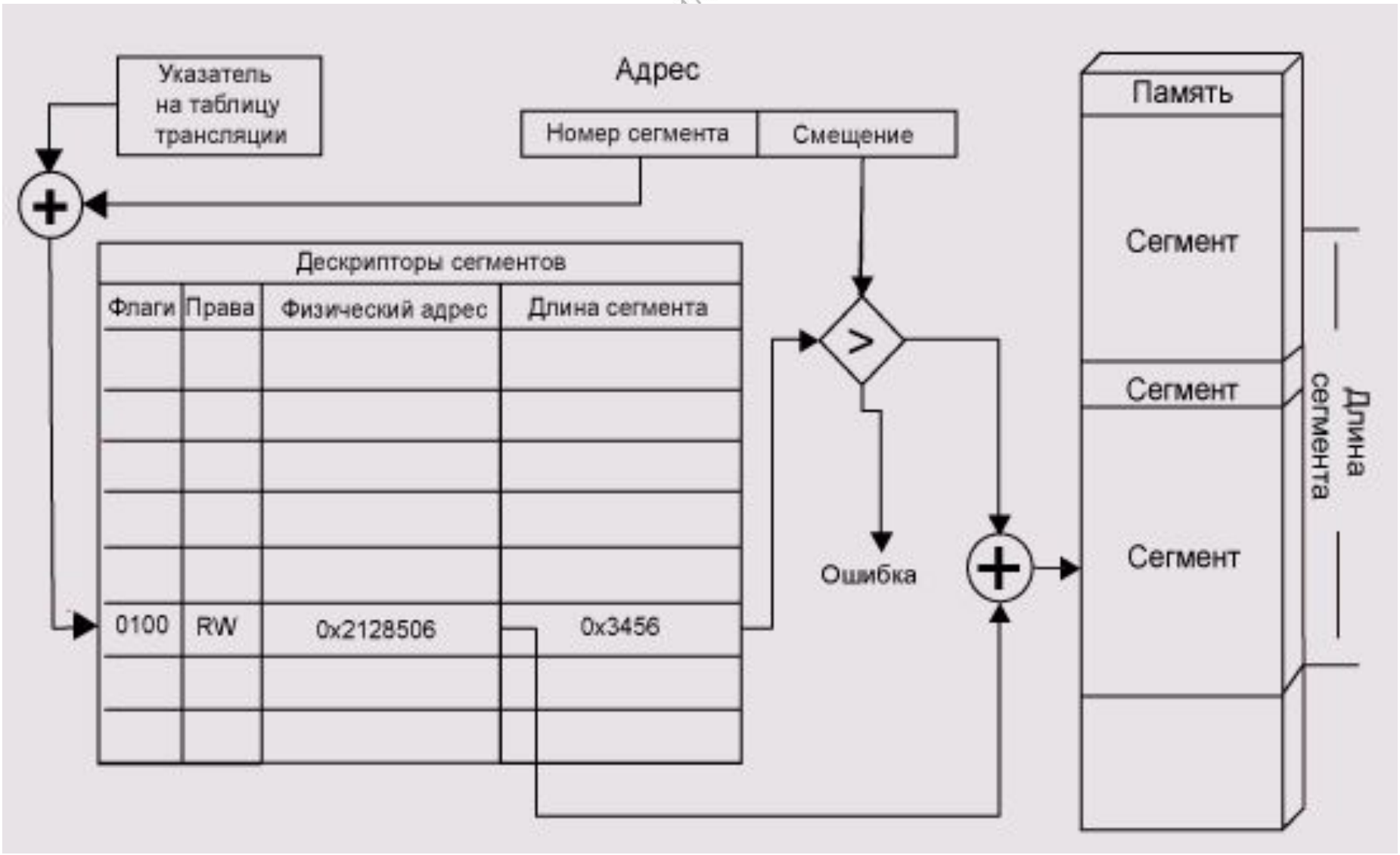
Страничное распределение памяти



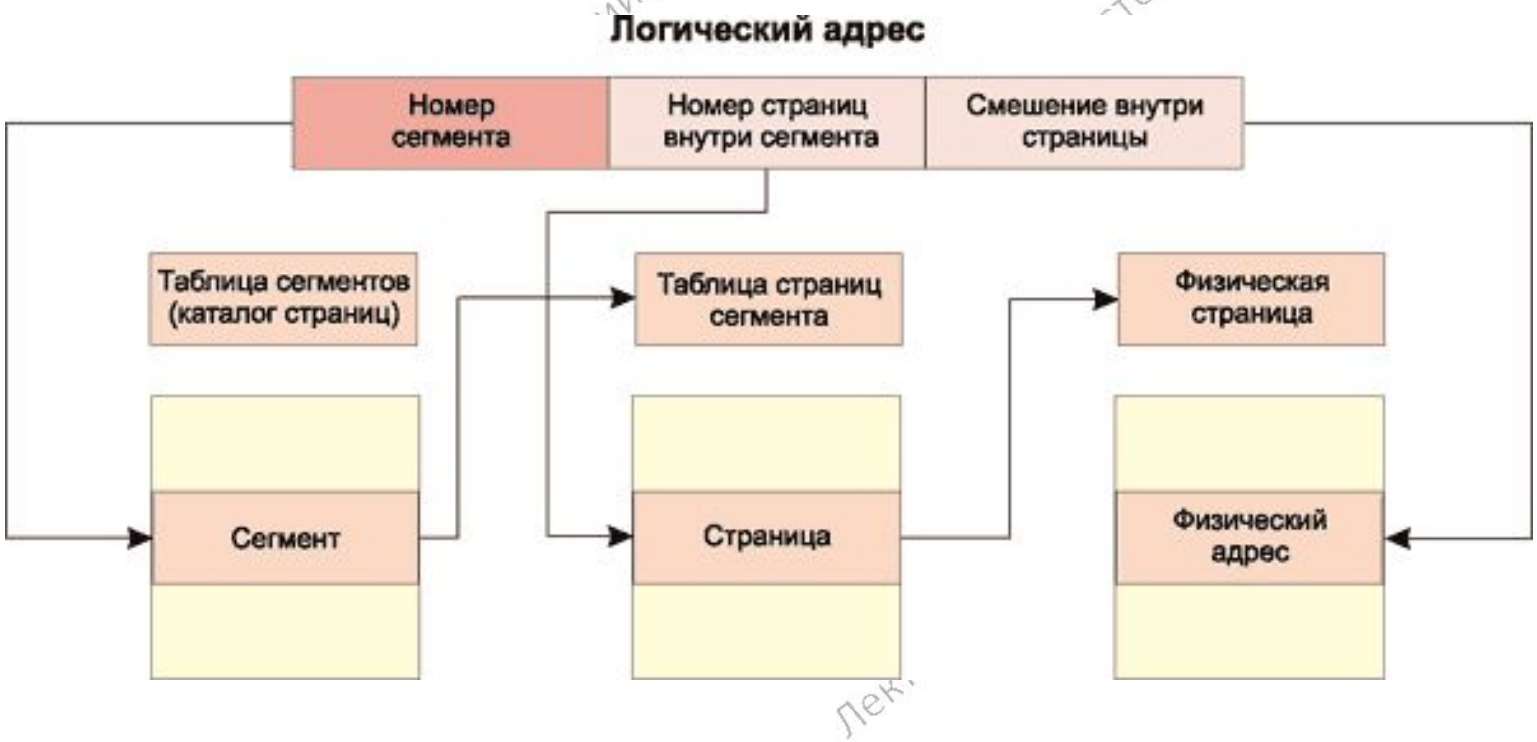
Механизм преобразования виртуального адреса в физический



Сегментное распределение



Формирование реального адреса



Управляющая ЭВМ

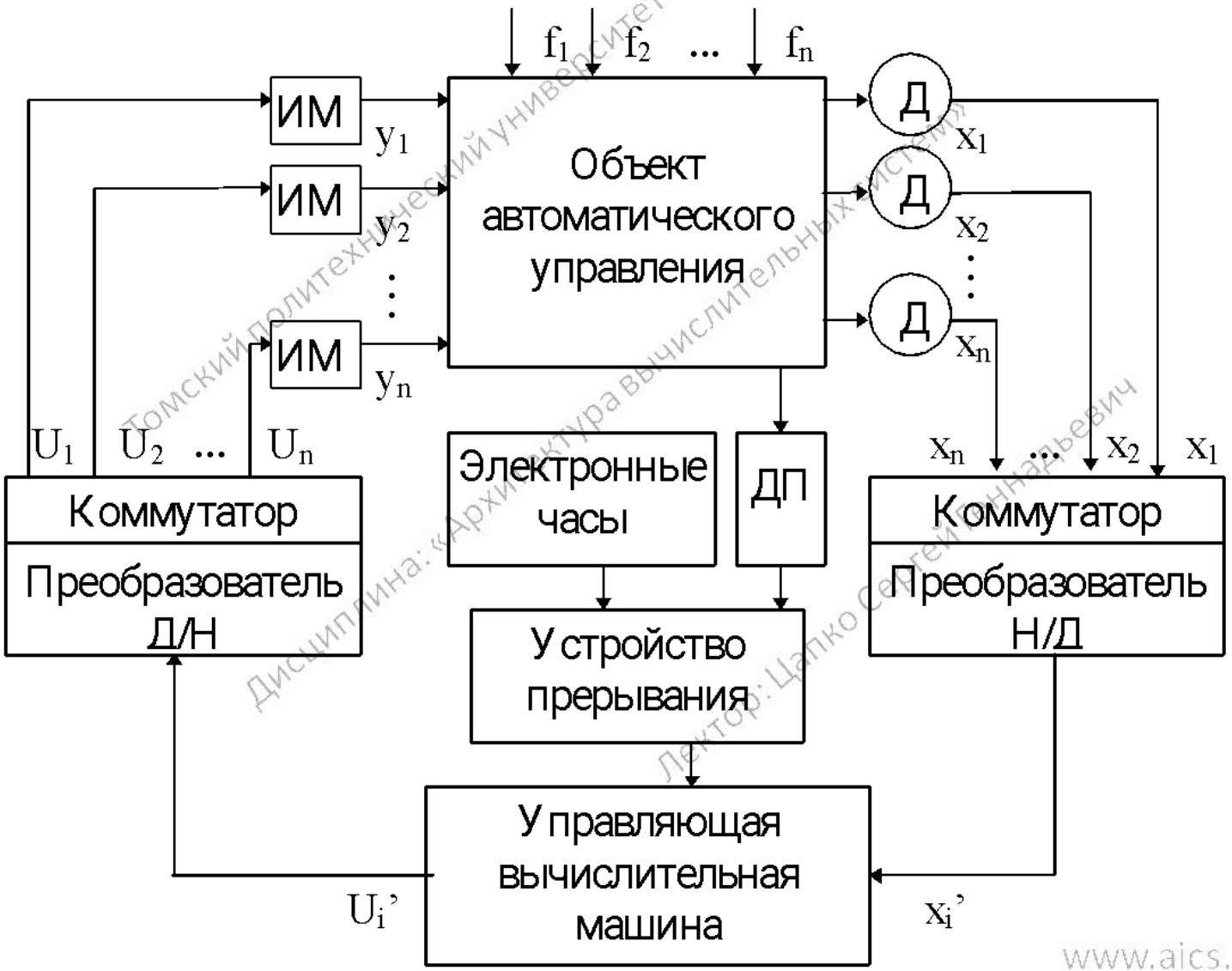
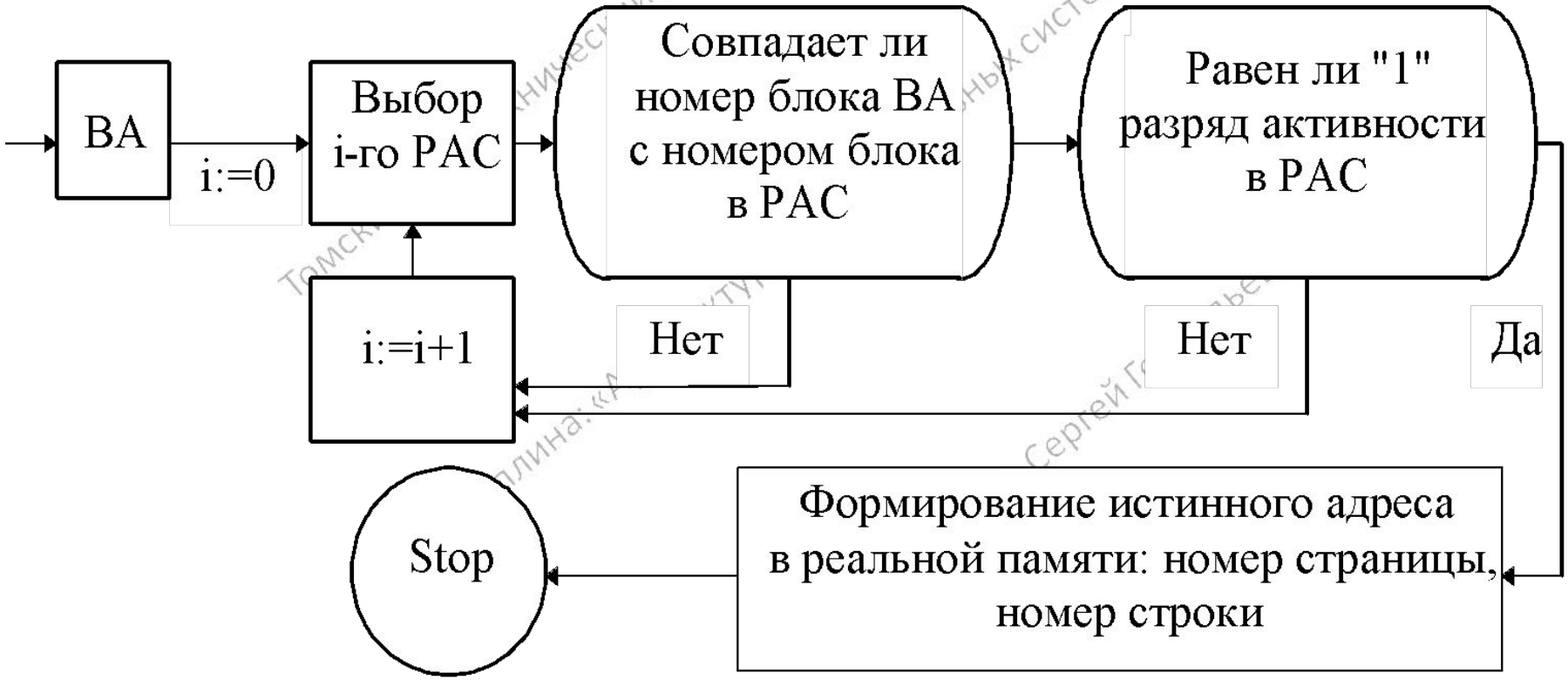


Схема отображения ВА в реальный адрес

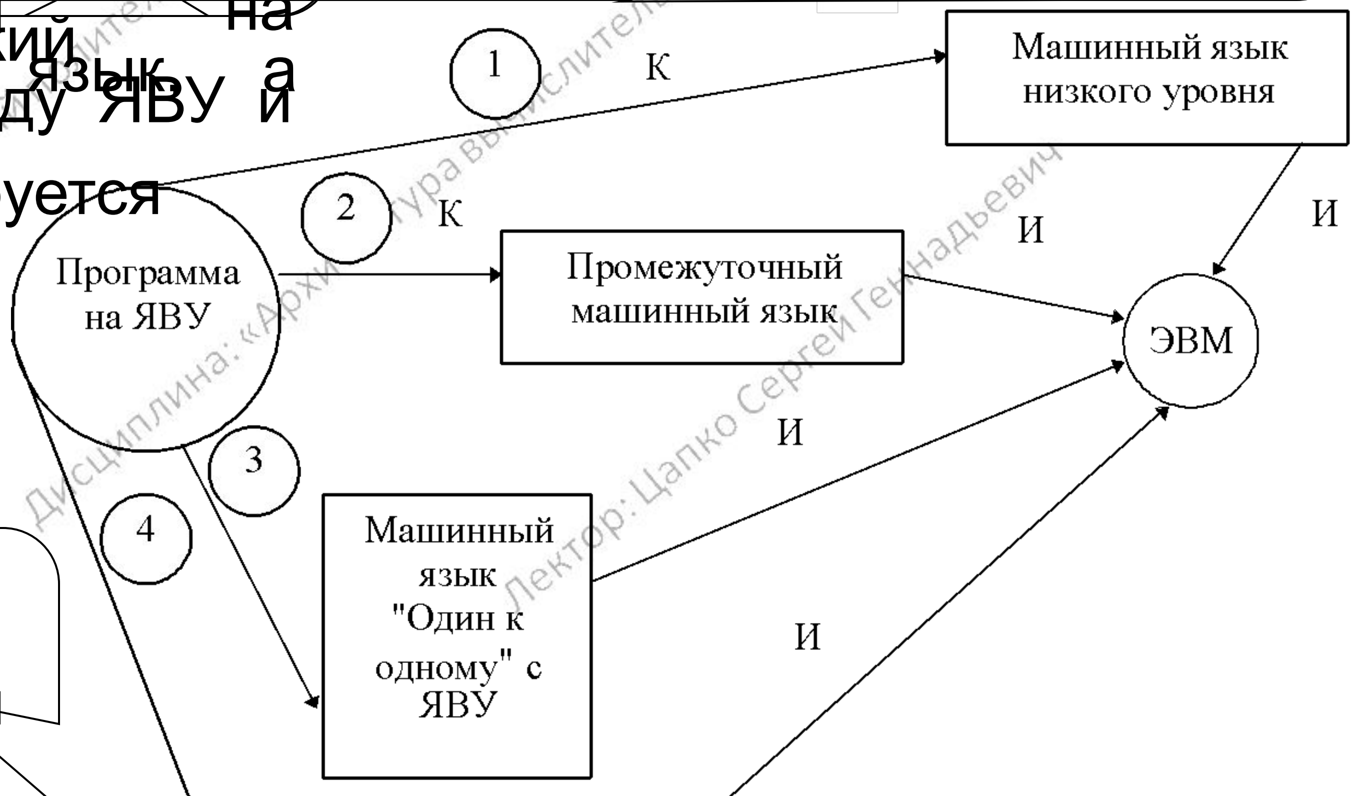


Соотношение программ на ЯВУ и машинном языке

Здесь ЯВУ можно рассматривать как язык ассемблера, т.е. имеется взаимно однозначное соответствие между типами операторов и знаков операций ЯВУ с командами машинного языка. Здесь идет ассемблирование, а не компилирование, во время которого удаляются комментарии и пробелы в исходной программе, преобразуются разделители, ключевые слова и знаки операций в машинные коды, имена – в адреса памяти. Таким образом, многих привычных функций компилятора здесь нет. Остальная привязка программы к ЭВМ происходит перед выполнением программы;

Компиляция идет на машинный язык более высокого уровня, сокращая тем самым семантический разрыв между ЯВУ и машиной;

Это традиционный подход. После компилирования программа переводится на машинный язык, а затем интерпретируется машиной;



Здесь машинный язык является ЯВУ и идет процесс интерпретаци

Основные принципы RISC-архитектуры

- каждая команда независимо от ее типа выполняется за один машинный цикл, длительность которого должна быть максимально короткой;
- все команды должны иметь одинаковую длину и использовать минимум адресных форматов, что резко упрощает логику центрального управления процессором;
- обращение к памяти происходит только при выполнении операций записи и чтения, вся обработка данных осуществляется исключительно в регистровой структуре процессора;
- система команд должна обеспечивать поддержку языка высокого уровня. (Имеется в виду подбор системы команд, наиболее эффективной для различных языков программирования.)

Отличительные особенности CISC- и RISC-архитектур

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

Достоинства RISC-архитектуры:

1. Компактность процессора, как следствие отсутствие проблем с охлаждением;
2. Высокая скорость арифметических вычислений;
3. Наличие механизма динамического прогнозирования ветвлений;
4. Большое количество оперативных регистров;
5. Многоуровневая встроенная кэш-память;

Недостаток – проблема в обновлении регистров

Экспериментальное измерение количественной оценки операций

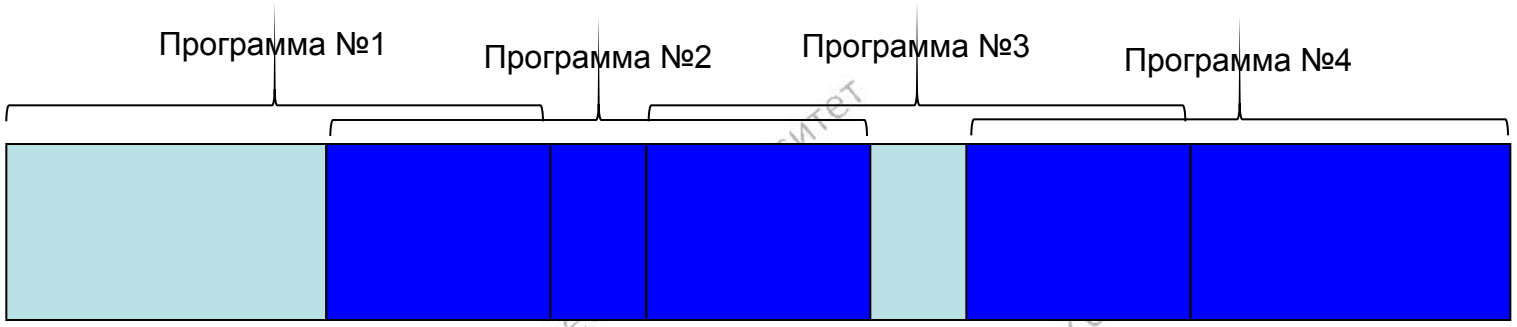
Результаты измерений в статике, проведенные для программ-компиляторов: операторы присваивания – 48 % условные операторы – 15; циклы – 16; операторы вызова-возврата – 18; прочие операторы – 3 %

Измерение процедур показали следующие измерения типов операндов: константы – 33 %; скаляры – 42; массивы (структуры) – 20 и прочие – 5 %

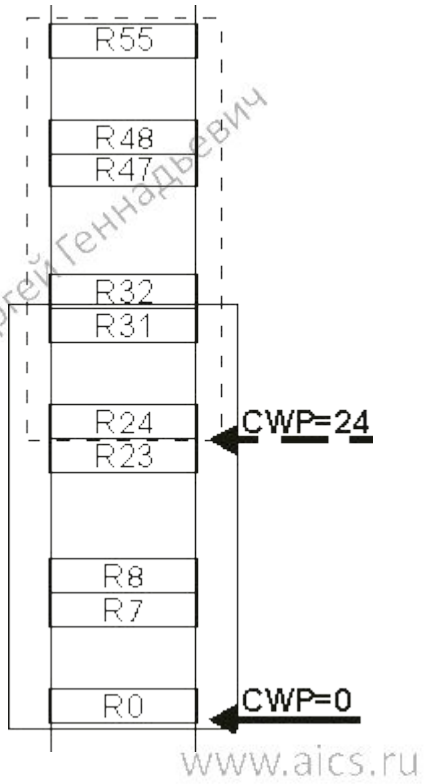
Статистика среди команд управления потоком данных следующая: команды условного перехода занимают от 66 до 78 %, команды безусловного перехода – от 12 до 18 %, частота переходов на выполнение составляет от 10 до 16 %.

Вывод:

операторы присваивания занимают основную часть в программах-компиляторах; операнды типа константа и локальные скаляры составляют основную часть

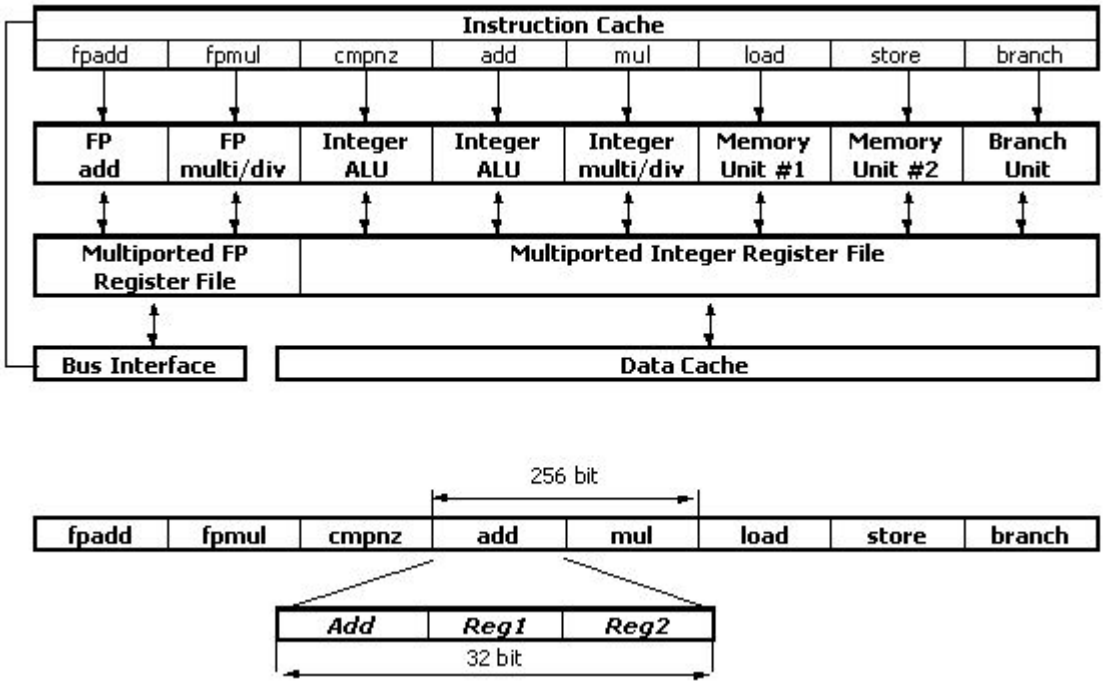


Регистровые окна



VLIW

VLIW-архитектура



Эта гипотетическая инструкция имеет восемь операционных полей, каждое из которых выполняет традиционную трехоперандную RISC-подобную инструкцию типа <регистр приемника> = <регистр источника 1> - <операция> - <регистр источника 2> (типа классической команды MOV AX BX) и может непосредственно управлять специфическим функциональным блоком при минимальном декодировании.

Процессор VLIW, имеющий схему, представленную выше, 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 самый главный момент, ограничивающий применимость данной архитектуры.

Методы адресации

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, (R1)	R4 = R4+M(R1)	Для обращения по указателю к вычисленному адресу
Индексная	Add R3, (R1+R2)	R3 = R3+M(R1+R2)	Полезна при работе с массивами: R1 – база, R3 – индекс

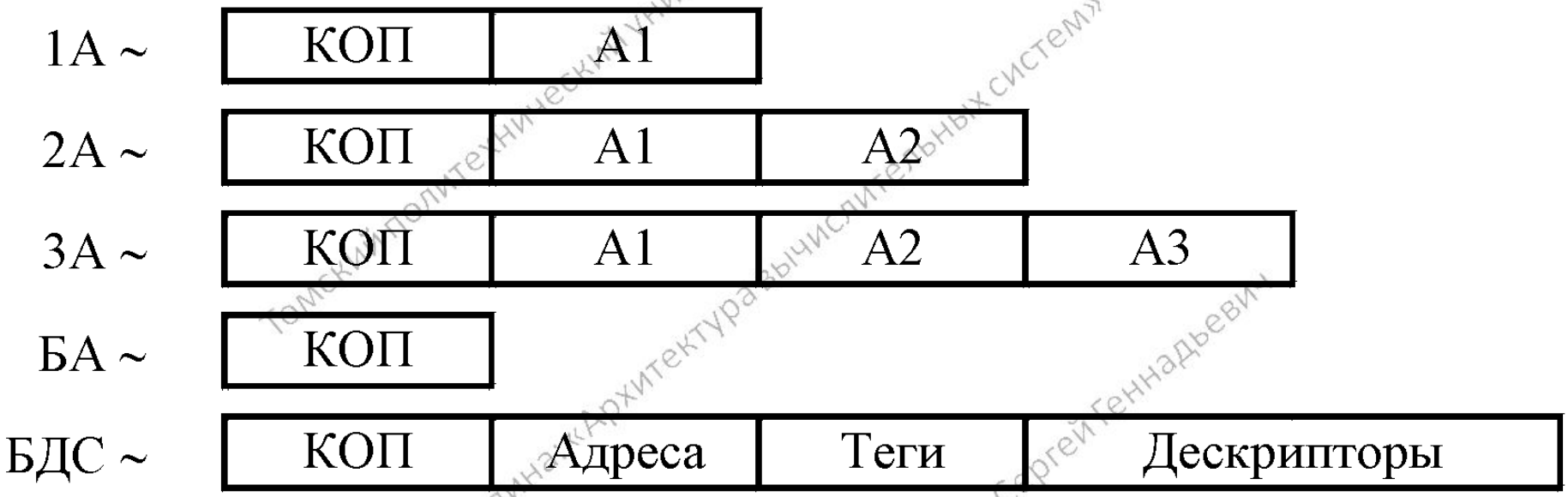
Методы адресации

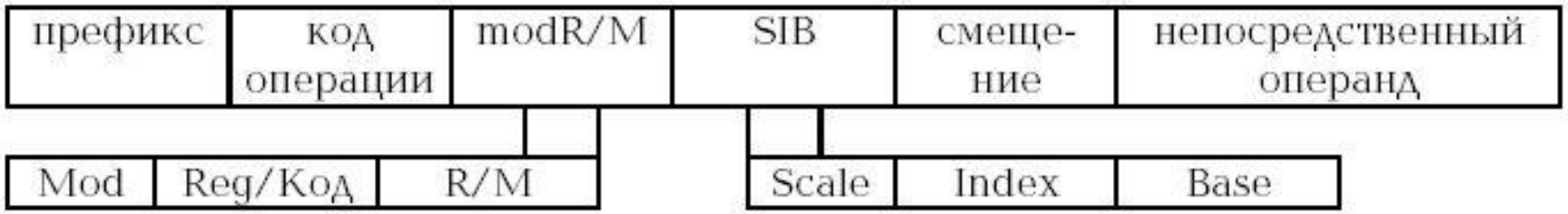
Метод адресации	Пример команды	Смысл команды	Использование команды
Прямая или абсолютная	Add R1, (1000)	R1=R1+M(1000)	Полезна для обращения к статическим данным
Косвенная	Add R1, @(R3)	R1 = R1+M(M(R3))	Если R3 – адрес указателя p, то выбирается значение по этому указателю
Автоинкрементная	Add R1, (R2)+	R1 = 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 R1, (R2)–	R2 = R2–d R1 = R1+M(R2)	Аналогична предыдущей. Обе могут использоваться для реализации стека
Базовая индексная со смещением и масштабированием	Add R1, 100(R2)(R3)	R1=R1+M(100)+R2 +R3*d	Для индексации массивов

Основные типы команд

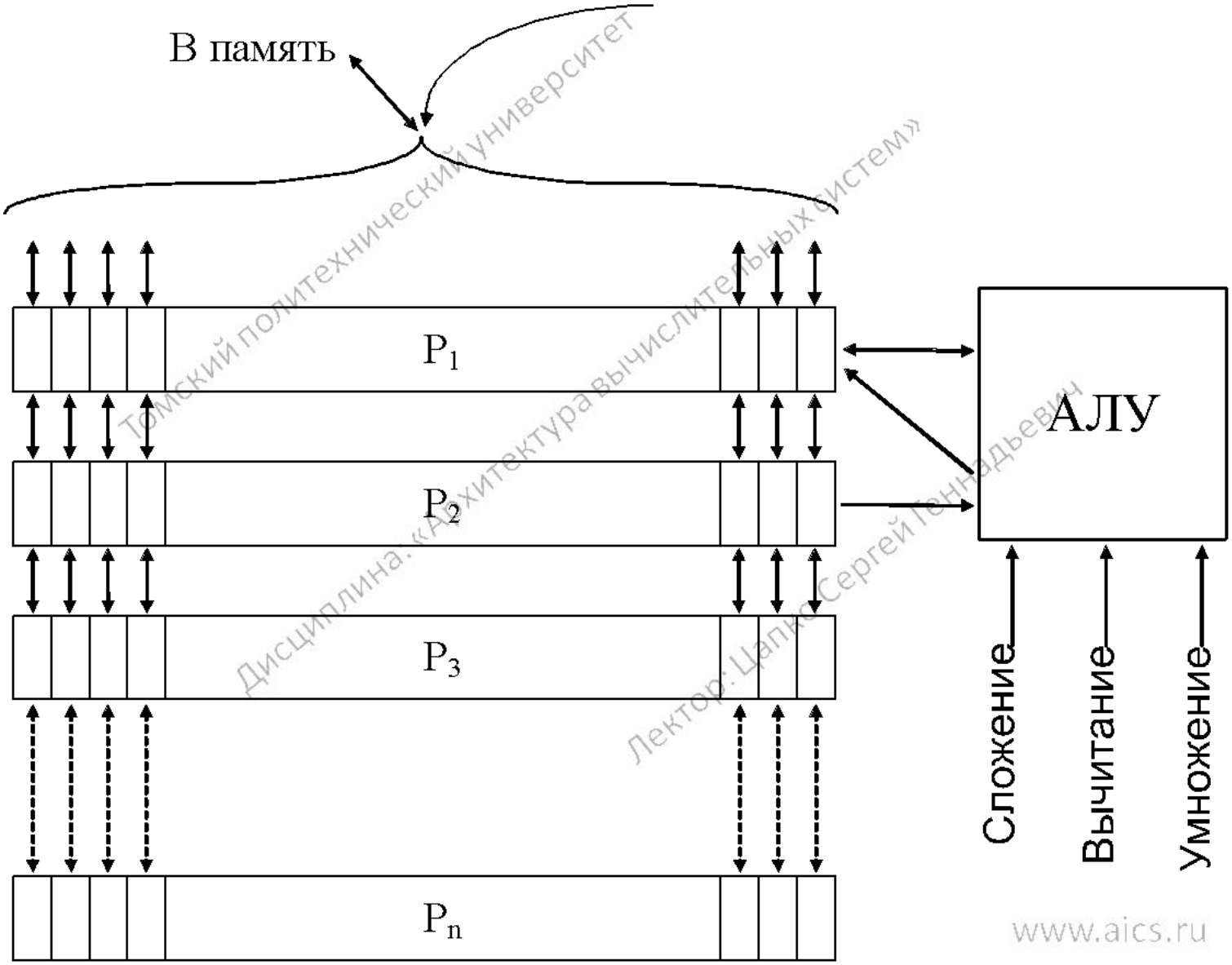
Тип операции	Примеры
Арифметические и логические	Целочисленные арифметические и логические операции: сложение, вычитание, логическое сложение, логическое умножение и т. д.
Пересылки данных	Операции загрузки/записи
Управление потоком команд	Безусловные и условные переходы, вызовы процедур и возвраты
Системные 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 строками	Пересылки, сравнения и поиск строк

Структура команд





Стековая организация регистровой памяти процессора



Основные операция и спецкоманды

Операции с регистрами:

- Движение вниз: (P1) → P2, (P2) → P3, ..., а P1 заполняется данными из главной памяти
- Движение вверх: (Pn) → Pn-1, (Pn-1) → Pn-2, а Pn заполняется нулями
- Регистры P1 и P2 связаны с АЛУ, образуют два операнда для выполнения операции. Результат операции записывается в P1, т.е. (P1) ⊗ (P2)→(P1)
- При выполнении любой операции над двумя регистрами осуществляется продвижение операндов вверх, не затрагивая P1, т. е. (P3) → P2, (P4) → P3 и т. д

Спецкоманды:

- дублирование ~ (P1) → P2, (P2) → P3, ... и т. д., а (P1) остается при этом неизменным;
- реверсирование ~ (P1) → P2, а (P2) → P1, что удобно для выполнения некоторых операций.

Программа решения математической задачи для одноадресного компьютера

$$X = \frac{a^2 + b^2}{b + c}$$

Номер команды	Команда	Комментарии
1	$C \rightarrow \sum$	
2	$(\sum) + b$	
3	$(\sum) \rightarrow P_1$	P_1 – рабочая ячейка
4	$a \rightarrow \sum$	
5	$(\sum) \bullet a$	
6	$(\sum) \rightarrow P_2$	P_2 – рабочая ячейка
7	$b \rightarrow \sum$	
8	$(\sum) \bullet b$	
9	$(\sum) + (P_2)$	
10	$(\sum) : (P_1)$	$\frac{a^2 + b^2}{b + c} \rightarrow \sum$

Программа решения математической задачи на ЭВМ со стековой организацией памяти

№ п/п	Команда	P_1	P_2	P_3	P_4
1	2	2	3	4	5
1	Вызов **b**	b			
2	Дублирование	b	B		
3	Вызов **c**	c	B	B	
4	Сложение	b+c	B		
5	Реверсирование	b	b+c		
6	Дублирование	b	B	b+c	
7	Умножение	b^2	b+c		
8	Вызов **a**	a	b^2	b+c	
9	Дублирование	a	A	b^2	b+c
10	Умножение	a^2	b^2	b+c	
11	Сложение	a^2+b^2	b+c		
12	Деление	$\frac{a^2+b^2}{b+c}$			

$$X = \frac{a^2 + b^2}{b + c}$$

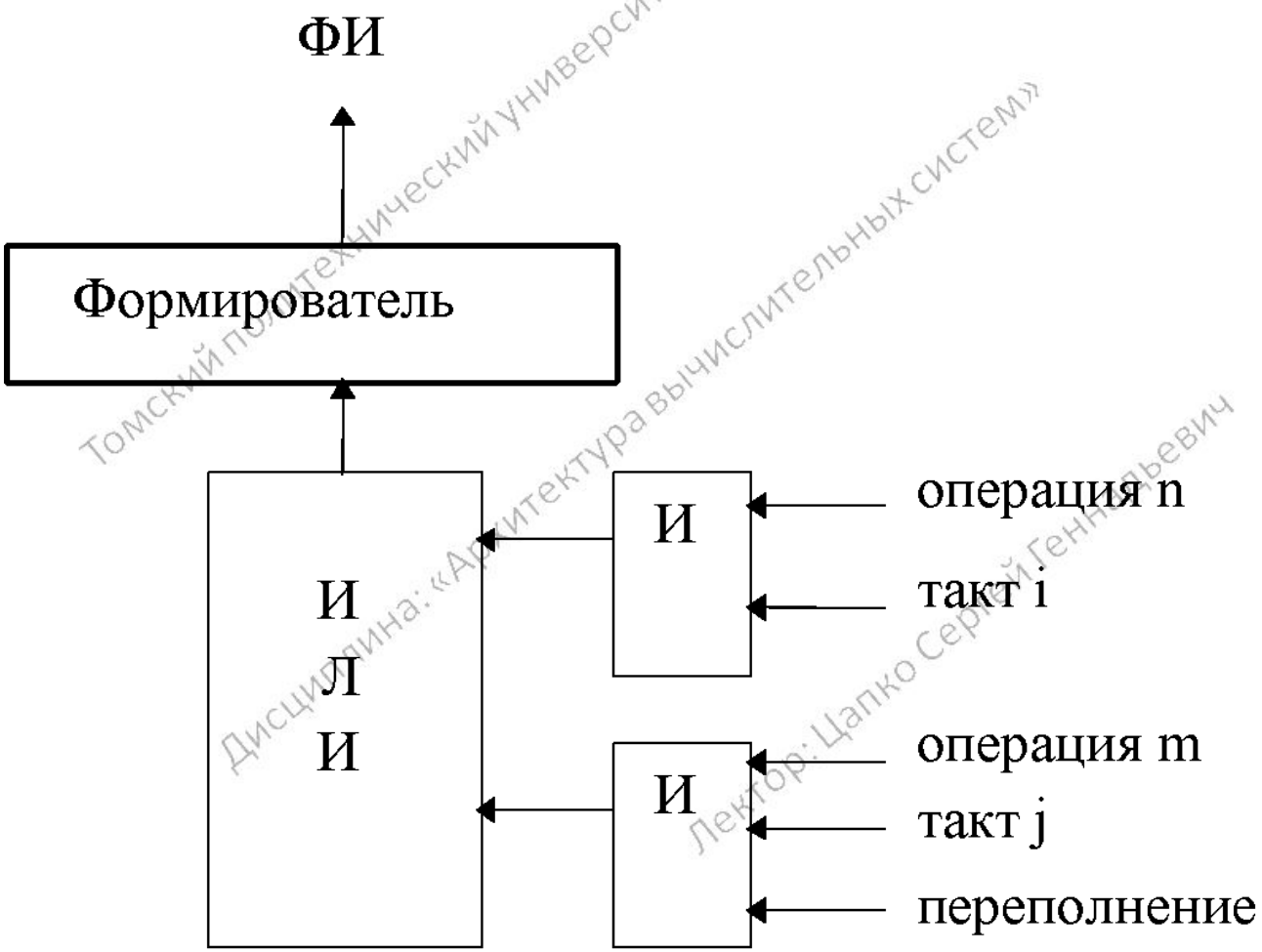
Способы проектирования системы команд

1. Сокращение набора команд, присущих СК выбранного микропроцессора. Все частоты встреч операций для задания их в СК всякий раз можно определить из соотношений "стоимость затрат – сложность реализации – получаемый выигрыш".
2. Второй путь проектирования СК состоит в расширении имеющейся системы команд. Один из способов такого расширения □–□создание макрокоманд, второй – используя имеющийся синтаксис языка СК, дополнить его новыми командами с последующим переассемблированием, через расширение функций ассемблера. Оба эти способа принципиально одинаковы, но отличаются в тактике реализации аппарата расширения.

Способы оптимизации системы команд

1. Выявление частоты повторений сочетаний двух или более команд, следующих друг за другом в некоторых типовых задачах для данного компьютера, с последующей заменой их одной командой, выполняющей те же функции.
2. Исследование часто генерируемых компилятором последовательностей команд с последующим редактированием и ликвидацией из них избыточных кодов.
3. Оптимизацию можно проводить и в пределах отдельной команды, исследуя ее информационную емкость. Для этого можно применить аппарат теории информации, в частности для оценки количества переданной информации – энтропию источника .

Принцип управления операциями на основе «жесткой» логики



Горизонтальное микропрограммирование

	ФИ1	ФИ2	ФИ3	ФИ4	…	$ФИ_M$
МК1	1	1	0	1		1
МК1	0	1	1	0		0
МК3	1	1	0	1		0
…						
$МК_N$	0	1	0	0		1

Вертикальное микропрограммирование

$ФИ_1$	$ФИ_2$	$ФИ_3$	$ФИ_4$		$ФИ_{16}$
0000	0001	0010	0011		1111

МК1	0010	1111				
МК2	0000	1010	1100			
МК3	1100					
МК4	0011	1011	1100	1101		
…						
$МК_N$	1100	1111				

Оценка современных компьютеров

- Узкие места современных ЭВМ
- Оценка производительности ВС
- Методы повышения производительности ЭВМ
- Компьютеры с общей памятью
- Компьютеры с распределенной памятью
- Языки параллельного программирования
- Характеристики основных классов современных параллельных компьютеров
- Классификация ВС

Основные причины возникновения узких мест в компьютере

- состав, принцип работы и временные характеристики арифметико-логического устройства;
- состав, размер и временные характеристики устройств памяти;
- структура и пропускная способность коммуникационной среды;
- компилятор, создающий неэффективные коды;
- операционная система, организующая неэффективную работу с памятью, особенно медленной.

Методы оценки производительности ВС

- Пиковая производительность (суммарное количество операций, выполняемых в единицу времени всеми имеющимися в компьютере обрабатывающими логико-арифметическими устройствами)
- Реальная производительность (определяется при выполнении реальных прикладных программ)

Группы тестов для измерения реальной производительности

1. Тесты производителей (тесты, подготовленные компаниями разработчика ВС).
2. Стандартные тесты (тестовые программы, основанные на выполнении стандартных операций и не зависящие от платформы ВС
3. Пользовательские тесты (учитывают конкретную специфику применения ВС)

Основные проблемы, связанные с анализом результатов контрольного тестирования производительности

- отделение показателей, которым можно доверять безоговорочно, от тех, которые должны восприниматься с известной долей настороженности (***проблема достоверности оценок***);
- выбор контрольно-оценочных тестов, наиболее точно характеризующих производительность при обработке типовых задач пользователя (***проблема адекватности оценок***);
- правильное истолкование результатов тестирования производительности, особенно если они выражены в довольно экзотических единицах типа MWIPS, Drystones/s и т.д. (***проблема интерпретации***).

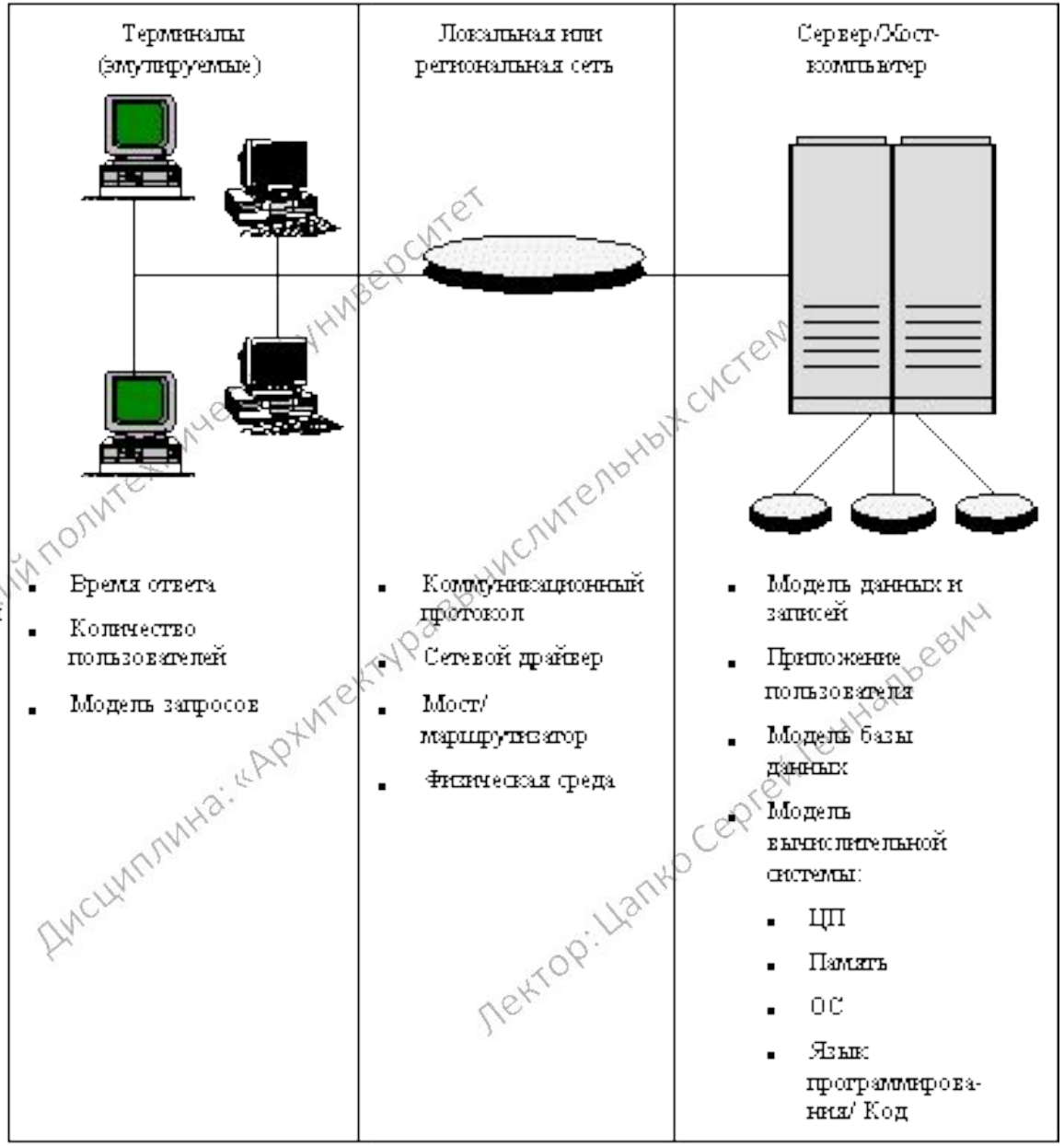
Sun Ultra5-10

Стандартные тесты

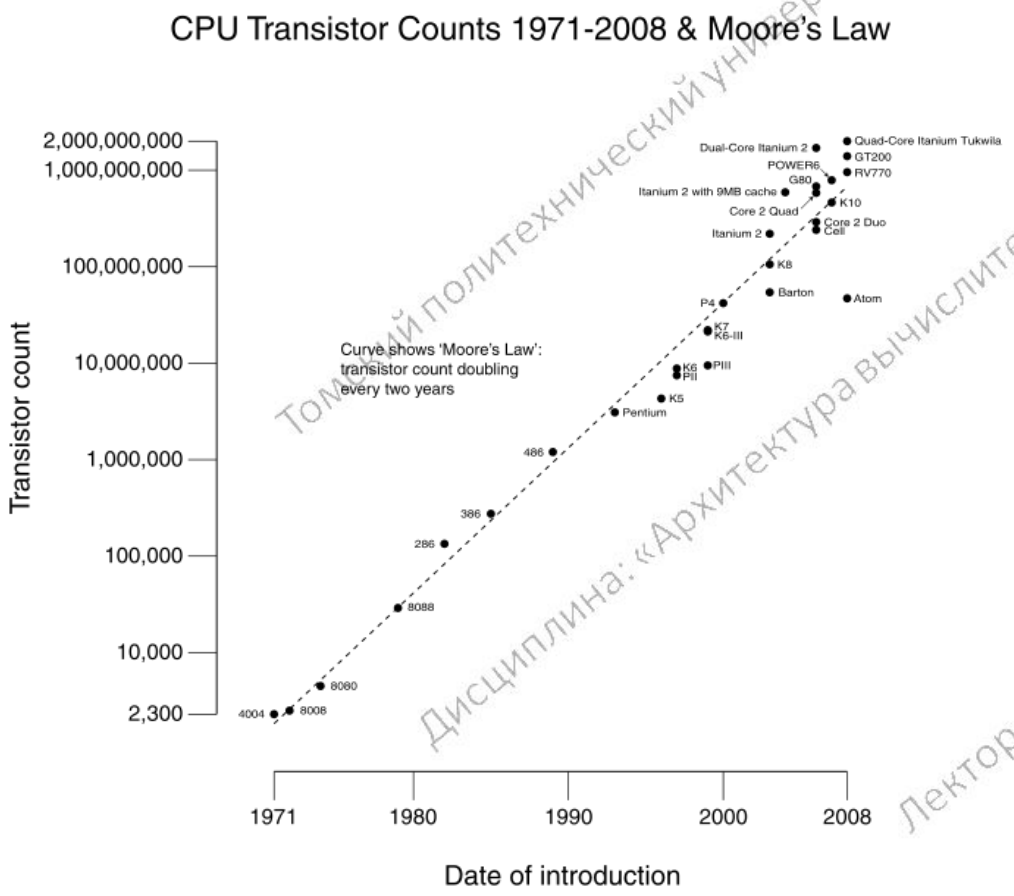
SPECratio

SPECmark

- ***LinPack*** - совокупность программ для решения задач линейной алгебры
  В качестве параметров используются: порядок матрицы, формат значений элементов матрицы, способ компиляции.
- ***SPEC XX*** - два тестовых набора Cint89 и Cfp89.
  SPEC 98 – четыре программы целочисленной обработки шесть программ с операциями на числами с плавающей запятой.
  SPEC 92 – 6 эталонных тестов, а также 14 реальных прикладных программ
  SPEC 95 – расширен набор тестовых программ, а также добавлена возможность тестирования многопроцессорных ВС.
  современные тесты SPEC – тестирование многомашинных и многопроцессорных вычислительных комплексов.
- ***TPC*** – оценка производительности систем при работе с базами данных. Тестирование позволяет определить:
  а. производительность обработки запросов QppD (Query Processing Performance), измеряемая количеством запросов, которое может быть обработано при монопольном использовании всех ресурсов тестируемой системы;
  б. пропускная способность системы QthD (Query Throughput), измеряемая количеством запросов, которое система в состоянии совместно обрабатывать в течение часа;
  в. отношение стоимости к производительности $/QphD, измеряемое как стоимость 5-летней эксплуатации системы, отнесенная к числу запросов, обработанных в час.



Закон Мура



Зако́н Му́ра — эмпирическое — эмпирическое наблюдение, сделанное в 1965 году — эмпирическое наблюдение, сделанное в 1965 году (через шесть лет после изобретения интегральной схемы — эмпирическое наблюдение, сделанное в 1965 году (через шесть лет после изобретения интегральной схемы), в процессе подготовки выступления Гордоном Муром — эмпирическое наблюдение, сделанное в 1965 году (через шесть лет после изобретения интегральной схемы), в процессе подготовки выступления Гордоном Муром (одним из основателей Intel — эмпирическое наблюдение, сделанное в 1965 году (через шесть лет после изобретения интегральной схемы), в процессе подготовки выступления Гордоном Муром (одним из основателей Intel). Он высказал предположение, что число транзисторов на кристалле будет удваиваться каждые 24 месяца.

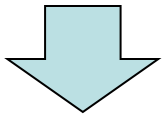
Увеличение производительности процессоров

Увеличение скорости исполнения команд

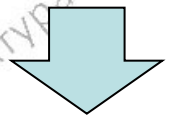
Использование новых архитектур процессоров

a. Суперскалярная архитектура.
b. Оптимизация выполнения команд.
c. Конвейерная обработка.
d. Предсказание ветвлений и переходов.

Увеличение скорости исполнения команд



Реализуется за счет уменьшения размеров электронных компонент схем и повышения тактовой частоты



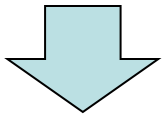
Pentium 200 МГц 0,25 мкм

Intel® Pentium® III (от 450 МГц до 1,33 ГГц) - 0,13 мкм

Intel Original LGA775 Celeron Dual Core-E1200 (1.6/800/512K) - 0,065 мкм

Intel Original LGA775 Core2Duo-E7200 (2.53/1066/3Mb) - 0,045 мкм

Использование новых архитектур процессоров



Опирается на схемотехнику и усовершенствование программных методов

- Суперскалярность
- Оптимизация последовательности выполнения команд
- Конвейерная обработка → Упреждение исполнения, предсказание ветвлений и переходов

Параллелизм на уровне команд (ILP — Instruction-Level Parallelism)

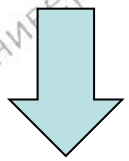
Общие свойства ILP-процессоров - способность одновременно и независимо выполнять несколько операций при наличии нескольких функциональных устройств различных типов, таких как, например, устройство обмена с памятью, арифметическое устройство и др.

Суперскалярные процессоры — это реализации ILP-процессора для последовательных архитектур, программа для которых не должна передавать и, фактически, не может передавать точную информацию о параллелизме. Поскольку программа не содержит точной информации о наличии ILP, задача обнаружения параллелизма должна решаться аппаратурой, которая, в свою очередь, должна создавать план действий для обнаружения «скрытого параллелизма».

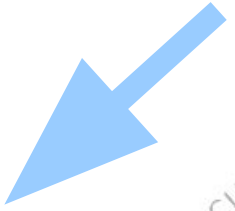
Конвейерный принцип обработки информаци подразумевает, что в каждый момент времени процессор работает над различными стадиями выполнения нескольких команд, причем на выполнение каждой стадии выделяются отдельные аппаратные ресурсы. По очередному тактовому импульсу каждая команда в конвейере продвигается на следующую стадию обработки, выполненная команда покидает конвейер, а новая поступает в него.

Идеология EPIC заключается в том, чтобы, с одной стороны, полностью возложить составление плана выполнения команд на компилятор, с другой стороны, предоставить необходимые аппаратные средства, позволяющие при статическом планировании на стадии компиляции использовать механизмы, подобные тем, которые применяются при динамическом планировании в суперскалярных архитектурах

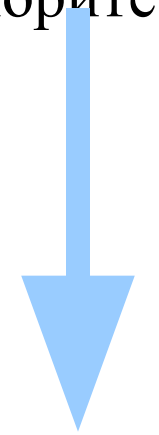
Оптимизация последовательности выполнения команд



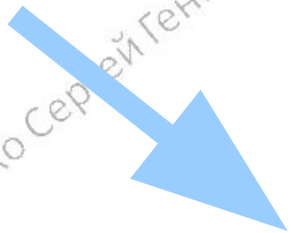
Подходы, используемые при оптимизации кода, могут существенно зависеть от критериев оптимизации. Обычно рассматривают три критерия или их комбинации с некоторыми приоритетами.



минимизация размера кода

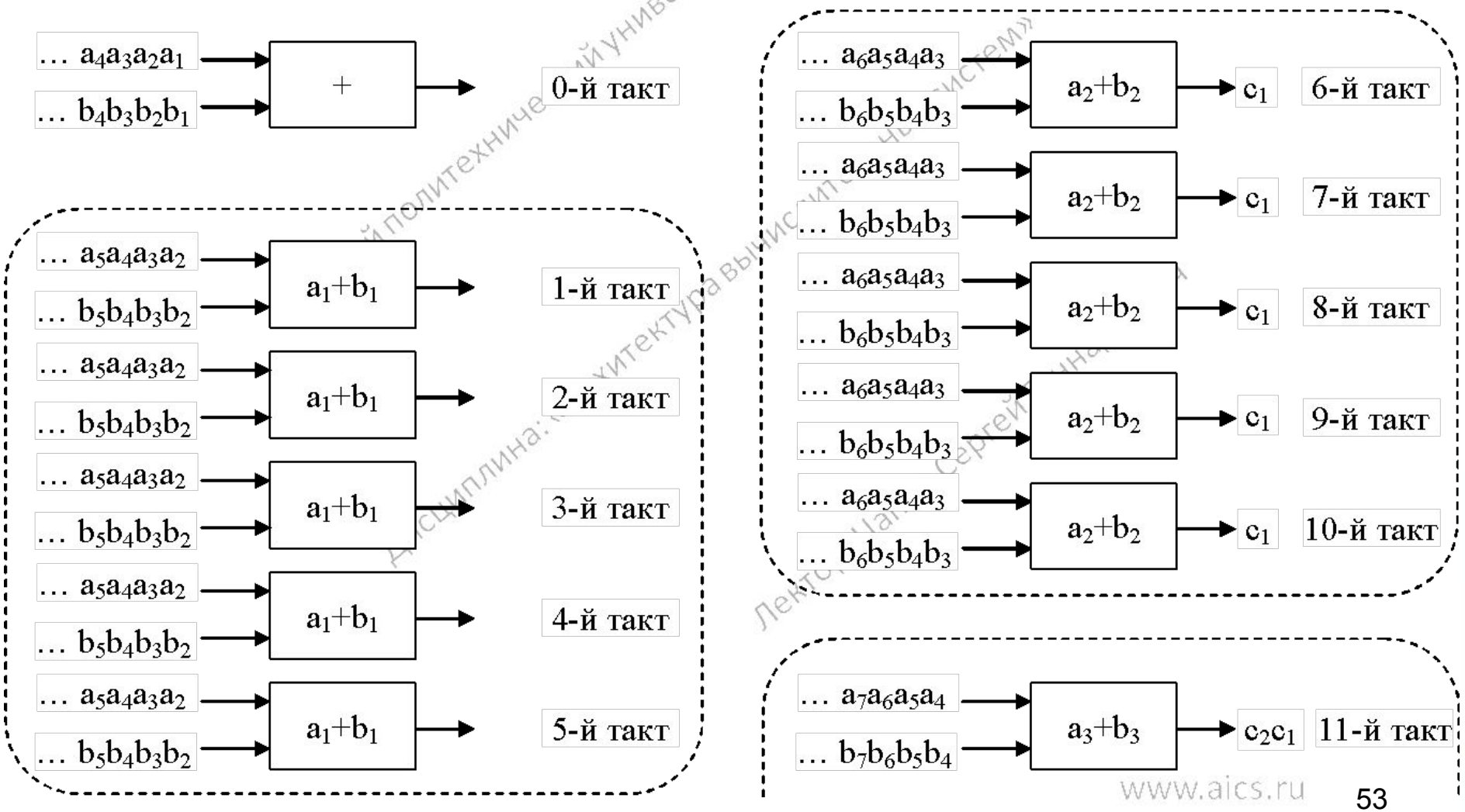


минимизация времени выполнения программы

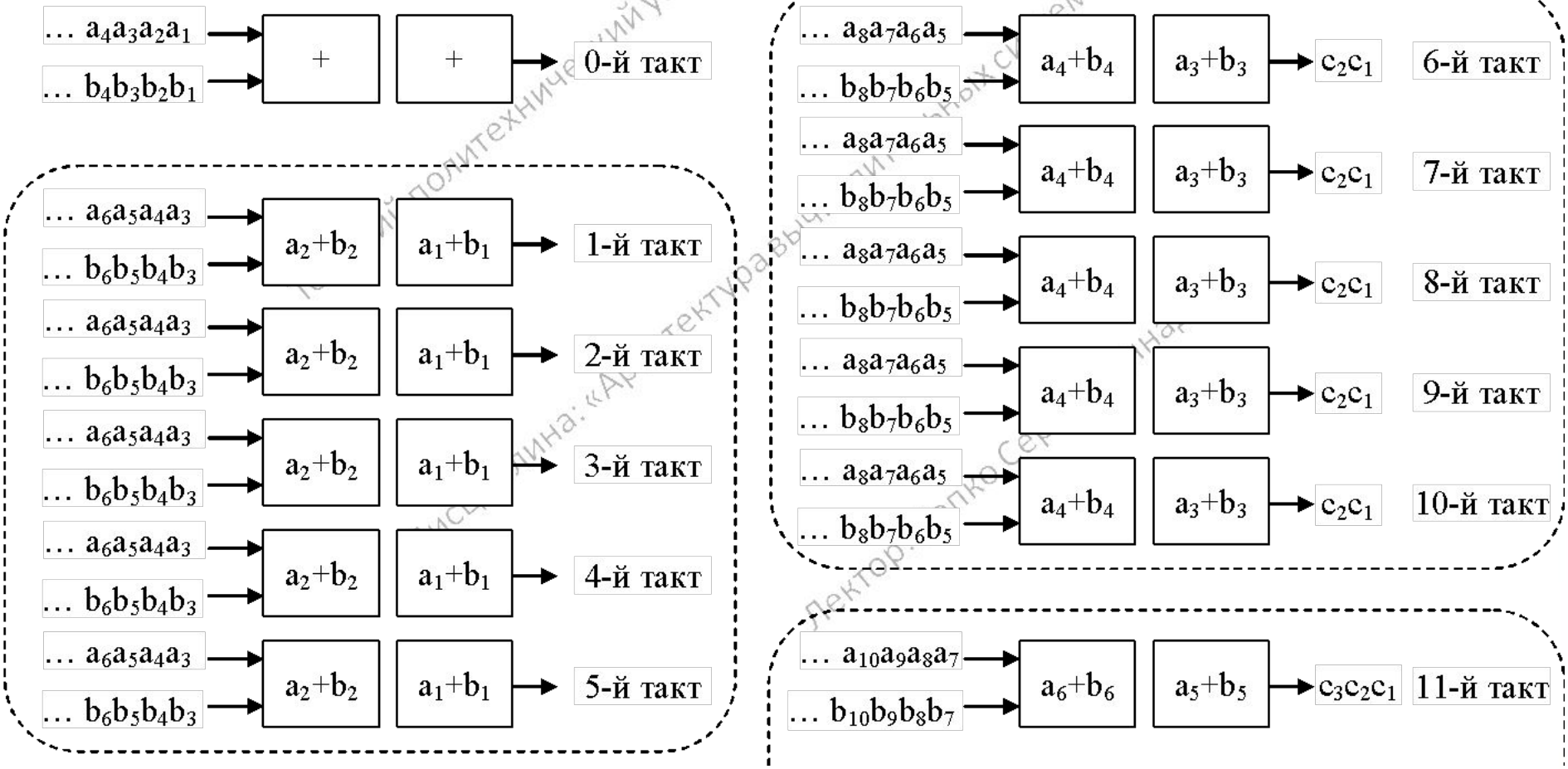


минимизация энергопотребления

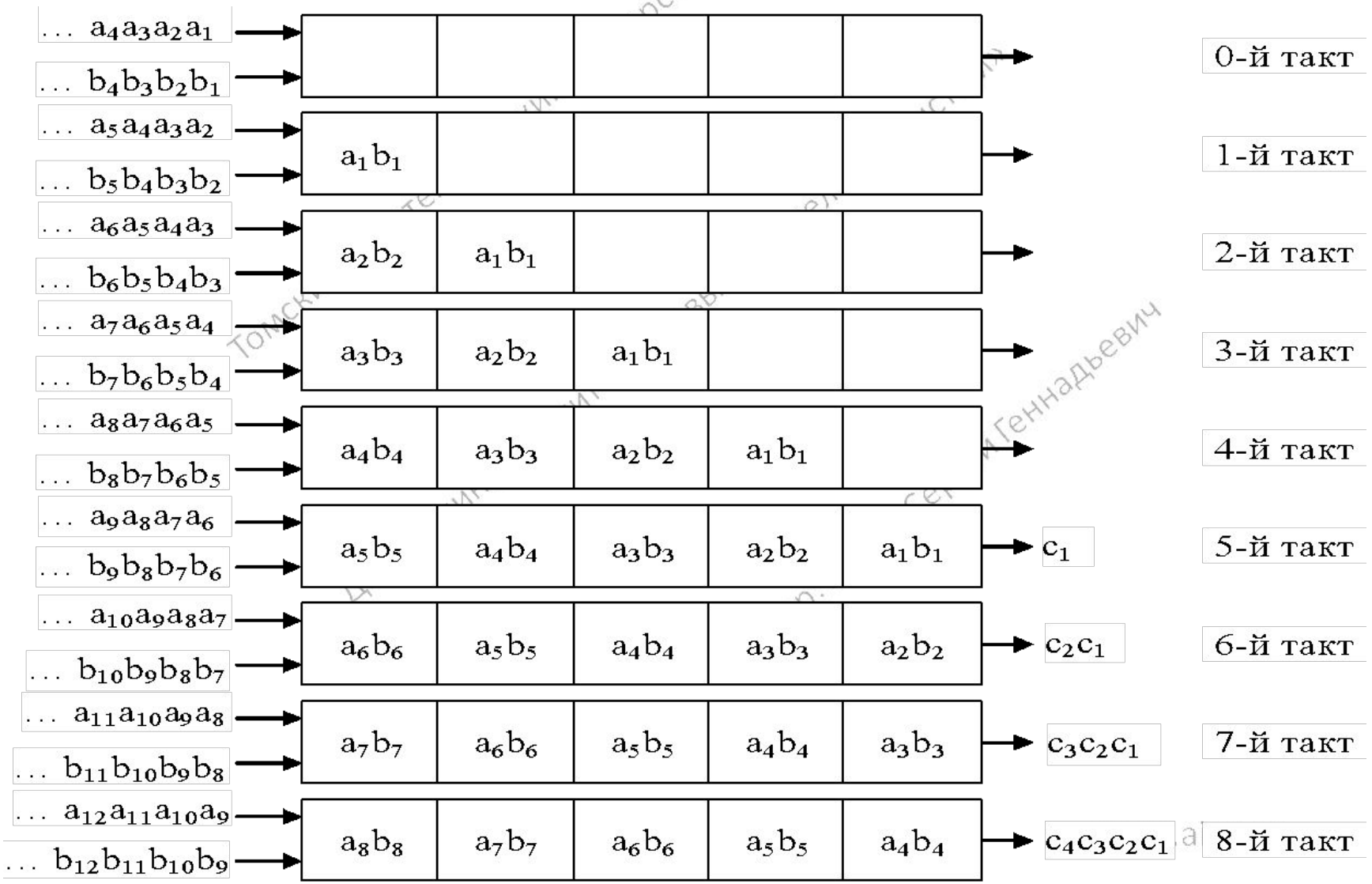
Суммирование векторов A=B+C с помощью последовательного устройства



Суммирование векторов A=B+C с помощью двух последовательных устройств



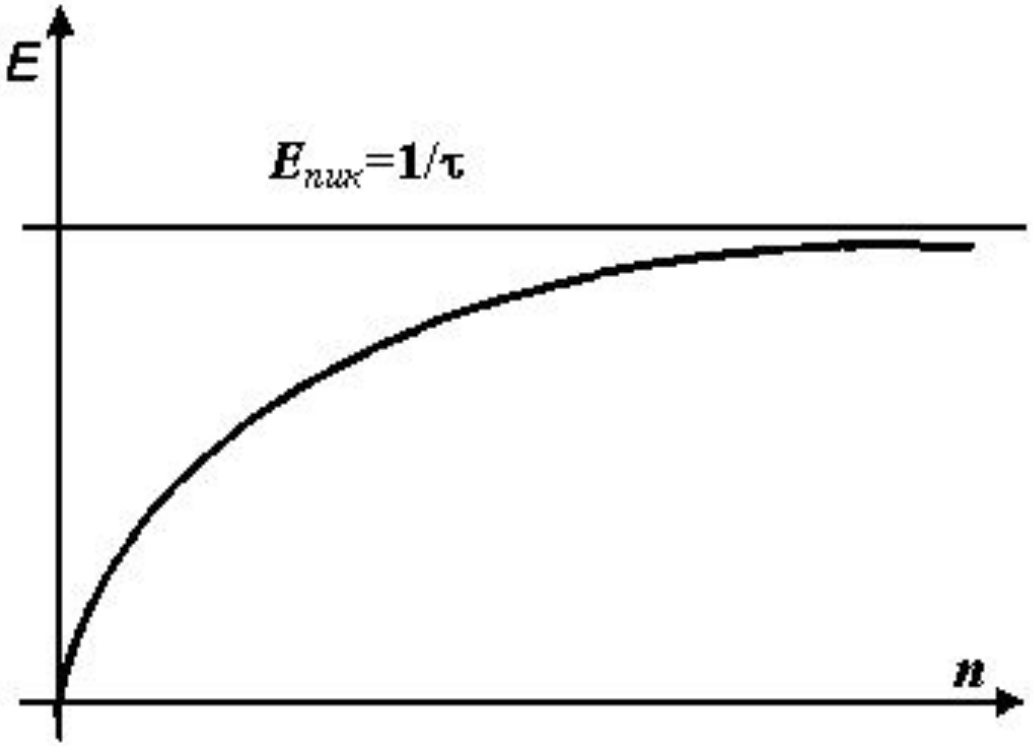
Суммирование векторов A=B+C с помощью конвейерного устройства



Эффективность конвейерной обработки

$$E = \frac{n}{t} = \frac{n}{(\sigma + L + n - 1)\tau} = \frac{1}{\tau + (\sigma + L - 1)\frac{\tau}{n}}$$

L – количество ступеней конвейера
τ – время такта работы конвейера
σ – время, необходимое для инициализации векторной команды



Повышение производительности за счет усовершенствования структуры ВС

- *Усовершенствование памяти:*

  разрядно-последовательная - разряды слова поступают для последующей обработки последовательно один за другим ;

  разрядно-параллельная - все разряды слова одновременно считываются из памяти и участвуют в выполнении операции арифметико-логическим устройством.
- *Повышение производительности за счет совмещения во времени различных этапов выполнения соседних команд:*

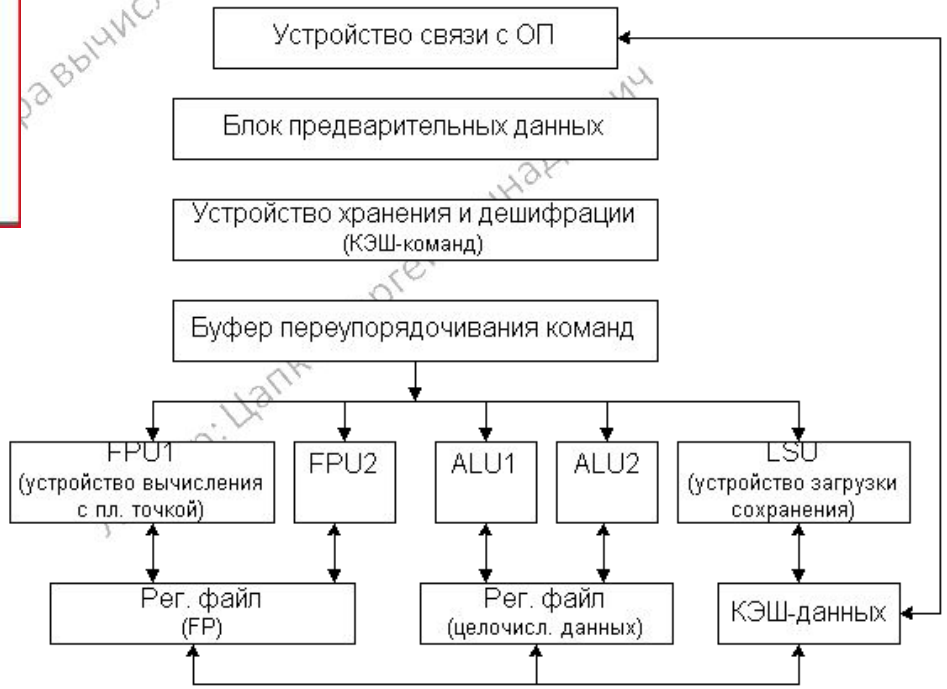
  опережающий просмотр для считывания, декодирования, вычисления адресов, а также предварительная выборка операндов нескольких команд;

  разбиение памяти на два и более независимых банка, способных передавать параллельно данные в АЛУ независимо друг от друга.
- *Конвейерный принцип обработки команд:*

  цикл обработки команды разбивается как минимум на N ступеней: выборка команды, вычисление адреса операнда, выборка операнда и выполнение операции.
- *Параллельное функционирование нескольких независимых функциональных устройств.*

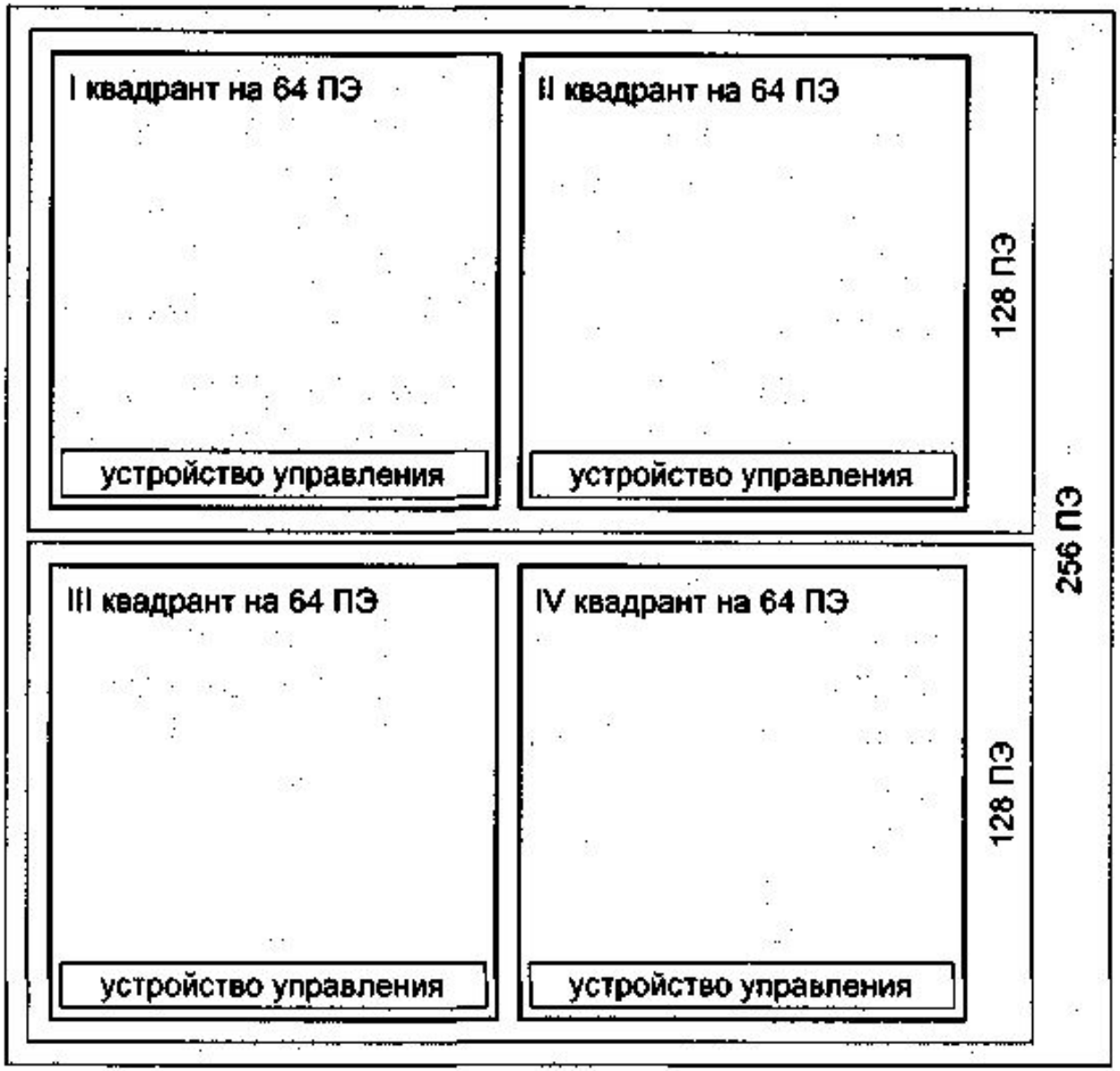
Расслоение памяти

Конвейерный принцип обработки



Параллельное функционирование нескольких независимых функциональных устройств

Матричные системы (структура ILLIAC IV)



Матричные вычислительные системы

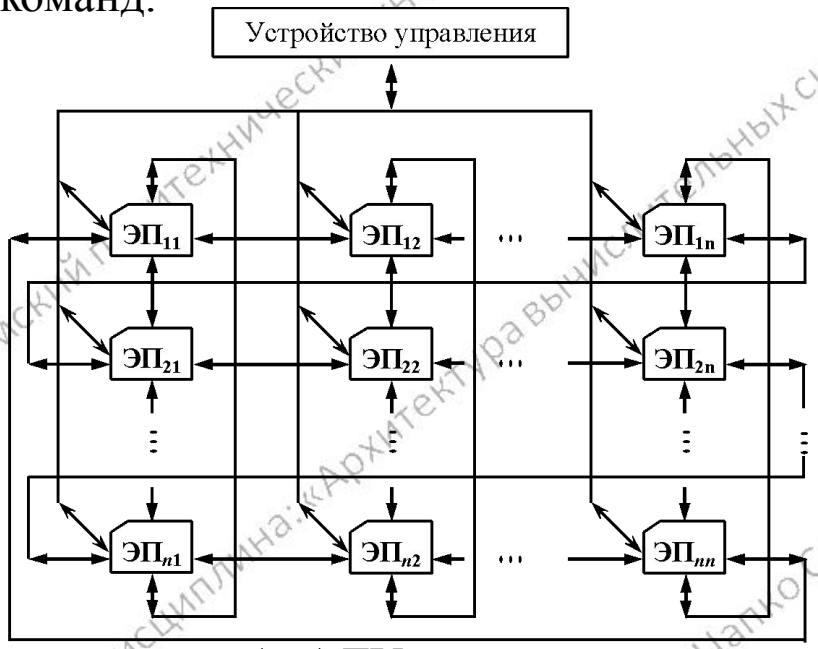
Матричные ВС обладают более широкими архитектурными возможностями, чем конвейерные ВС: их каноническая архитектура относится к классу SIMD.

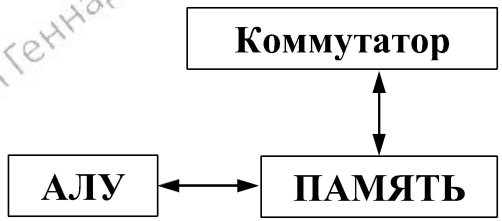
Матричные ВС относятся к классу систем с массовым параллелизмом (Massively Parallel Processing Systems) и, следовательно, не имеют принципиальных ограничений в наращивании своей производительности.

Матричные ВС предназначены для решения сложных задач, связанных с выполнением операций над векторами, матрицами и массивами данных (Data Arrays).

Функциональная структура матричного процессора

Матричный, или векторный *процессор* (Array Processor) представляет собой «матрицу» связанных идентичных элементарных процессоров (ЭП), управляемых одним потоком команд.





Каждый ЭП включает в себя АЛУ, память и локальный коммутатор. Сеть связей между ЭП (точнее, локальными коммутаторами) позволяет осуществлять обмен данными между любыми процессорами. Поток команд поступает на матрицу ЭП от единого устройства управления (SIMD-архитектура, в каноническом виде).

Функциональная структура матричного процессора

При решении сложных задач фактически один и тот же алгоритм параллельно (одновременно) реализуется над многими частями исходного массива данных.

Матричные процессоры ориентированы на работу в монопрограммном режиме (когда решается только одна задача, представленная в параллельной форме) .

Реализация мультипрограммных режимов в матричном процессоре осуществляется за счет разделения и «времени», и «пространства». В матричном процессоре имеется единственное устройство управления и множество ЭП, следовательно, в мультипрограммной ситуации должны «делиться» время устройства управления и элементарные процессоры («пространство») между программами.

Первый матричный компьютер

Первая матричная ВС SOLOMON (Simultaneous Operation Linked Ordinal MOdular Network — вычислительная сеть синхронно функционирующих упорядоченных модулей) была разработана в Иллинойском университете (University of Illinois) США под руководством Даниэля Слотника.

Работы по проекту SOLOMON велись с 1962 г., однако этот проект промышленного воплощения не нашел; в 1963 г. был создан лишь макет ВС размером 3x3 элементарных процессора. Позднее была построена конфигурация ВС размером 10x10 ЭП в фирме Westinghouse Electric Corp.

Время показало, что технология пока не достигла возможностей создания матричных ЭВМ.

Вычислительная система ILLIAC IV

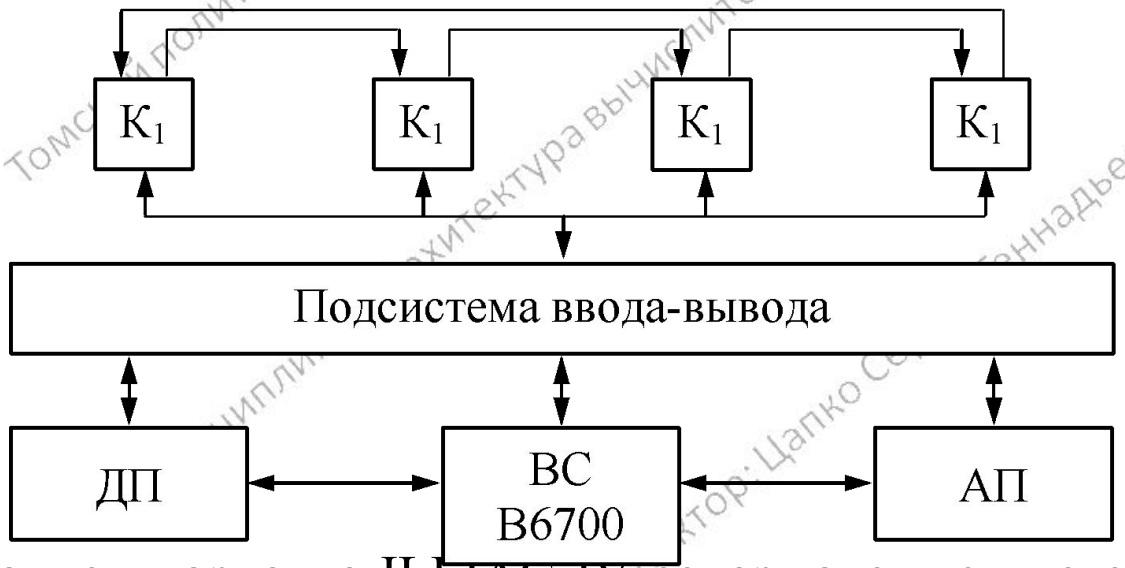
Матричная ВС ILLIAC IV создана Иллинойским университетом и корпорацией Бэрроуз (Burroughs Corporation). Работы по созданию ILLIAC IV были начаты в 1966 г. под руководством Д.Л. Слотника. Монтаж системы был закончен в мае 1972 г. в лаборатории фирмы Burroughs (Паоли, штат Панама), а установка для эксплуатации осуществлена в октябре 1972 г. в Научно-исследовательском центре НАСА им. Эймса в штате Калифорния (NASA's Ames Research Center; NASA - National Aeronautics and Space Administration - Национальное управление аэронавтики и космоса).

Количество процессоров в системе - 64;
быстродействие - $2 \cdot 10^8$ опер./с (Core 2 Duo до $3{,}1 \cdot 10^{11}$ опер./с);
емкость оперативной памяти - 1 Мбайт;
полезное время составляло 80...85 % общего времени работы ILLIAC IV,
стоимость 40 000 000 долл.,
вес - 75 т,
занимаемая площадь - 930 $м^2$.

Система ILLIAC IV была включена в вычислительную сеть ARPA (Advanced Research Projects Agency - Управление перспективных исследований и разработок Министерства обороны США) и успешно эксплуатировалась до 1981 г.

Функциональная структура системы ILLIAC IV

Матричная ВС ILLIAC IV (рис. 5.2) должна была состоять из четырех квадрантов ($К_1$-$К_4$) подсистемы ввода-вывода информации, ведущей ВС В6700 (или В6500), дисковой памяти (ДП) и архивной памяти (АП). Планировалось, что ВС обеспечит быстродействие 10^9 опер./с.



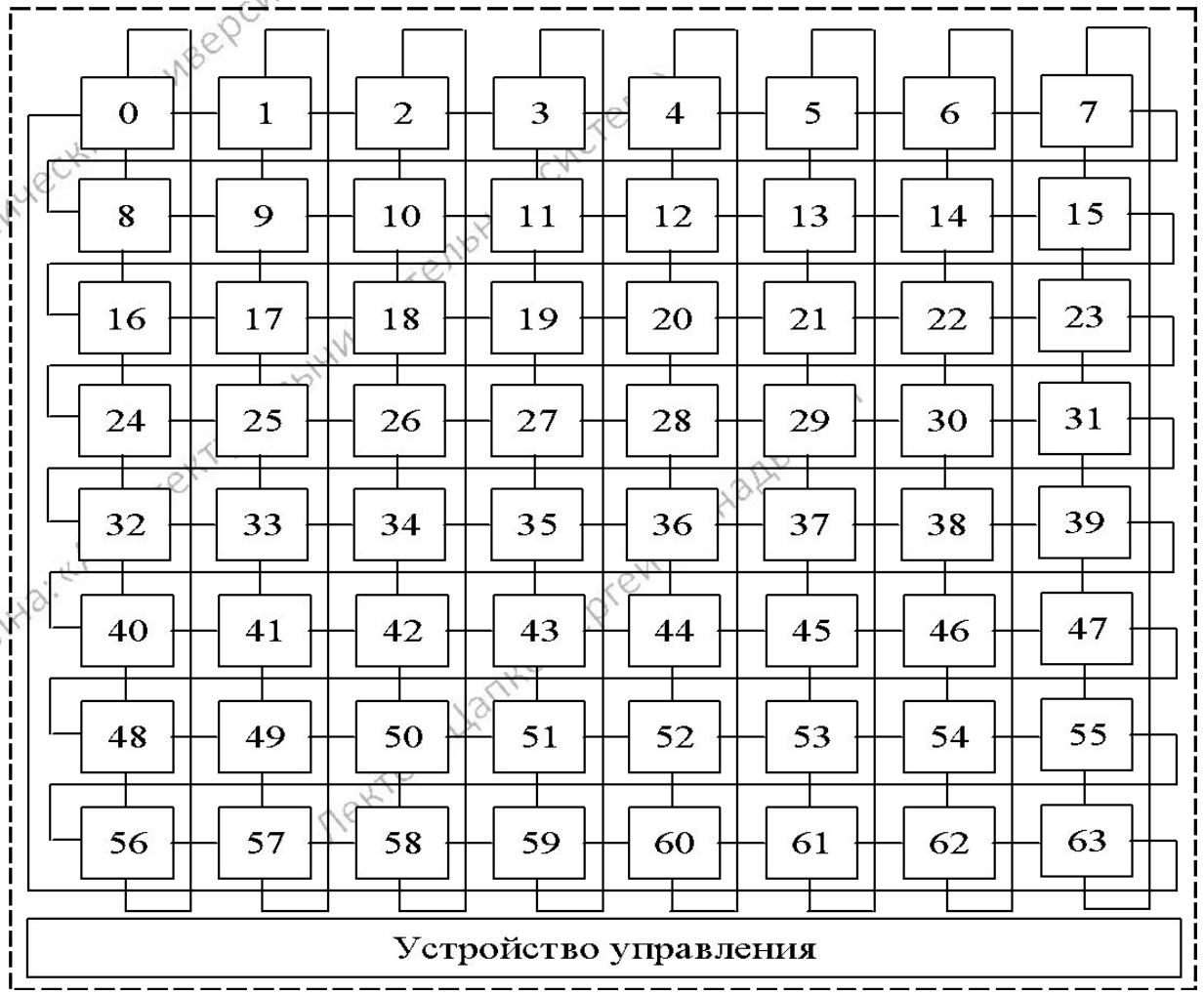
В реализованном варианте ILLIAC IV содержался только один квадрант, что обеспечило быстродействие $2 \cdot 10^8$ опер./с. При этом ILLIAC IV оставалась самой быстродействующей ВС вплоть до 80-х годов XX в.

Функциональная структура системы ILLIAC IV

Квадрант — матричный процессор, включавший в себя устройство управления и 64 ЭП. Устройство управления представляло собой специализированную ЭВМ, которая использовалась для выполнения операций над скалярами и формировала поток команд на матрицу ЭП.

Элементарные процессоры матрицы регулярным образом были связаны друг с другом. Структура квадранта системы ILLIAC IV представлялась двумерной решеткой, в которой граничные ЭП были связаны по канонической схеме.

Все 64 ЭП работали синхронно и единообразно. Допускалось одновременное выполнение скалярных и векторных операций.



Формат представления данных системы ILLIAC IV

В системе ILLIAC IV использовалось слово длиной 64 двоичных разряда. Числа могли представляться в следующих форматах:

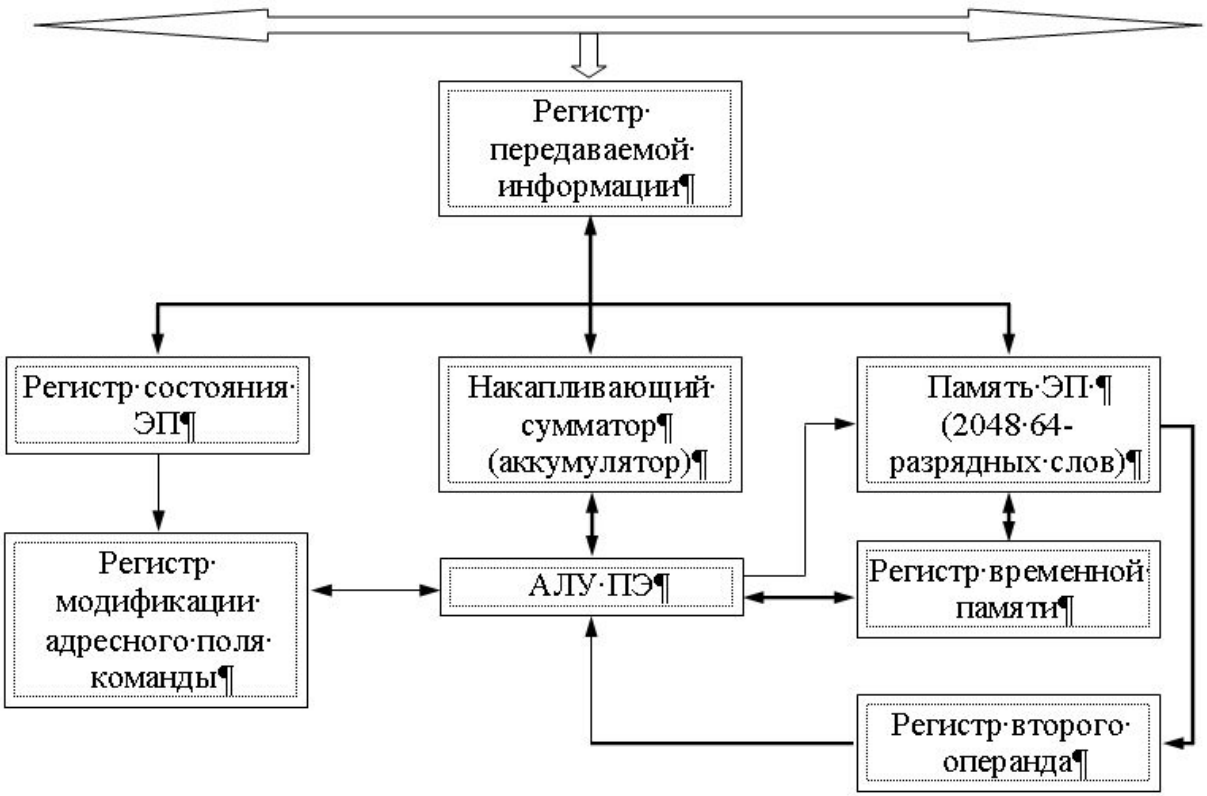
64 или 32 разряда с плавающей запятой;
48, или 24, или 8 разрядов с фиксированной запятой.

При использовании 64-, 32- и 8-разрядных форматов матрица из 64 ЭП могла обрабатывать векторы операндов, состоявшие из 64, 128 и 512 компонентов соответственно.

Система ILLIAC IV при суммировании 512 8-разрядных чисел имела быстродействие почти 10^{10} опер./с, а при сложении 64-разрядных чисел с плавающей запятой - $1,5 \cdot 10^{8}$ опер./с.

Компонентная структура системы ILLIAC IV

Элементарный процессор мог находиться в одном из двух состояний - активном или пассивном. В первом состоянии ему разрешалось, а во втором запрещалось выполнять команды, поступавшие из устройства управления.



Разрядность сумматора и всех регистров ЭП – 64 бита, регистр модификации адреса – 16 бит и состояния – 8 бит

Аппаратный состав системы ILLIAC IV

Подсистема ввода-вывода состояла из устройства управления, буферного запоминающего устройства и коммутатора.

Ведущая ВС B 6700 — мультипроцессорная система корпорации Burroughs; могла иметь в своем составе от 1 до 3 центральных процессоров и от 1 до 3 процессоров ввода-вывода информации и обладала быстродействием (1...3) • 10^6 опер./с.

Дисковая память (ДП) состояла из двух дисков и обрамляющих электронных схем, она имела емкость порядка 10^9 бит и была снабжена двумя каналами со скоростью 0,5 • 10^9 бит/с. Среднее время обращения к диску составляло 20 мс.

Архивная память (АП) — постоянная лазерная память с однократной записью, разработанная фирмой Precision Instrument Company емкостью 10^{12} бит. Имелось 400 информационных полосок по 2,9 млрд. бит, которые размещались на вращающемся барабане. Время поиска данных на любой из 400 полосок достигало 5 с; время поиска в пределах полоски – 200 нс. Существовало два канала обращения к архивной памяти, скорость считывания и записи данных по каждому из которых была равна 4 • 10^6 бит/с.

Программное обеспечение системы ILLIAC IV

Цель разработки ILLIAC IV — создание мощной ВС для решения задач с большим числом операций.

Программа структурно содержала три части:

- «Предпроцессорная» часть обеспечивала инициирование задачи и десятично-двоичные преобразования (последовательная форма)
- «Ядро» осуществляло решение поставленной задачи и представлялось в параллельной форме. Размер ядра составлял 5... 10 % полного объема программы (его исполнение на последовательной машине требовало 80...95 % рабочего времени)
- «Постпроцессорная» часть осуществляла запись результатов в архивные файлы, двоично-десятичные преобразования, вычерчивание графиков, вывод результатов на печать и т. п. (последовательная форма)

Программное обеспечение системы ILLIAC IV

Операционная система ILLIAC IV состояла из набора асинхронных программ, выполнявшихся под управлением главной управляющей программы B6700. Она работала в двух режимах: в первом режиме выполнялся контроль и диагностика неисправностей в квадранте и в подсистеме вводавывода информации; во втором – осуществлялось управление работой ILLIAC IV при поступлении на B6700 заданий от пользователей.

Задание для ILLIAC IV состояло из:

- Программы B6700, написанной, как правило, на языках ALGOL или FORTRAN и осуществлявшей подготовку (и преобразование) входных двоичных файлов («предпроцессорная» часть).
- Программы ILLIAC IV, обычно написанных на языках Glynpir или FORTRAN, которые использовались ILLIAC IV (составляли «ядро») для обработки файлов, подготовленных программами B6700, а также для формирования двоичных выходных файлов.
- Программы B6700 (на версиях языков ALGOL или FORTRAN), которые преобразовывали двоичные файлы ILLIAC IV в требуемый выходной формат («постпроцессорная» часть).
- Программа на управляющем языке Illiac, определявшая задание. Эта программа ориентировала операционную систему на работу, предусмотренную заданием.

Средства программирования системы ILLIAC IV

Средства программирования ILLIAC IV включали язык ассемблера (Assembler Language) и три языка высокого уровня: Tranquil, Glynpir, FORTRAN.

Язык ассемблера ILLIAC IV – традиционный язык программирования, адаптированный под архитектуру ВС. В частности, он имел сложные макроопределения, которые можно было применять для включения стандартных операций ввода-вывода и других операций связи между программами B6700 и ILLIAC IV.

Языки высокого уровня благодаря архитектурным особенностям ILLIAC IV отличались от соответствующих языков ЭВМ.

1. Распределение двумерной памяти. Была разрешена адресация отдельных слов в памяти ЭП и строк (из 64 слов) в пределах запоминающих устройств матрицы ЭП. Адресация по «столбцу» группы слов в памяти одного ЭП была недопустима.
2. Параллелизм и управление режимом обработки. Параллелизм ВС предопределяет работу с векторами данных. Следовательно, языки ILLIAC IV должны были допускать операции над векторами данных или строками матриц. Размерность вектора и количество подлежащих обработке элементов вектора определялись словами режима. Языки ILLIAC IV обеспечивали эффективную реализацию широкого круга вычислений и обработку слов режима.
3. Вид команд пересылок и индексации. Каждый из языков ILLIAC IV должен был содержать команды пересылок и индексации с различными приращениями в каждом элементарном процессоре.

Языки высокого уровня системы ILLIAC IV

Tranquil подобен языку ALGOL и полностью не зависел от архитектуры ILLIAC IV. Он был разработан для обеспечения параллельной обработки массивов информации.

Компилятор языка Tranquil потребовал больших системных затрат, связанных с маскированием архитектуры ILLIAC IV. Поэтому работы по созданию компилятора были приостановлены и предприняты шаги по модификации языка Tranquil.

Glynpir являлся языком также алгольного типа с блочной структурой. Он позволял опытному программисту использовать значительные возможности архитектуры ВС ILLIAC IV.

Язык FORTRAN был разработан для рядовых пользователей ILLIAC IV. Он в отличие от Glynpir освобождал пользователя от детального распределения памяти и предоставлял ему возможность мыслить в терминах строк любой длины. Он позволял путем применения модифицированных операторов ввода-вывода воспользоваться параллельной программой как последовательной и выполнить ее на одном ЭП. FORTRAN давал возможность отлаживать параллельные программы на одном элементарном процессоре.

Применение системы ILLIAC IV

Практически установлено, что ILLIAC IV была эффективна при решении широкого спектра сложных задач.

Классы задач: матричная арифметика, системы линейных алгебраических уравнений, линейное программирование, исчисление конечных разностей в одномерных, двумерных и трехмерных случаях, квадратуры (включая быстрое преобразование Фурье), обработка сигналов.

Классы задачи, обеспечивающих не полное использование ЭП: : движение частиц (метод Монте-Карло и т. д.), несимметричные задачи на собствен ные значения, нелинейные уравнения, отыскание корней полиномов.

Эффективность системы ILLIAC IV на примере решения типичной задачи линейного программирования, имеющей 4000 ограничений и 10 000 переменных, можно оценить по времени решения, которое составляло менее 2 мин, а для большой ЭВМ третьего поколения – 6...8 часов

Повышение интеллектуальности управления ЭВМ

- ***Поддержка параллелизма в аппаратно-программной среде ВС***

  Повышение эффективности операционных систем и компиляторов, технологии параллельного программирования и исполнения программ, а также поддержка параллелизма в процессоре и согласование особенностей работы с памятью
- ***Спецпроцессоры***

  реализация операций на уровне аппаратуры, операций выполняемых на уровне программного обеспечения;

  требуется обеспечить компромисс между универсальностью и специализированностью
- ***Параллелизм на уровне машинных команд***

  отсутствие у пользователя необходимости в специальном параллельном программировании;

  проблемы с переносимостью остаются на уровне общих проблем переносимости программ в классе последовательных машин.
  - *суперскалярные процессоры*

    Задача обнаружения параллелизма в машинном коде возлагается на аппаратуру;

    аппаратура строит соответствующую последовательность исполнения команд.
  - *VLIW-процессоры*

    Команда VLIW-процессора состоит из набора полей, каждое из которых отвечает за свою операцию;

    если какая-то часть процессора на данном этапе выполнения программы не востребована, то соответствующее поле команды не задействуется.

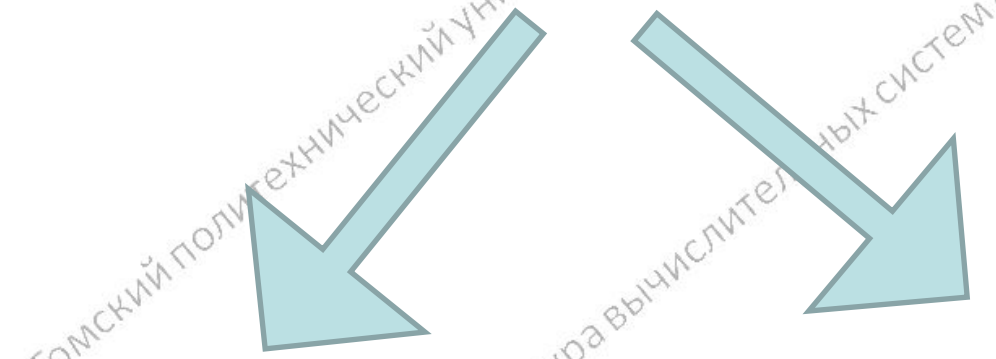
Ссылки в сети Internet

- Оценка производительности ВС
  http://www.osp.ru/os/1996/02/58.htm
  http://www.sdteam.com/index.php?id=5752
  http://freekniga7.narod.ru/sovremkomp/glava_3.htm

- Параллельная обработка данных
  http://www2.sscc.ru/Litera/vvv/Default.htm
  http://globus.smolensk.ru/user/sgma/MMORPH/N-3-html/23.htm
  http://www.ctc.msiu.ru/program/t-system/diploma/node5.html

- Конвейерная обработка данных
  http://www.ctc.msiu.ru/program/t-system/diploma/node5.html
  http://www.macro.aaanet.ru/apnd_4.html

Вычислительные системы



Компьютерные с общей памятью (мультипроцессорные системы)

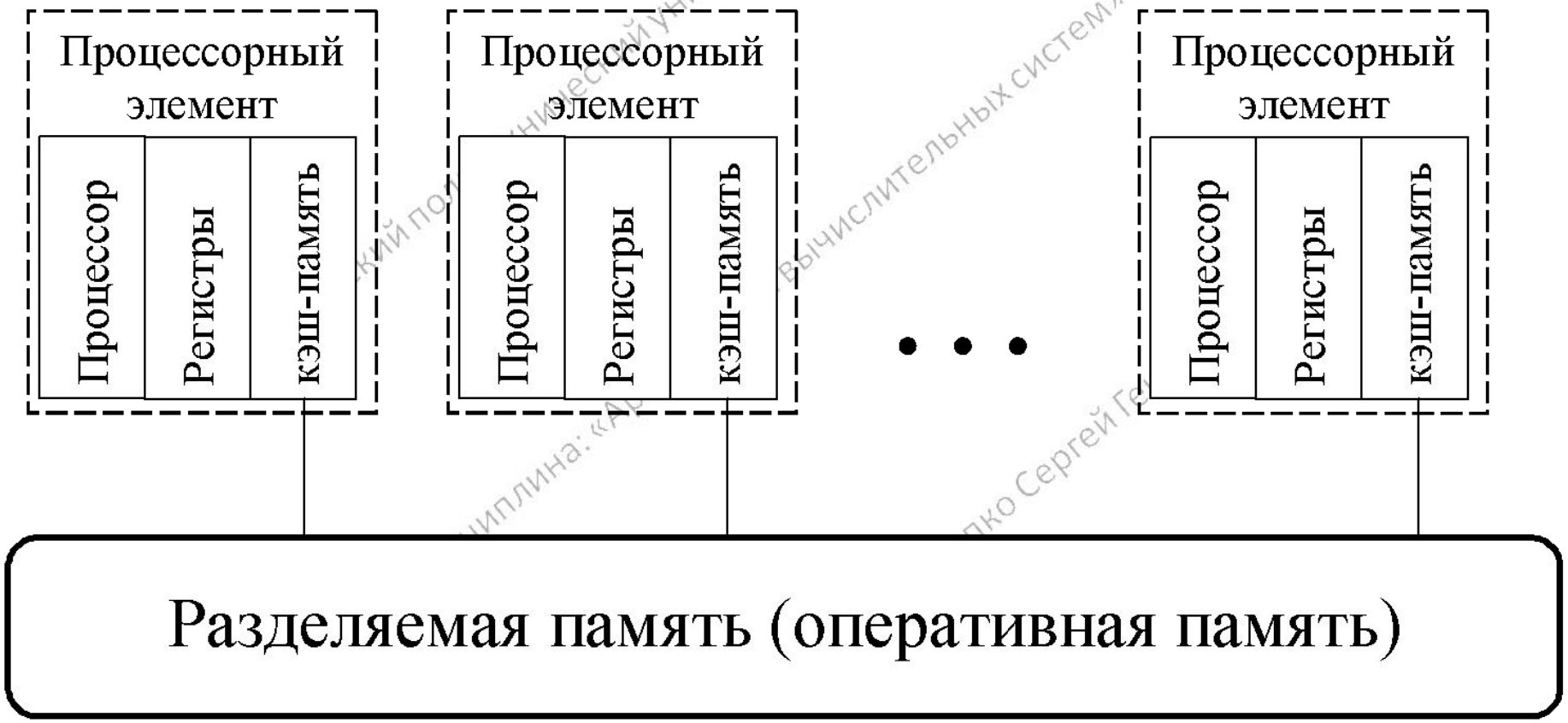
Компьютерные с распределенной памятью (мультикомпьютерные системы)

Мультипроцессорные системы

- Первый класс – это компьютеры с общей памятью. Системы, построенные по такому принципу, иногда называют мультипроцессорными "системами или просто мультипроцессорами. В системе присутствует несколько равноправных процессоров, имеющих одинаковый доступ к единой памяти. Все процессоры "разделяют" между собой общую память.

  Все процессоры работают с единым адресным пространством: если один процессор записал значение 79 в ячейку по адресу 1024, то другой процессор, прочитав содержимое ячейки, расположенное по адресу 1024, получит значение 79.

Параллельные компьютеры с общей памятью



Мультикомпьютерные системы

- Второй класс — это компьютеры с распределенной памятью, которые по аналогии с предыдущим классом иногда называют мультикомпьютерными системами. Каждый вычислительный узел является полноценным компьютером со своим процессором, памятью, подсистемой ввода/вывода, операционной системой.

  В такой ситуации, если один процессор запишет значение 79 по адресу 1024, то это никак не повлияет на то, что по тому же адресу прочитает другой, поскольку каждый из них работает в своем адресном пространстве.

Параллельные компьютеры с распределенной памятью

Blue Gene/L

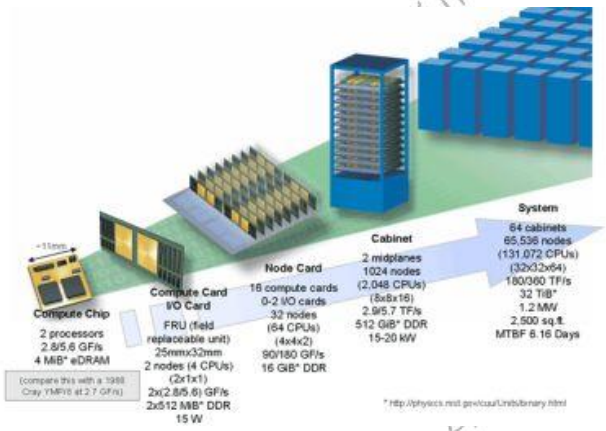
Расположение:
Ливерморская национальная лаборатория имени Лоуренса

Общее число процессоров 65536 штук

Состоит из 64 стоек

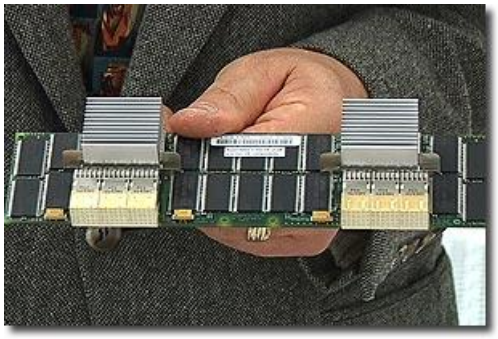
Производительность 280,6 терафлопс

В штате лаборатории - порядка 8000 сотрудников, из которых - более 3500 ученых и инженеров.



Машина построена по сотовой архитектуре, то есть, из однотипных блоков, что предотвращает появление "узких мест" при расширении системы.

Стандартный модуль BlueGene/L - "compute card" - состоит из двух блоков-узлов (node), модули группируются в модульную карту по 16 штук, по 16 модульных карт устанавливаются на объединительной панели (midplane) размером 43,18 x 60,96 x 86,36 см, при этом каждая такая панель объединяет 512 узлов. Две объединительные панели монтируются в серверную стойку, в которой уже насчитывается 1024 базовых блоков-узлов.



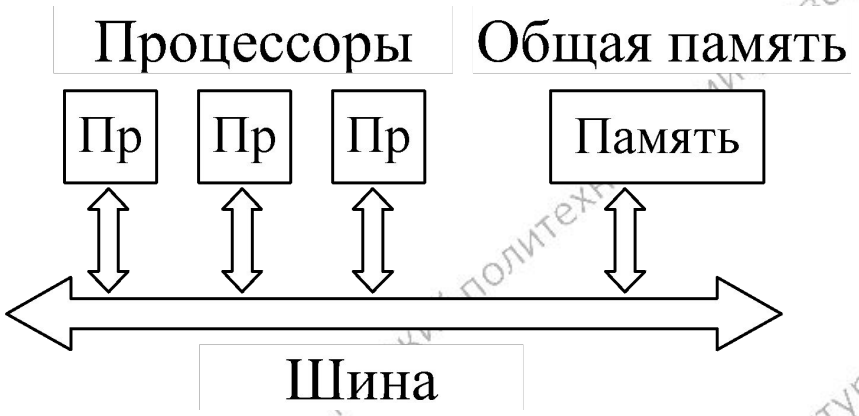
На каждом вычислительном блоке (compute card) установлено по два центральных процессора и по четыре мегабайта выделенной памяти

Процессор PowerPC 440 способен выполнять за такт четыре операции с плавающей запятой, что для заданной тактовой частоты соответствует пиковой производительности в 1,4 терафлопс для одной объединительной панели (midplane), если считать, что на одном узле установлено по одному процессору. Однако на каждом блоке-узле имеется еще один процессор, идентичный первому, но он призван выполнять телекоммуникационные функции.

Задачи параллельных вычислений

- Построении вычислительных систем с максимальной производительностью
  - компьютеры с распределенной памятью
  - единственным способом программирования подобных систем является использование систем обмена сообщениями
- Поиск методов разработки эффективного программного обеспечения для параллельных вычислительных систем
  - компьютеры с общей памятью
  - технологии программирования проще
  - по технологическим причинам не удается объединить большое число процессоров с единой оперативной памятью
  - проблемным звеном является система коммутации

Организация мультипроцессорных систем (общая шина)



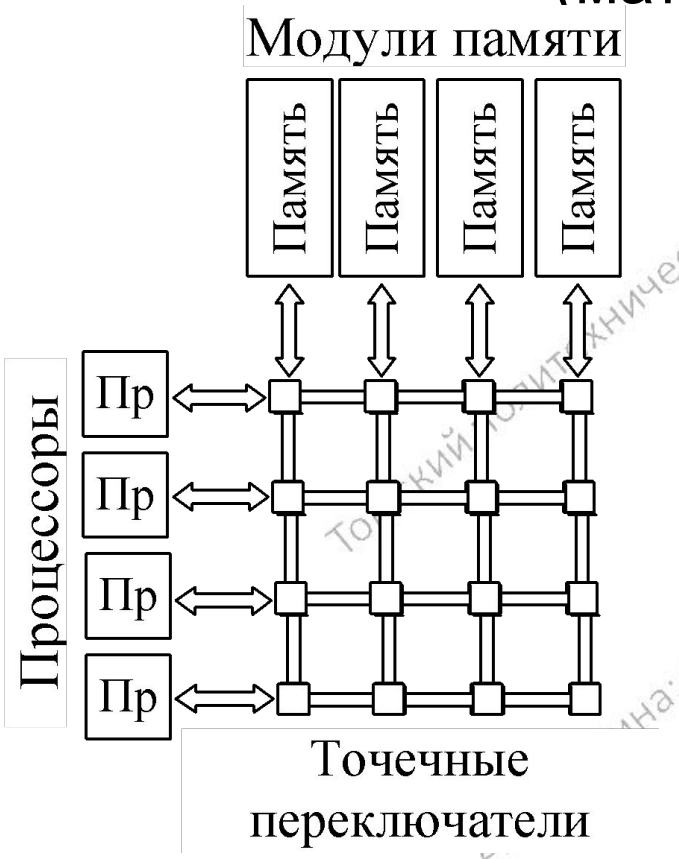
Мультипроцессорная система с общей шиной

Чтобы предотвратить одновременное обращение нескольких процессоров к памяти, используются схемы арбитража, гарантирующие монопольное владение шиной захватившим ее устройством.

Недостаток: заключается в том, что даже небольшое увеличение числа устройств на шине (4-5) очень быстро делает ее узким местом, вызывающим значительные задержки при обменах с памятью и катастрофическое падение производительности системы в целом

Достоинство: простота и дешевизна конструкции

Организация мультипроцессорных систем (матричный коммутатор)



Мультипроцессорная система с матричным коммутатором

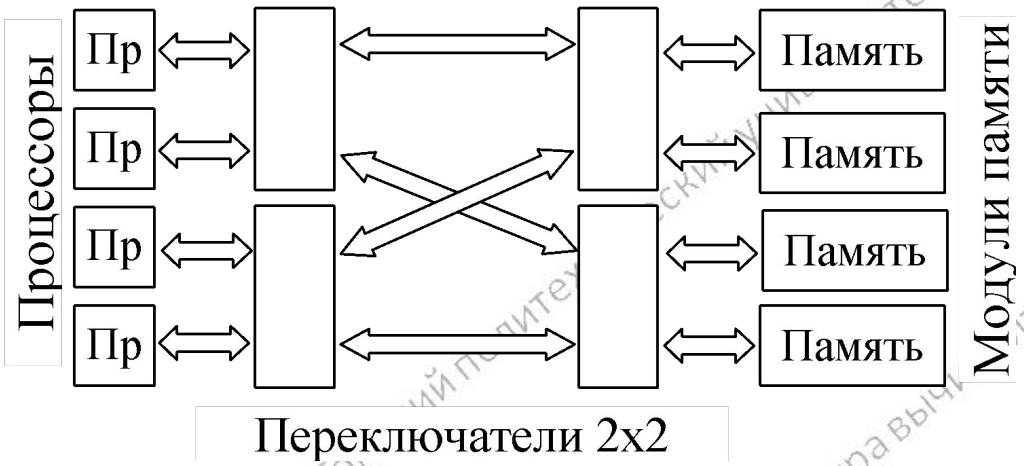
Матричный коммутатор позволяет разделить память на независимые модули и обеспечить возможность доступа разных процессоров к различным модулям одновременно.

На пересечении линий располагаются элементарные точечные переключатели, разрешающие или запрещающие передачу информации между процессорами и модулями памяти.

Преимущества:
возможность одновременной работы процессоров с различными модулями памяти

Недостаток:
большой объем необходимого оборудования, поскольку для связи n процессоров с n модулями памяти требуется

Организация мультипроцессорных систем



Мультипроцессорная система с омега-сетью

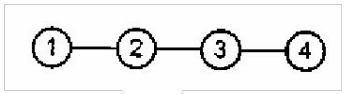
Использование *каскадных переключателей*

Проблема: задержки

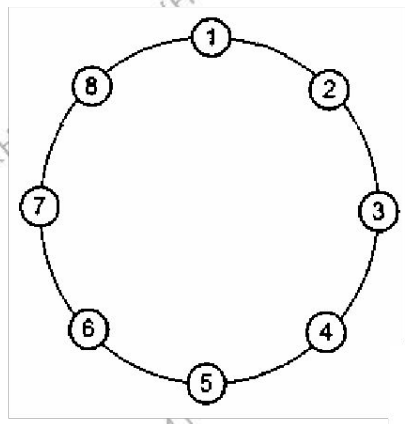
Каждый использованный коммутатор может соединить любой из двух своих входов с любым из двух своих выходов. Это свойство и использованная схема коммутации позволяют любому процессору вычислительной системы обращаться к любому модулю памяти.

В общем случае для соединения *n* процессоров с *n* модулями памяти потребуется $\log_2 n$ каскадов по *n*/2

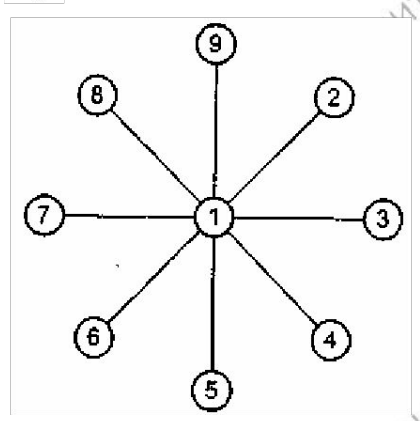
Топологические связи модулей ВС



а)



б)

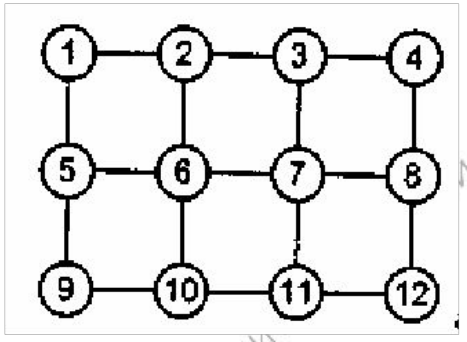


в)

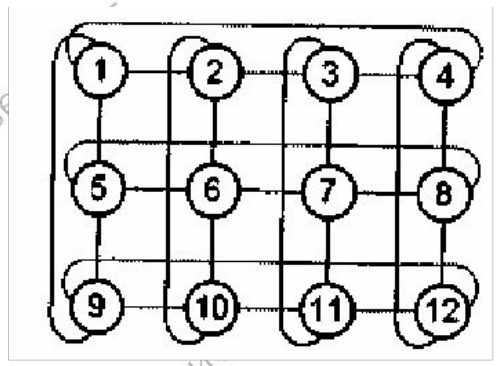
а – линейка; б – кольцо; в – звезда

Выбор той топологии связи процессоров в конкретной вычислительной системе может быть обусловлен самыми разными причинами. Это могут быть соображениями стоимости, технологической реализуемости, простоты сборки и программирования, надежности,

Варианты топологий связи процессоров и ВМ



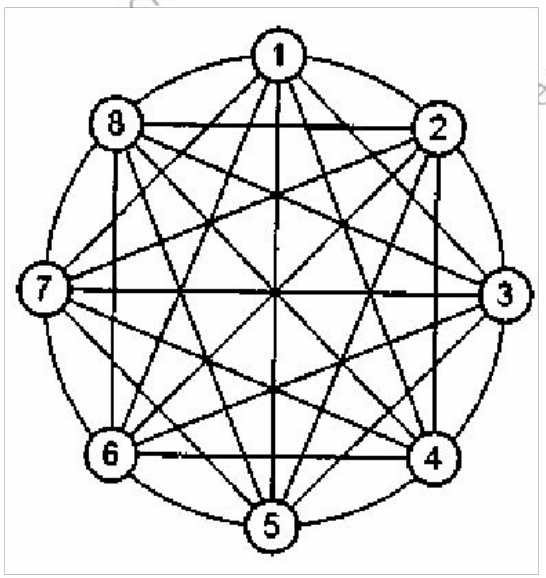
а)



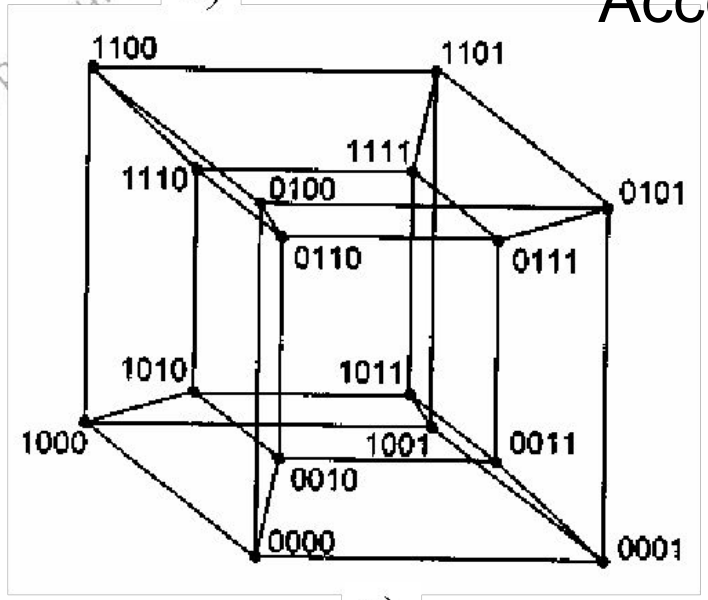
б)

NUMA

Non Uniform Memory Access



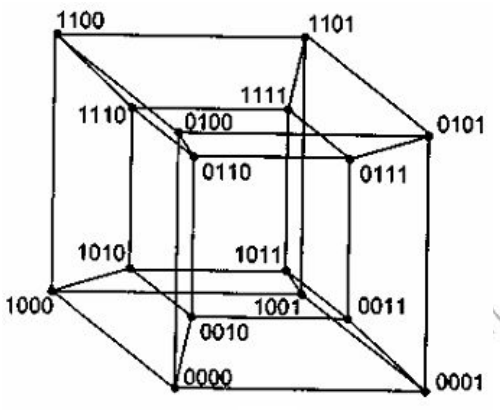
в)



г)

а – решетка; б – 2-тор; в – полная связь; г – гиперкуб

Топология двоичного гиперкубы



В n-мерном пространстве в вершинах единичного n-мерного куба размещаются процессоры системы, т. е. точки (x_1, x_2, …, x_n), в которых все координаты x_i могут быть равны либо 0, либо 1. Каждый процессор соединим с ближайшим непосредственным соседом вдоль каждого из n измерений. В результате получается n-мерный куб для системы из $N = 2^n$ процессоров. Двумерный куб соответствует простому квадрату, а четырехмерный вариант условно изображен на рисунке. В гиперкубе каждый процессор связан лишь с $\log_2 N$ непосредственными соседями, а не с N, как в случае полной связности.

Гиперкуб имеет массу полезных свойств. Например, для каждого процессора очень просто определить всех его соседей: они отличаются от него лишь значением какой-либо одной координаты x_i. Каждая "грань" n-мерного гиперкуба является гиперкубом размерности n-1. Максимальное расстояние между вершинами n-мерного гиперкуба равно n. Гиперкуб

Достоинства и недостатки компьютеров с общей и распределенной памятью

Для компьютеров с общей памятью проще создавать параллельные программы, но их максимальная производительность сильно ограничивается небольшим числом процессоров.

Для компьютеров с распределенной памятью все наоборот.

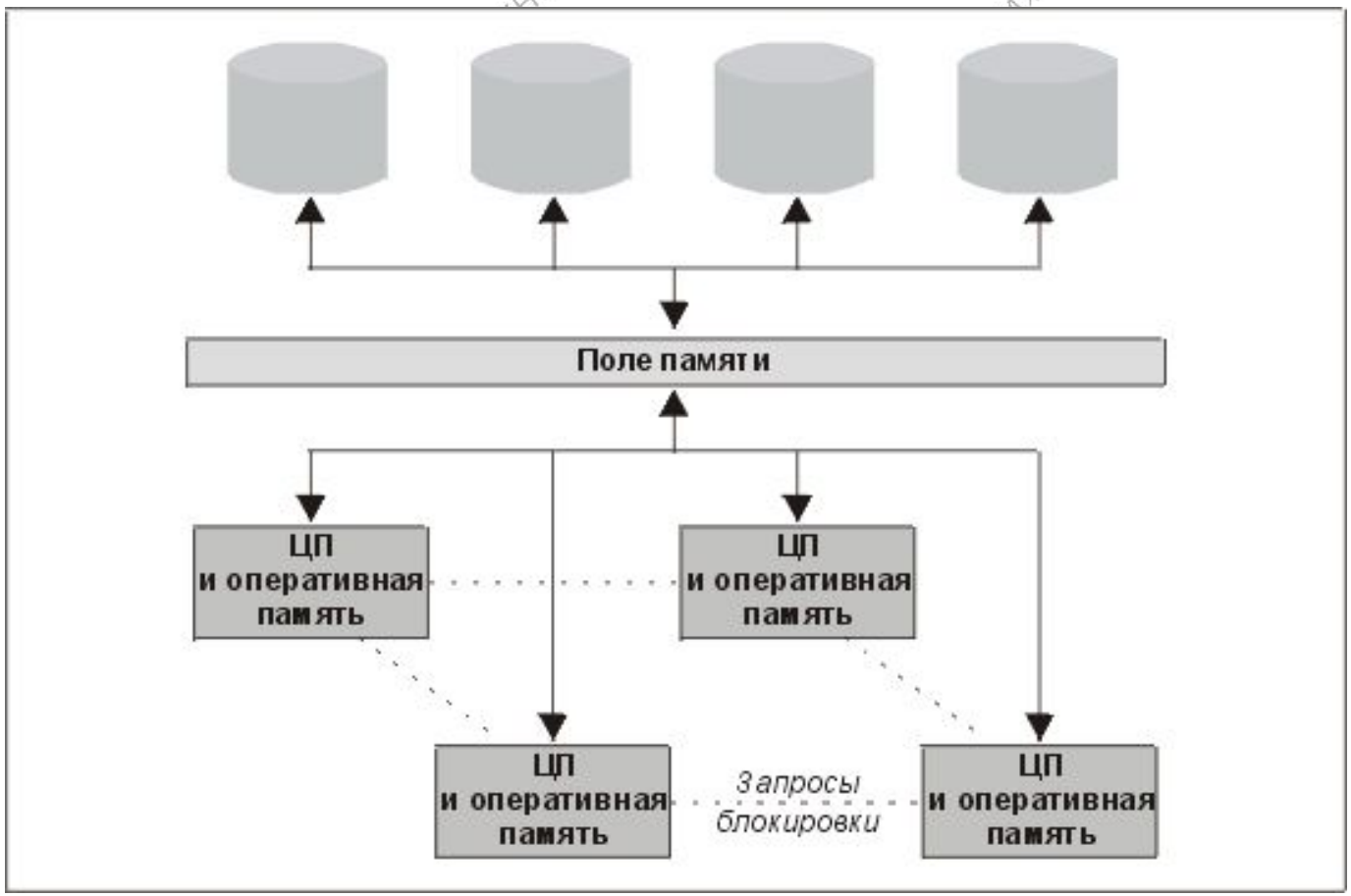
Одним из возможных направлений объединения достоинств этих двух классов является проектирование компьютеров с архитектурой NUMA (Non Uniform Memory Access).

Данный компьютер состоит из набора кластеров, соединенных друг с другом через межкластерную шину. Каждый кластер объединяет процессор, контроллер памяти, модуль памяти и иногда некоторые устройства ввода/вывода, соединенные между собой посредством локальной шины.

Когда процессору нужно выполнить операции чтения или записи, он посылает запрос с нужным адресом своему контроллеру памяти. Контроллер анализирует старшие разряды адреса, по которым и определяет, в каком модуле хранятся нужные данные. Если адрес локальный, то запрос выставляется на локальную шину, в противном случае запрос для удаленного кластера отправляется через межкластерную шину.

В таком режиме программа, хранящаяся в одном модуле памяти, может выполняться любым процессором системы. Единственное различие заключается в скорости выполнения. Все локальные ссылки отрабатываются намного быстрее, чем удаленные. Поэтому и процессор того кластера, где хранится программа, выполнит ее на порядок быстрее, чем любой другой.

Простая конфигурация с архитектурой NUMA



NUMA - архитектура

NUMA-компьютеры обладают серьезным недостатком, который выражается в наличии отдельной кэш-памяти у каждого процессорного элемента

Кэш-память для многопроцессорных систем оказывается узким местом

Объяснение:

Если процессор P1 сохранил значение X в ячейке q, а затем процессор P2 хочет прочитать содержимое той же ячейки q. Процессор P2 получит результат отличный от X, так как X попало в кэш процессора P1.

Эта проблема носит название проблемы согласования содержимого кэш-памяти

Решение:

Архитектура ccNUMA

Проблема неоднородности доступа

Архитектура NUMA имеет неоднородную память (распределенность памяти между модулями), что в свою очередь требует от пользователя понимания неоднородности архитектуры. Если обращение к памяти другого узла требует на 5-10% больше времени, чем обращение к своей памяти, то это может и не вызвать никаких вопросов. Большинство пользователей будут относиться к такой системе, как к UMA (SMP), и практически все разработанные для SMP программы будут работать достаточно хорошо. Однако для современных NUMA систем это не так, и разница времени локального и удаленного доступа лежит в промежутке 200-700%.

Языки параллельного программирования

- Специальные комментарии:
  внедрение дополнительных директив для компилятора, использование данных директив в процессе написания программы для указания компилятору параллельных участков программы.
  Использование спецкомментариев не только добавляет возможность параллельного исполнения, но и полностью сохраняет исходный вариант программы. Если компилятор ничего не знает о параллелизме, то все спецкомментарии он просто пропустит, взяв за основу последовательную семантику программы.
  Пример: стандарт OpenMP
  для Fortran - !$OPM
  для C – директива “#progma opm”
- Расширение существующих языков программирования (ЯП):
  разработка на основе существующих ЯП новых языков, путем добавления набора команд параллельной обработки информации, либо модификации системы компиляции и выполнения программы.
  Пример: язык High Performance Fortran (HPF)
- Разработка специальных языков программирования:
  использование ЯП годных для использования только для многомашинных и многопроцессорных комплексов. В данных ЯП параллелизм заложен на уровне алгоритмизации и выполнения программы.
  Пример: языки Occam (для программирования транспьютерных систем),
  Sisal (для программирования потоковых машин),
  Норма (декларативный язык для описания решения вычислительных задач сеточными методами)

Языки параллельного программирования

- Использование библиотек и интерфейсов, поддерживающих взаимодействие параллельных процессов:
  подготовка программного кода на любом доступном языке программирования, но с использованием интерфейса доступа к свойствам и методам, обеспечивающим параллельную обработку информации.
  Программист сам явно определяет какие параллельные процессы приложения в каком месте программы и с какими процессами должны либо обмениваться данными, либо синхронизировать свою работу.
  Такой идеологии следуют MPI и PVM
  Существует специализированная система Linda, добавляющая в любой последовательный язык лишь четыре дополнительные функции in, out, read и eval, что и позволяет создавать параллельные программы
- Использование подпрограмм и функций параллельных предметных библиотек в критических по времени счета фрагментах программы:
  использование дополнительных модулей, подключаемых к стандартному ЯП в процессе подготовки программного кода, позволяющие обеспечить параллельное функционирования программы только для некоторого набора алгоритмов.
  Весь параллелизм и вся оптимизация спрятаны в вызовах, а пользователю остается лишь написать внешнюю часть своей программы и грамотно воспользоваться стандартными блоками.
  Примеры библиотек: Lapack, Cray Scientific Library, HP Mathematical Library
- Использование специализированных пакетов и программных комплексов:
  применяются в основном для выполнения типовых задач и не требуют от пользователя каких-либо знаний программирования, либо архитектуры ВС.
  Основная задача — это правильно указать все необходимые входные данные и правильно воспользоваться функциональностью пакета.
  Пример: пакет GAMESS для выполнения квантово-химических расчетов

Примеры языков программирования и надстроек

1. OpenMP
2. High Performance Fortran (HPF)
3. Occam, Sisal, Норма
4. Linda, Massage Passing Interface (MPI)
5. Lapack,
6. Gamess

Массивно-параллельная архитектура

Массивно-параллельная архитектура (англ. MPP, Massive Parallel Processing) — класс архитектур параллельных вычислительных систем. Особенность архитектуры состоит в том, что память физически разделена.

Система строится из отдельных модулей, содержащих процессор, локальный банк операционной памяти, коммуникационные процессоры или сетевые адаптеры, иногда — жесткие диски и/или другие устройства ввода/вывода. Доступ к банку операционной памяти из данного модуля имеют только процессоры из этого же модуля. Модули соединяются специальными коммуникационными каналами.

Используются два варианта работы операционной системы на машинах MPP-архитектуры. В одном полноценная операционная система работает только на управляющей машине (front-end), на каждом отдельном модуле функционирует сильно урезанный вариант ОС, обеспечивающий работу только расположенной в нем ветви параллельного приложения. Во втором варианте на каждом модуле работает полноценная UNIX-подобная ОС, устанавливаемая отдельно.

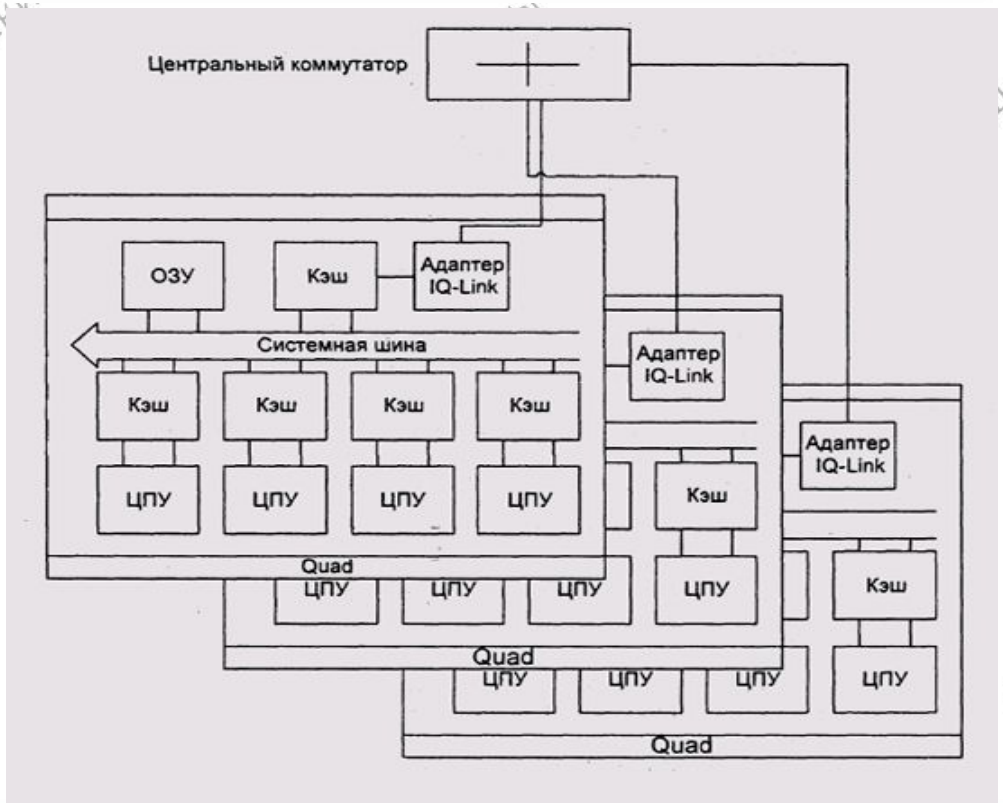
Преимущества архитектуры

Главным преимуществом систем с раздельной памятью является хорошая масштабируемость: в отличие от SMP-систем, в машинах с раздельной памятью каждый процессор имеет доступ только к своей локальной памяти, в связи с чем не возникает необходимости в потактовой синхронизации процессоров. Практически все рекорды по производительности на сегодня устанавливаются на машинах именно такой архитектуры, состоящих из нескольких тысяч процессоров (ASCI Red, ASCI Blue Pacific)

Недостатки архитектуры

- отсутствие общей памяти заметно снижает скорость межпроцессорного обмена, поскольку нет общей среды для хранения данных, предназначенных для обмена между процессорами. Требуется специальная техника программирования для реализации обмена сообщениями между процессорами;
- каждый процессор может использовать только ограниченный объем локального банка памяти;
- вследствие указанных архитектурных недостатков требуются значительные усилия для того, чтобы максимально использовать системные ресурсы. Именно этим определяется высокая цена программного обеспечения для массивно-параллельных систем с раздельной памятью.


Оперативная память
Оперативная память
Оперативная память
CACHE-память
CACHE-память
CACHE-память
Процессор
Процессор
Процессор
Коммуникационная среда


Центральный коммутатор
ОЗУ
Кэш
Адаптер IQ-Link
Системная шина
Кэш
Кэш
Кэш
Кэш
ЦПУ
ЦПУ
ЦПУ
ЦПУ
Адаптер IQ-Link
Адаптер IQ-Link
Кэш
Кэш
Quad
ЦПУ
ЦПУ
ЦПУ
ЦПУ
Quad
ЦПУ
ЦПУ
ЦПУ
ЦПУ
Quad

Основные классы современных параллельных компьютеров

Массивно-параллельные системы (MPP)

Архитектура

Система состоит из однородных *вычислительных узлов*, включающих:

- один или несколько центральных процессоров (обычно RISC),
- локальную память (прямой доступ к памяти других узлов невозможен),
- коммуникационный процессор или сетевой адаптер
- жесткие диски и/или другие устройства В/В

К системе могут быть добавлены специальные узлы ввода-вывода и управляющие узлы. Узлы связаны через некоторую коммуникационную среду (высокоскоростная сеть, коммутатор и т.п.)

Примеры

- IBM RS/6000 SP2IBM RS/6000 SP2, Intel PARAGON/ASCI Red, SGI/CRAY T3EIBM RS/6000 SP2, Intel PARAGON/ASCI Red, SGI/CRAY T3E, Hitachi SR8000IBM RS/6000 SP2, Intel PARAGON/ASCI Red, SGI/CRAY T3E, Hitachi SR8000, транспьютерные системы Parsytec.

Масштабируемость

- *(Масштабируемость представляет собой возможность наращивания числа и мощности процессоров, объемов оперативной и внешней памяти и других ресурсов вычислительной системы. Масштабируемость должна обеспечиваться архитектурой и конструкцией компьютера, а также соответствующими средствами программного обеспечения.)*

  Общее число процессоров в реальных системах достигает нескольких тысяч (ASCI Red, Blue Mountain).

Операционная система

Существуют два основных варианта:

- Полноценная ОС работает только на управляющей машине (front-end), на каждом узле работает сильно урезанный вариант ОС, обеспечивающие только работу расположенной в нем ветви параллельного приложения. Пример: Cray T3E.
- На каждом узле работает полноценная UNIX-подобная ОС (вариант, близкий к кластерному подходу). Пример: IBM RS/6000 SP + ОС AIX, устанавливаемая отдельно на каждом узле.

Модель программирования

- Программирование в рамках модели передачи сообщений (MPIПрограммирование в рамках модели передачи сообщений (MPI, PVMПрограммирование в рамках модели передачи сообщений (MPI, PVM, BSPlib)

Симметричное мультипроцессирование

SMP часто применяется в науке, промышленности, бизнесе, где программное обеспечение специально разрабатывается для многопоточного выполнения. В то же время, большинство потребительских продуктов, таких как текстовые редакторы и компьютерные игры написаны так, что они не могут получить много пользы от SMP систем.

Преимущества архитектуры

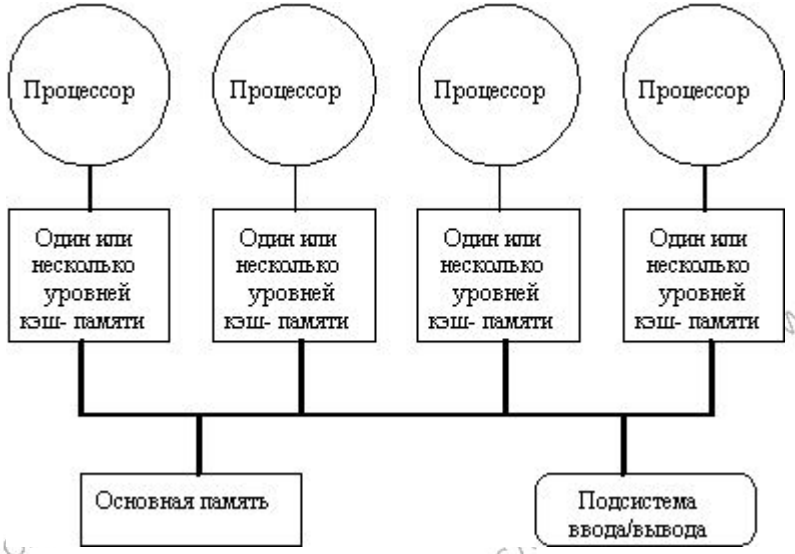
Пограммы, запущенные на SMP системах, получают прирост производительности даже если они были написаны для однопроцессорных систем. Это связано с тем, что аппаратные прерывания, обычно приостанавливающие выполнение программы для их обработки ядром, могут обрабатываться на свободном процессоре. Эффект в большинстве приложений проявляется не столько в приросте производительности, сколько в ощущении, что программа выполняется более плавно. В некоторых приложениях, в частности программных компиляторах и некоторых проектах распределённых вычислений, повышение производительности будет почти прямо пропорционально числу дополнительных процессоров.

Недостатки архитектуры

- Ограничение на количество процессоров

  При увеличении числа процессоров заметно увеличивается требование к полосе пропускания шины памяти. Это накладывает ограничение на количество процессоров в SMP архитектуре. Современные конструкции позволяют разместить до четырех процессоров на одной системной плате.
- Проблема когерентности кэша

  Возникает из-за того, что значение элемента данных в памяти, хранящееся в двух разных процессорах, доступно этим процессорам только через их индивидуальные кэши. Если процессор изменит значение элемента данных в своем кэше, то при попытке вывода данных из памяти, будет получено старое значение. Наоборот, если подсистема ввода/вывода вводит в ячейку основной памяти новое значение, в кэш памяти процессора по прежнему остается старо.


Процессор
Процессор
Процессор
Процессор
Один или несколько уровней кэш- памяти
Один или несколько уровней кэш- памяти
Один или несколько уровней кэш- памяти
Один или несколько уровней кэш- памяти
Основная память
Подсистема ввода/вывода


Процессор
Процессор
Кэш-память
Кэш-память
Коммуникационная система
Основная память
Система ввода/вывода

Основные классы современных параллельных компьютеров

Симметричные мультипроцессорные системы (SMP)

Архитектура

- Система состоит из нескольких однородных процессоров и массива общей памяти (обычно из нескольких независимых блоков). Все процессоры имеют доступ к любой точке памяти с одинаковой скоростью. Процессоры подключены к памяти либо с помощью общей шины (базовые 2-4 процессорные SMP-сервера), либо с помощью crossbar-коммутатора (HP 9000). Аппаратно поддерживается когерентность кэшей.

Примеры

- HP 9000 V-class, N-class; SMP-сервера и рабочие станции на базе процессоров Intel (IBM, HP, Compaq, Dell, ALR, Unisys, DG, Fujitsu и др.).

Масштабируемость

- Наличие общей памяти сильно упрощает взаимодействие процессоров между собой, однако накладывает сильные ограничения на их число - не более 32 в реальных системах. Для построения масштабируемых систем на базе SMP используются кластерныеНаличие общей памяти сильно упрощает взаимодействие процессоров между собой, однако накладывает сильные ограничения на их число - не более 32 в реальных системах. Для построения масштабируемых систем на базе SMP используются кластерные или NUMA-архитектуры.

Операционная система

- Вся система работает под управлением единой ОС (обычно UNIX-подобной, но для Intel-платформ поддерживается Windows NT). ОС автоматически (в процессе работы) распределяет процессы/нити по процессорам (scheduling), но иногда возможна и явная привязка

Модель программирования

- Программирование в модели общей памяти. (POSIX threads, OpenMPПрограммирование в модели общей памяти. (POSIX threads, OpenMP). Для SMP-систем существуют сравнительно эффективные средства автоматического распараллеливания.

Основные классы современных параллельных компьютеров

Системы с неоднородным доступом к памяти (NUMA)

Архитектура

- Система состоит из однородных базовых модулей (плат), состоящих из небольшого числа процессоров и блока памяти. Модули объединены с помощью высокоскоростного коммутатора. Поддерживается единое адресное пространство, аппаратно поддерживается доступ к удаленной памяти, т.е. к памяти других модулей. При этом доступ к локальной памяти в несколько раз быстрее, чем к удаленной.

  В случае, если аппаратно поддерживается когерентность кэшей во всей системе (обычно это так), говорят об архитектуре c-NUMA (cache-coherent NUMA)

Примеры

- HP HP 9000 V-classHP HP 9000 V-class в SCA-конфигурациях, SGI Origin2000HP HP 9000 V-class в SCA-конфигурациях, SGI Origin2000, Sun HPC 10000HP HP 9000 V-class в SCA-конфигурациях, SGI Origin2000, Sun HPC 10000, IBM/Sequent NUMA-Q 2000HP HP 9000 V-class в SCA-конфигурациях, SGI Origin2000, Sun HPC 10000, IBM/Sequent NUMA-Q 2000, SNI RM600.

Масштабируемость

- Масштабируемость NUMA-систем ограничивается объемом адресного пространства, возможностями аппаратуры поддержки когерентности кэшей и возможностями операционной системы по управлению большим числом процессоров. На настоящий момент, максимальное число процессоров в NUMA-системах составляет 256 (Origin2000).

Операционная система

- Обычно вся система работает под управлением единой ОС, как в SMP. Но возможны также варианты динамического "подразделения" системы, когда отдельные "разделы" системы работают под управлением разных ОС (например, Windows NT и UNIX в NUMA-Q 2000).

Модель программирования

- Аналогично SMP.

Основные классы современных параллельных компьютеров

Параллельные векторные системы (PVP)

Архитектура

- Основным признаком PVP-систем является наличие специальных векторно-конвейерных процессоров, в которых предусмотрены команды однотипной обработки векторов независимых данных, эффективно выполняющиеся на конвейерных функциональных устройствах.

Примеры

- NEC SX-4/SX-5NEC SX-4/SX-5, линия векторно-конвейерных компьютеров CRAY: от CRAY-1, CRAY J90/T90NEC SX-4/SX-5, линия векторно-конвейерных компьютеров CRAY: от CRAY-1, CRAY J90/T90, CRAY SV1NEC SX-4/SX-5, линия векторно-конвейерных компьютеров CRAY: от CRAY-1, CRAY J90/T90, CRAY SV1, серия Fujitsu VPP.

Масштабируемость

Как правило, несколько таких процессоров (1-16) работают одновременно над общей памятью (аналогично SMP Как правило, несколько таких процессоров (1-16) работают одновременно над общей памятью (аналогично SMP) в рамках многопроцессорных конфигураций. Несколько таких узлов могут быть объединены с помощью коммутатора (аналогично MPP).

Модель программирования

- Эффективное программирование подразумевает векторизацию циклов (для достижения разумной производительности одного процессора) и их распараллеливание (для одновременной загрузки нескольких процессоров одним приложением).

Основные классы современных параллельных компьютеров

Кластерные системы

Архитектура

- Набор рабочих станций (или даже ПК) общего назначения, используется в качестве дешевого варианта массивно-параллельного компьютера. Для связи узлов используется одна из стандартных сетевых технологий (Fast/Gigabit Ethernet, Myrinet) на базе шинной архитектуры или коммутатора. При объединении в кластер компьютеров разной мощности или разной архитектуры, говорят о гетерогенных (неоднородных) кластерах.

Примеры

- NT-кластерNT-кластер в NCSA, Beowulf-кластеры.

Масштабируемость

- Узлы кластера могут одновременно использоваться в качестве пользовательских рабочих станций. В случае, когда это не нужно, узлы могут быть существенно облегчены и/или установлены в стойку.

Операционная система

- Используются стандартные для рабочих станций ОС, чаще всего, свободно распространяемые - Linux/FreeBSD, вместе со специальными средствами поддержки параллельного программирования и распределения нагрузки.

Модель программирования

- Программирование, как правило, в рамках модели передачи сообщений (чаще всего - MPI). Дешевизна подобных систем оборачивается большими накладными расходами на взаимодействие параллельных процессов между собой, что сильно сужает потенциальный класс решаемых задач. Используются стандартные для рабочих станций ОС, чаще всего, свободно распространяемые - Linux/FreeBSD, вместе со специальными средствами поддержки параллельного программирования и распределения нагрузки.

Ссылки на литературу

Анализ мультипроцессорных систем с иерархической памятью
- http://masters.donntu.edu.ua/2001/fvti/prokopenko/diss/index.htm

Языки параллельной обработки
- http://ibd.tsi.lv/cgi/sart2.pl?T1=ZAG

Архитектура и топология многопроцессорных вычислительных систем
- http://informika.ru/text/teach/topolog/8.htm

Эволюция языков программировния
- http://iais.kemsu.ru/odocs/progs/lang.html

Специализирванные параллельные языки и расширения существующих языков
- http://www.parallel.ru/tech/tech_dev/par_lang.html
- Основные классы современных параллельных компьютеров
- http://www.parallel.ru/computers/classes.html
- Управление процессорами
- http://www.osu.cctpu.edu.ru/lectors/325/oper_system/tema12.htm
- Мультипроцессорные системы
- http://afc.deepweb.ru/texts/daitel/glava11.php
- Архитектура и топология многопроцессорных вычислительных систем
- http://informika.ru/text/teach/topolog/index.htm
- Системы параллельной обработки данных
- http://www.referatfrom.ru/ref/0/0/37346.html

Матричные вычислительные системы

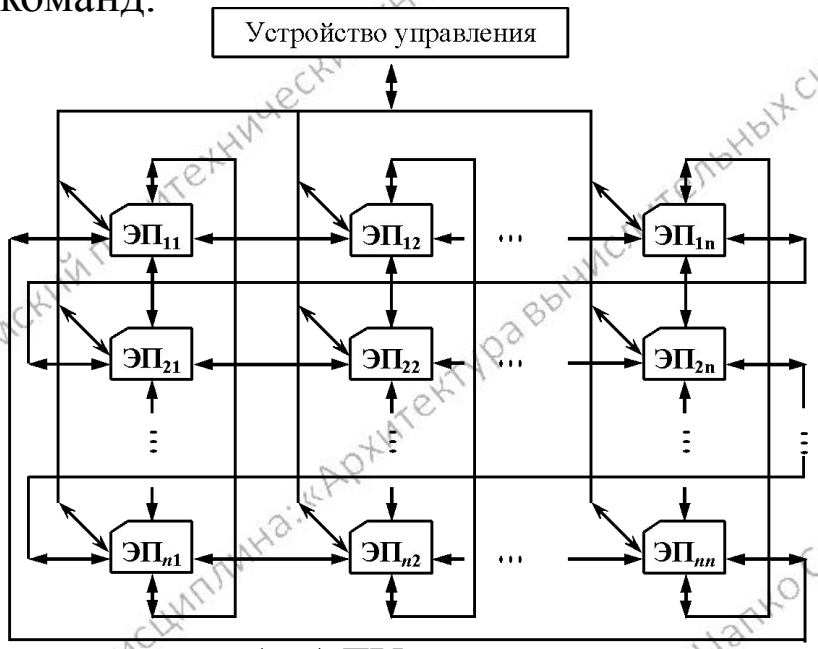
Матричные ВС обладают более широкими архитектурными возможностями, чем конвейерные ВС: их каноническая архитектура относится к классу SIMD.

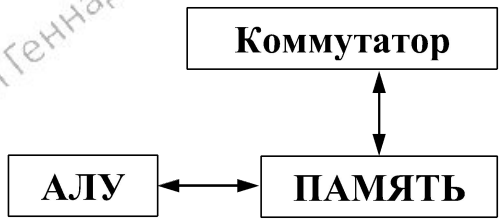
Матричные ВС относятся к классу систем с массовым параллелизмом (Massively Parallel Processing Systems) и, следовательно, не имеют принципиальных ограничений в наращивании своей производительности.

Матричные ВС предназначены для решения сложных задач, связанных с выполнением операций над векторами, матрицами и массивами данных (Data Arrays).

Функциональная структура матричного процессора

Матричный, или векторный *процессор* (Array Processor) представляет собой «матрицу» связанных идентичных элементарных процессоров (ЭП), управляемых одним потоком команд.





Каждый ЭП включает в себя АЛУ, память и локальный коммутатор. Сеть связей между ЭП (точнее, локальными коммутаторами) позволяет осуществлять обмен данными между любыми процессорами. Поток команд поступает на матрицу ЭП от единого устройства управления (SIMD-архитектура, в каноническом виде).

Функциональная структура матричного процессора

При решении сложных задач фактически один и тот же алгоритм параллельно (одновременно) реализуется над многими частями исходного массива данных.

Матричные процессоры ориентированы на работу в монопрограммном режиме (когда решается только одна задача, представленная в параллельной форме) .

Реализация мультипрограммных режимов в матричном процессоре осуществляется за счет разделения и «времени», и «пространства». В матричном процессоре имеется единственное устройство управления и множество ЭП, следовательно, в мультипрограммной ситуации должны «делиться» время устройства управления и элементарные процессоры («пространство») между программами.

Первый матричный компьютер

Первая матричная ВС SOLOMON (Simultaneous Operation Linked Ordinal MOdular Network — вычислительная сеть синхронно функционирующих упорядоченных модулей) была разработана в Иллинойском университете (University of Illinois) США под руководством Даниэля Слотника.

Работы по проекту SOLOMON велись с 1962 г., однако этот проект промышленного воплощения не нашел; в 1963 г. был создан лишь макет ВС размером 3x3 элементарных процессора. Позднее была построена конфигурация ВС размером 10x10 ЭП в фирме Westinghouse Electric Corp.

Время показало, что технология пока не достигла возможностей создания матричных ЭВМ.

Вычислительная система ILLIAC IV

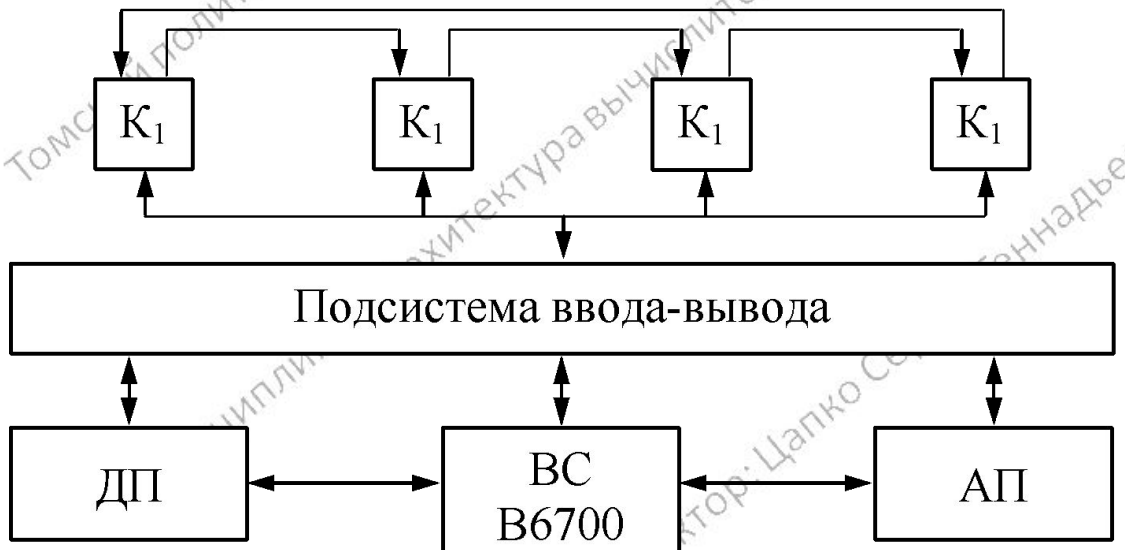
Матричная ВС ILLIAC IV создана Иллинойским университетом и корпорацией Бэрроуз (Burroughs Corporation). Работы по созданию ILLIAC IV были начаты в 1966 г. под руководством Д.Л. Слотника. Монтаж системы был закончен в мае 1972 г. в лаборатории фирмы Burroughs (Паоли, штат Панама), а установка для эксплуатации осуществлена в октябре 1972 г. в Научно-исследовательском центре НАСА им. Эймса в штате Калифорния (NASA's Ames Research Center; NASA - National Aeronautics and Space Administration - Национальное управление аэронавтики и космоса).

Количество процессоров в системе - 64;
быстродействие - $2 \cdot 10^8$ опер./с (Core 2 Duo до $3{,}1 \cdot 10^{11}$ опер./с);
емкость оперативной памяти - 1 Мбайт;
полезное время составляло 80...85 % общего времени работы ILLIAC IV,
стоимость 40 000 000 долл.,
вес - 75 т,
занимаемая площадь - 930 $м^2$.

Система ILLIAC IV была включена в вычислительную сеть ARPA (Advanced Research Projects Agency - Управление перспективных исследований и разработок Министерства обороны США) и успешно эксплуатировалась до 1981 г.

Функциональная структура системы ILLIAC IV

Матричная ВС ILLIAC IV (рис. 5.2) должна была состоять из четырех квадрантов ($К_1$-$К_4$) подсистемы ввода-вывода информации, ведущей ВС В6700 (или В6500), дисковой памяти (ДП) и архивной памяти (АП). Планировалось, что ВС обеспечит быстродействие 10^9 опер./с.



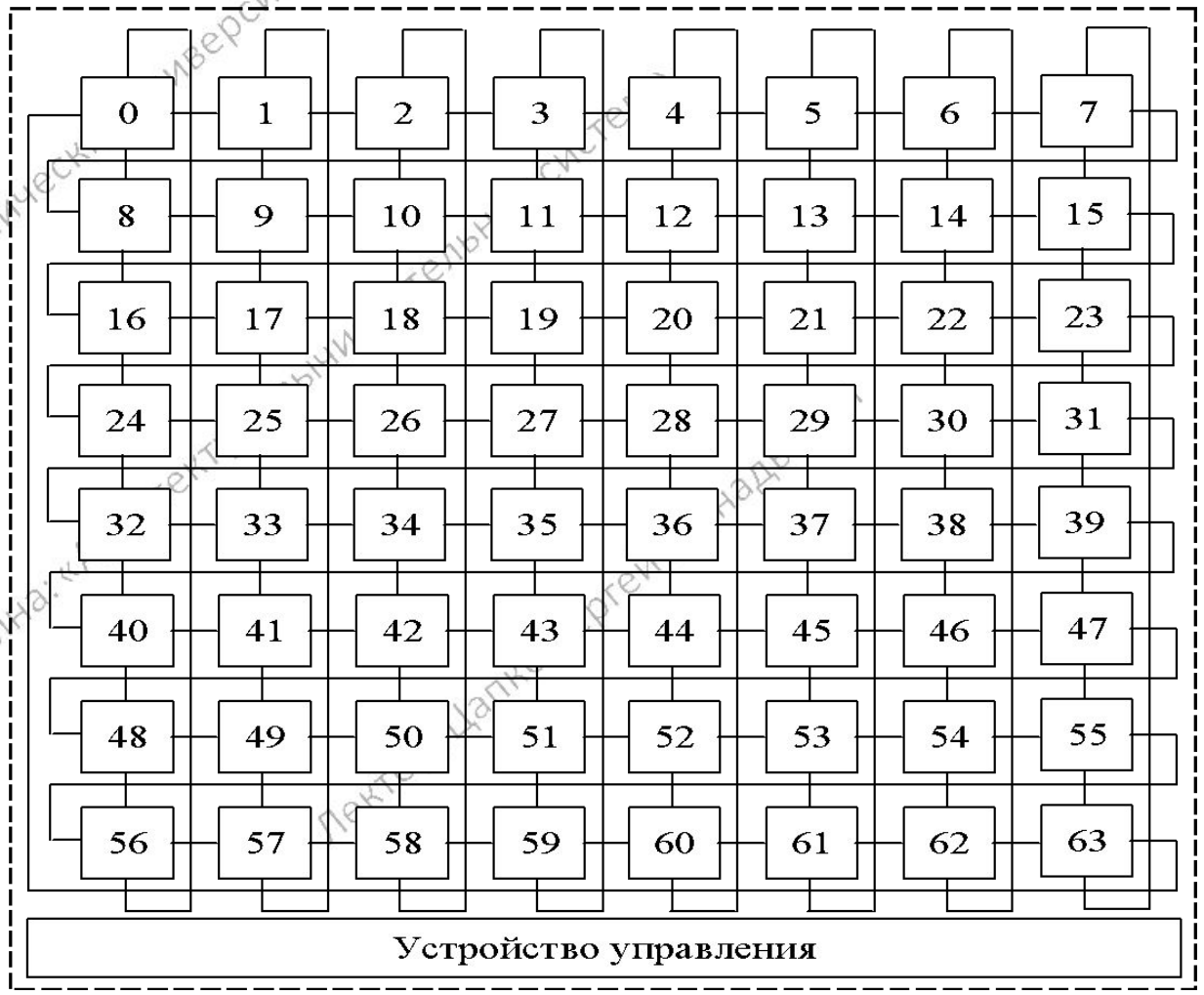
В реализованном варианте ILLIAC IV содержался только один квадрант, что обеспечило быстродействие $2 \cdot 10^8$ опер./с. При этом ILLIAC IV оставалась самой быстродействующей ВС вплоть до 80-х годов XX в.

Функциональная структура системы ILLIAC IV

Квадрант — матричный процессор, включавший в себя устройство управления и 64 ЭП. Устройство управления представляло собой специализированную ЭВМ, которая использовалась для выполнения операций над скалярами и формировала поток команд на матрицу ЭП.

Элементарные процессоры матрицы регулярным образом были связаны друг с другом. Структура квадранта системы ILLIAC IV представлялась двумерной решеткой, в которой граничные ЭП были связаны по канонической схеме.

Все 64 ЭП работали синхронно и единообразно. Допускалось одновременное выполнение скалярных и векторных операций.



Формат представления данных системы ILLIAC IV

В системе ILLIAC IV использовалось слово длиной 64 двоичных разряда. Числа могли представляться в следующих форматах:

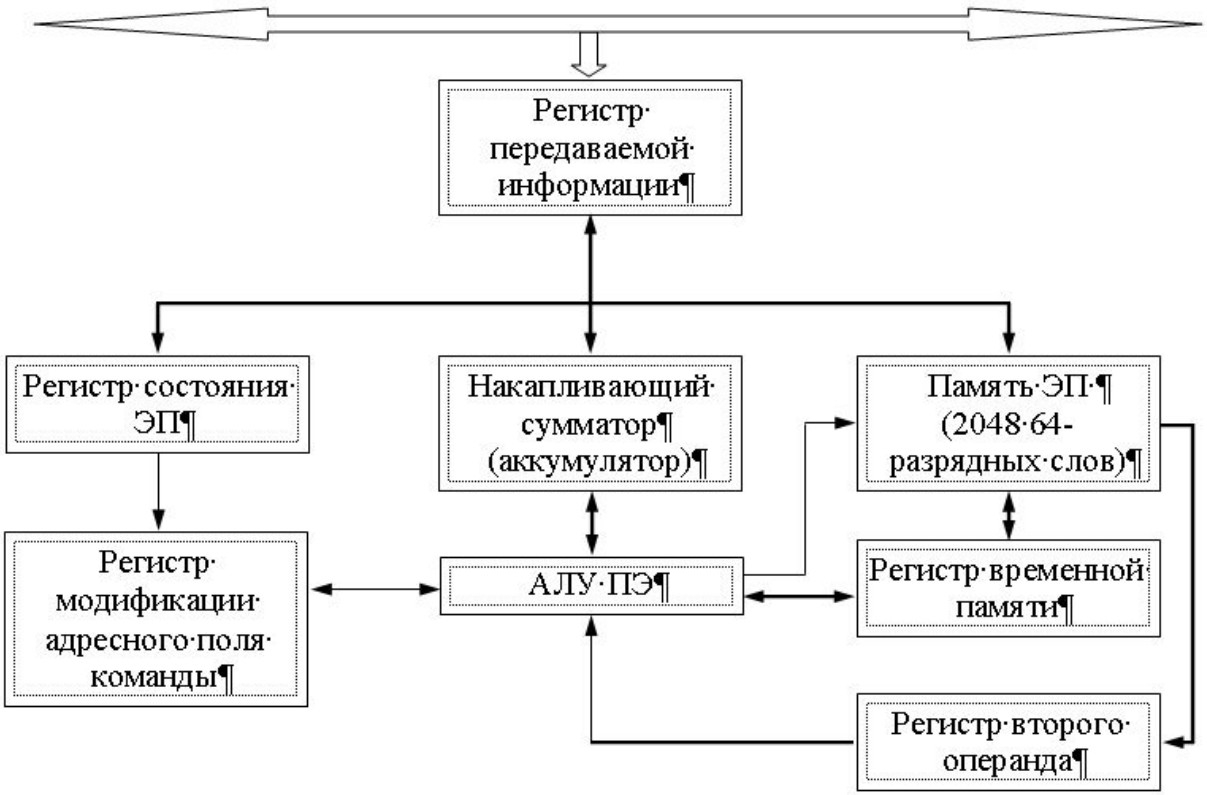
64 или 32 разряда с плавающей запятой;
48, или 24, или 8 разрядов с фиксированной запятой.

При использовании 64-, 32- и 8-разрядных форматов матрица из 64 ЭП могла обрабатывать векторы операндов, состоявшие из 64, 128 и 512 компонентов соответственно.

Система ILLIAC IV при суммировании 512 8-разрядных чисел имела быстродействие почти 10^{10} опер./с, а при сложении 64-разрядных чисел с плавающей запятой - $1,5 \cdot 10^{8}$ опер./с.

Компонентная структура системы ILLIAC IV

Элементарный процессор мог находиться в одном из двух состояний - активном или пассивном. В первом состоянии ему разрешалось, а во втором запрещалось выполнять команды, поступавшие из устройства управления.



Разрядность сумматора и всех регистров ЭП – 64 бита, регистр модификации адреса – 16 бит и состояния – 8 бит

Аппаратный состав системы ILLIAC IV

Подсистема ввода-вывода состояла из устройства управления, буферного запоминающего устройства и коммутатора.

Ведущая ВС B 6700 — мультипроцессорная система корпорации Burroughs; могла иметь в своем составе от 1 до 3 центральных процессоров и от 1 до 3 процессоров ввода-вывода информации и обладала быстродействием (1...3) • 10^6 опер./с.

Дисковая память (ДП) состояла из двух дисков и обрамляющих электронных схем, она имела емкость порядка 10^9 бит и была снабжена двумя каналами со скоростью 0,5 • 10^9 бит/с. Среднее время обращения к диску составляло 20 мс.

Архивная память (АП) — постоянная лазерная память с однократной записью, разработанная фирмой Precision Instrument Company емкостью 10^{12} бит. Имелось 400 информационных полосок по 2,9 млрд. бит, которые размещались на вращающемся барабане. Время поиска данных на любой из 400 полосок достигало 5 с; время поиска в пределах полоски – 200 нс. Существовало два канала обращения к архивной памяти, скорость считывания и записи данных по каждому из которых была равна 4 • 10^6 бит/с.

Программное обеспечение системы ILLIAC IV

Цель разработки ILLIAC IV — создание мощной ВС для решения задач с большим числом операций.

Программа структурно содержала три части:

- «Предпроцессорная» часть обеспечивала инициирование задачи и десятично-двоичные преобразования (последовательная форма)
- «Ядро» осуществляло решение поставленной задачи и представлялось в параллельной форме. Размер ядра составлял 5... 10 % полного объема программы (его исполнение на последовательной машине требовало 80...95 % рабочего времени)
- «Постпроцессорная» часть осуществляла запись результатов в архивные файлы, двоично-десятичные преобразования, вычерчивание графиков, вывод результатов на печать и т. п. (последовательная форма)

Программное обеспечение системы ILLIAC IV

Операционная система ILLIAC IV состояла из набора асинхронных программ, выполнявшихся под управлением главной управляющей программы B6700. Она работала в двух режимах: в первом режиме выполнялся контроль и диагностика неисправностей в квадранте и в подсистеме вводавывода информации; во втором – осуществлялось управление работой ILLIAC IV при поступлении на B6700 заданий от пользователей.

Задание для ILLIAC IV состояло из:

- Программы B6700, написанной, как правило, на языках ALGOL или FORTRAN и осуществлявшей подготовку (и преобразование) входных двоичных файлов («предпроцессорная» часть).
- Программы ILLIAC IV, обычно написанных на языках Glynpir или FORTRAN, которые использовались ILLIAC IV (составляли «ядро») для обработки файлов, подготовленных программами B6700, а также для формирования двоичных выходных файлов.
- Программы B6700 (на версиях языков ALGOL или FORTRAN), которые преобразовывали двоичные файлы ILLIAC IV в требуемый выходной формат («постпроцессорная» часть).
- Программа на управляющем языке Illiac, определявшая задание. Эта программа ориентировала операционную систему на работу, предусмотренную заданием.

Средства программирования системы ILLIAC IV

Средства программирования ILLIAC IV включали язык ассемблера (Assembler Language) и три языка высокого уровня: Tranquil, Glynpir, FORTRAN.

Язык ассемблера ILLIAC IV – традиционный язык программирования, адаптированный под архитектуру ВС. В частности, он имел сложные макроопределения, которые можно было применять для включения стандартных операций ввода-вывода и других операций связи между программами B6700 и ILLIAC IV.

Языки высокого уровня благодаря архитектурным особенностям ILLIAC IV отличались от соответствующих языков ЭВМ.

1. Распределение двумерной памяти. Была разрешена адресация отдельных слов в памяти ЭП и строк (из 64 слов) в пределах запоминающих устройств матрицы ЭП. Адресация по «столбцу» группы слов в памяти одного ЭП была недопустима.
2. Параллелизм и управление режимом обработки. Параллелизм ВС предопределяет работу с векторами данных. Следовательно, языки ILLIAC IV должны были допускать операции над векторами данных или строками матриц. Размерность вектора и количество подлежащих обработке элементов вектора определялись словами режима. Языки ILLIAC IV обеспечивали эффективную реализацию широкого круга вычислений и обработку слов режима.
3. Вид команд пересылок и индексации. Каждый из языков ILLIAC IV должен был содержать команды пересылок и индексации с различными приращениями в каждом элементарном процессоре.

Языки высокого уровня системы ILLIAC IV

Tranquil подобен языку ALGOL и полностью не зависел от архитектуры ILLIAC IV. Он был разработан для обеспечения параллельной обработки массивов информации.

Компилятор языка Tranquil потребовал больших системных затрат, связанных с маскированием архитектуры ILLIAC IV. Поэтому работы по созданию компилятора были приостановлены и предприняты шаги по модификации языка Tranquil.

Glynpir являлся языком также алгольного типа с блочной структурой. Он позволял опытному программисту использовать значительные возможности архитектуры ВС ILLIAC IV.

Язык FORTRAN был разработан для рядовых пользователей ILLIAC IV. Он в отличие от Glynpir освобождал пользователя от детального распределения памяти и предоставлял ему возможность мыслить в терминах строк любой длины. Он позволял путем применения модифицированных операторов ввода-вывода воспользоваться параллельной программой как последовательной и выполнить ее на одном ЭП. FORTRAN давал возможность отлаживать параллельные программы на одном элементарном процессоре.

Применение системы ILLIAC IV

Практически установлено, что ILLIAC IV была эффективна при решении широкого спектра сложных задач.

Классы задач: матричная арифметика, системы линейных алгебраических уравнений, линейное программирование, исчисление конечных разностей в одномерных, двумерных и трехмерных случаях, квадратуры (включая быстрое преобразование Фурье), обработка сигналов.

Классы задачи, обеспечивающих не полное использование ЭП: : движение частиц (метод Монте-Карло и т. д.), несимметричные задачи на собствен ные значения, нелинейные уравнения, отыскание корней полиномов.

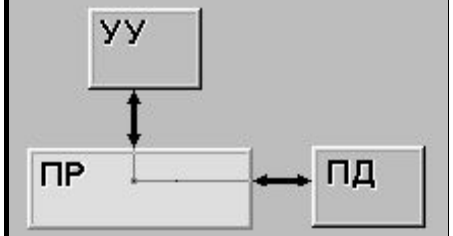
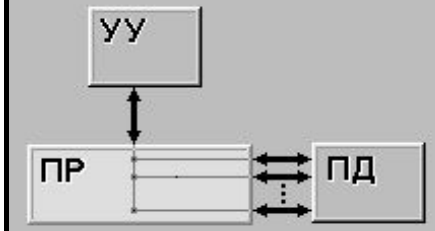
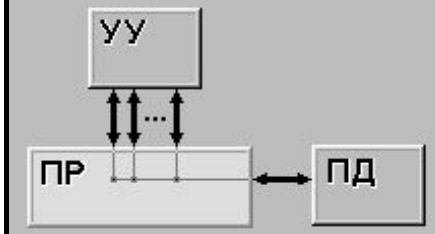
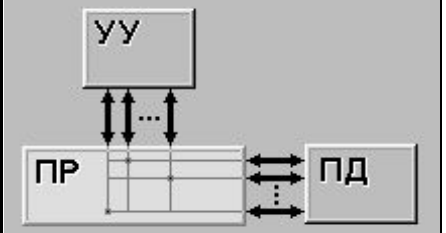
Эффективность системы ILLIAC IV на примере решения типичной задачи линейного программирования, имеющей 4000 ограничений и 10 000 переменных, можно оценить по времени решения, которое составляло менее 2 мин, а для большой ЭВМ третьего поколения – 6...8 часов

Классификация вычислительных систем

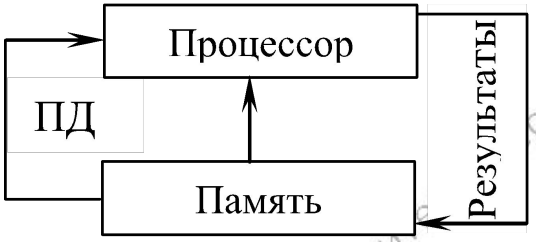
- Классификация Флинна
- Классификация Хокни
- Классификация Фенга
- Классификация Дункана
- Классификация Хендлера
- Классификация Шнайдера
- Классификация Скилликорна

Классификация Флина

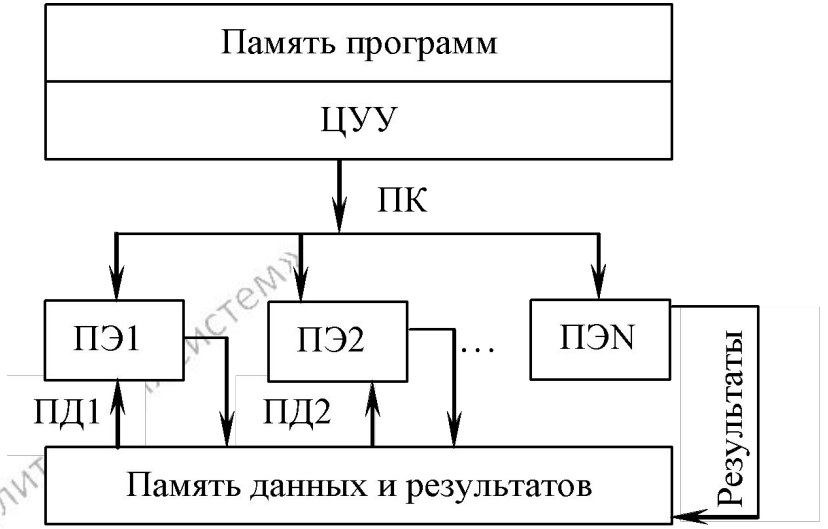
Базируется на понятии потока, под которым понимается последовательность элементов, команд или данных, обрабатываемая процессором.

	**SISD** (single instruction stream / single data stream) - одиночный поток команд и одиночный поток данных.
	**SIMD** (single instruction stream / multiple data stream) - одиночный поток команд и множественный поток данных.
	**MISD** (multiple instruction stream / single data stream) - множественный поток команд и одиночный поток данных.
	**MIMD** (multiple instruction stream / multiple data stream) - множественный поток команд и множественный поток данных.

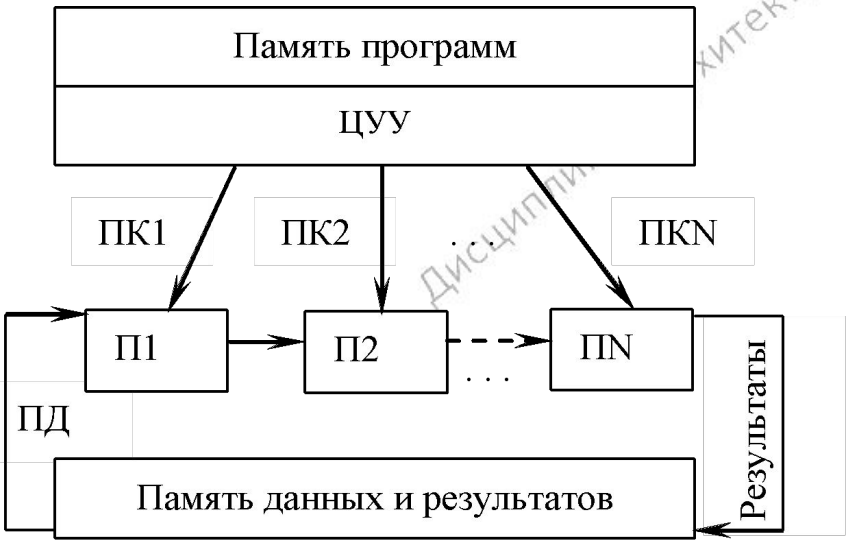
Архитектуры ЭВМ



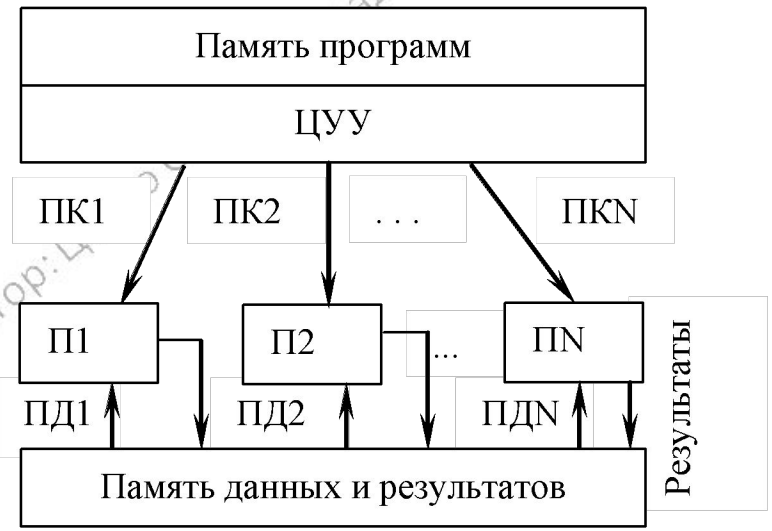
SISD- архитектура



SIMD- архитектура



MISD-архитектура



MIMD-архитектура

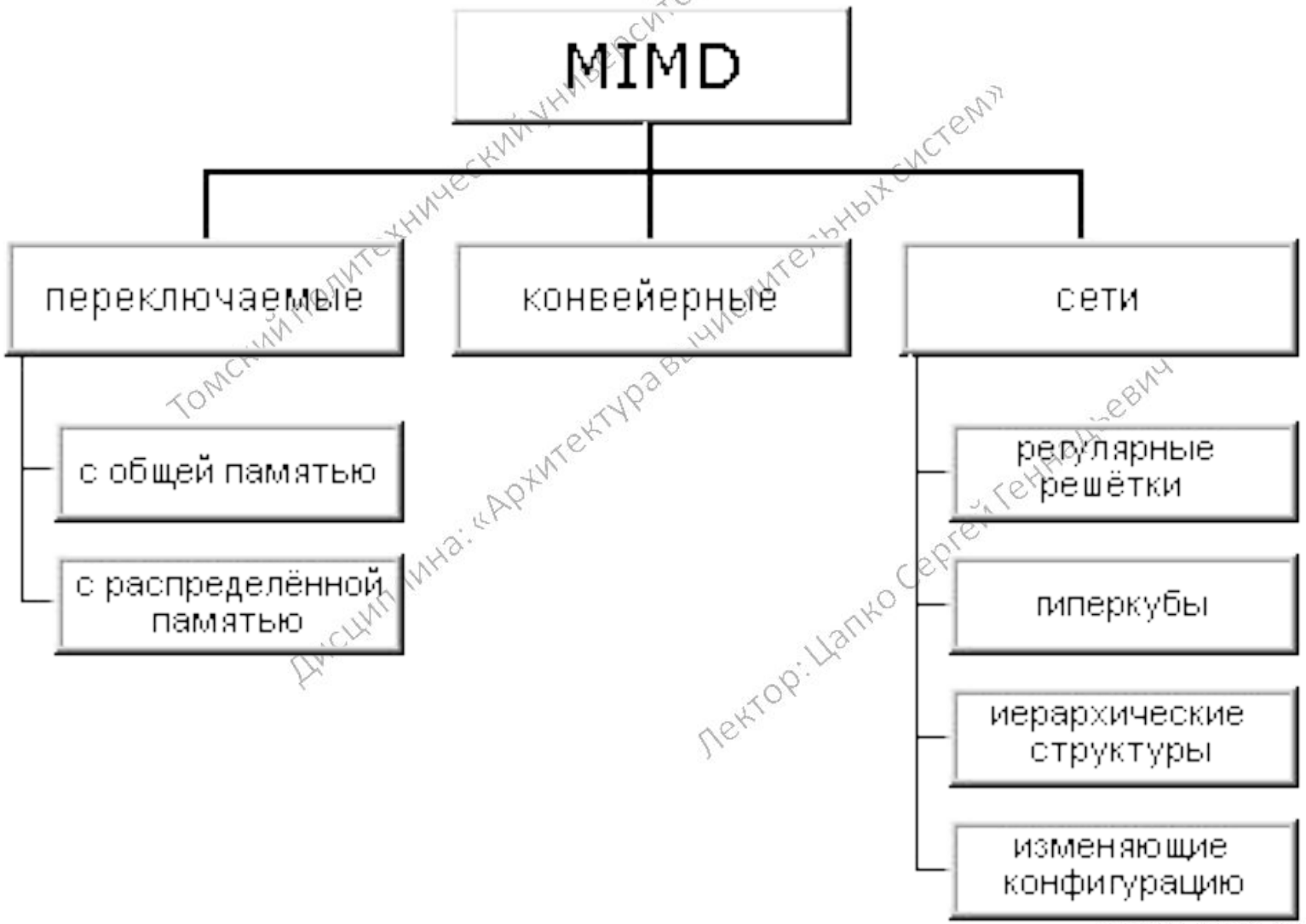
Недостатки классификации Флина

1. Некоторые архитектуры четко не вписываются в данную классификацию
2. Чрезмерная заполненность класса MIMD

Классификация Хокни

Основная идея классификации состоит в следующем. Множественный поток команд может быть обработан двумя способами: либо одним конвейерным устройством обработки, работающем в режиме разделения времени для отдельных потоков, либо каждый поток обрабатывается своим собственным устройством.

Классификация Хокни



Примеры классификации Флина

- SISD – PDP-11, VAX 11/780, CDC 6600 и CDC 7600
- SIMD – ILLIAC IV, CRAY-1
- MISD – нет
- MIMD – большинство современных машин

Примеры классификации Хокни

MIMD конвейерные – Denelcor HEP

MIMD переключаемые с распределенной памятью – PASM, PRINGLE

MIMD переключаемы с общей памятью – CRAY X-MP, BBN Butterfly

MIMD звездообразная сеть – ICAP

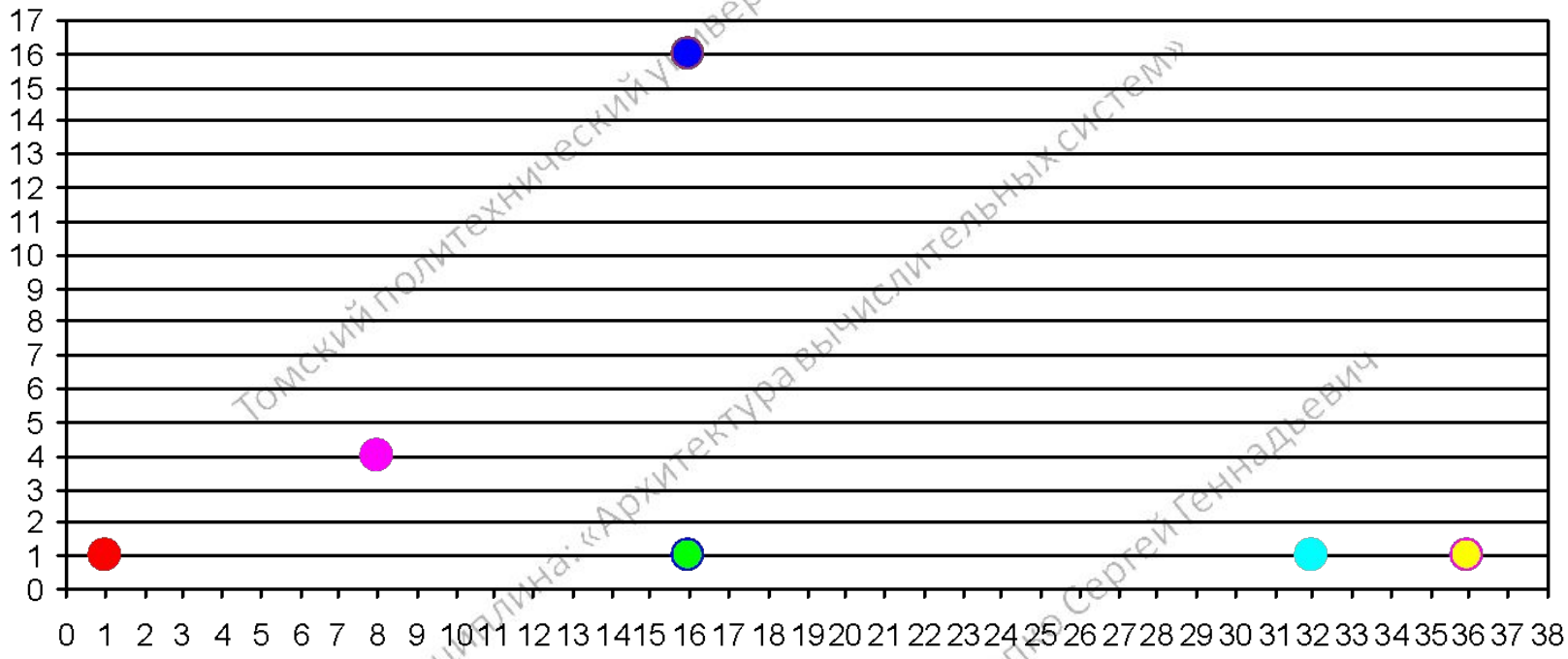
MIMD регулярные решетки – Intel Paragon, CRAY T3D

MIMD гиперкубы – NCube, InteliPSC

Классификация Фенга

Идея классификации вычислительных систем на основе двух простых характеристик. Первая - число бит n в машинном слове, обрабатываемых параллельно при выполнении машинных инструкций. Вторая характеристика равна числу слов m, обрабатываемых одновременно данной вычислительной системой. Вторую характеристику обычно называют шириной битового слоя.

Классификация Фенга



Любую вычислительную систему С можно описать парой чисел (n, m) и представить точкой на плоскости в системе координат длина слова - ширина битового слоя. Площадь прямоугольника со сторонами n и m определяет интегральную характеристику потенциала параллельности В архитектуры и носит название

Примеры классификации Фенга

- Разрядно-последовательные пословно-последовательные (n=m=1):
  MINIMA с естественным описанием (1,1)
- Разрядно-параллельные пословно-последовательные (n>1; m=1):
  IBM 701 с описанием (36,1), PDP-11 (16,1), IBM 360/50,
  VAX 11/780 - обе с описанием (32,1)
- Разрядно-последовательные пословно-параллельные (n=1; m>1):
  STARAN (1, 256) и MPP (1,16384) фирмы Goodyear Aerospace,
  прототип системы ILLIAC IV компьютер SOLOMON (1, 1024),
  ICL DAP (1, 4096).
- Разрядно-параллельные пословно-параллельные (n>1; m>1):
  ILLIAC IV (64, 64), TI ASC (64, 32), C.mmp (16, 16), CDC 6600 (60, 10),
  BBN Butterfly GP1000 (32, 256).

Недостаток

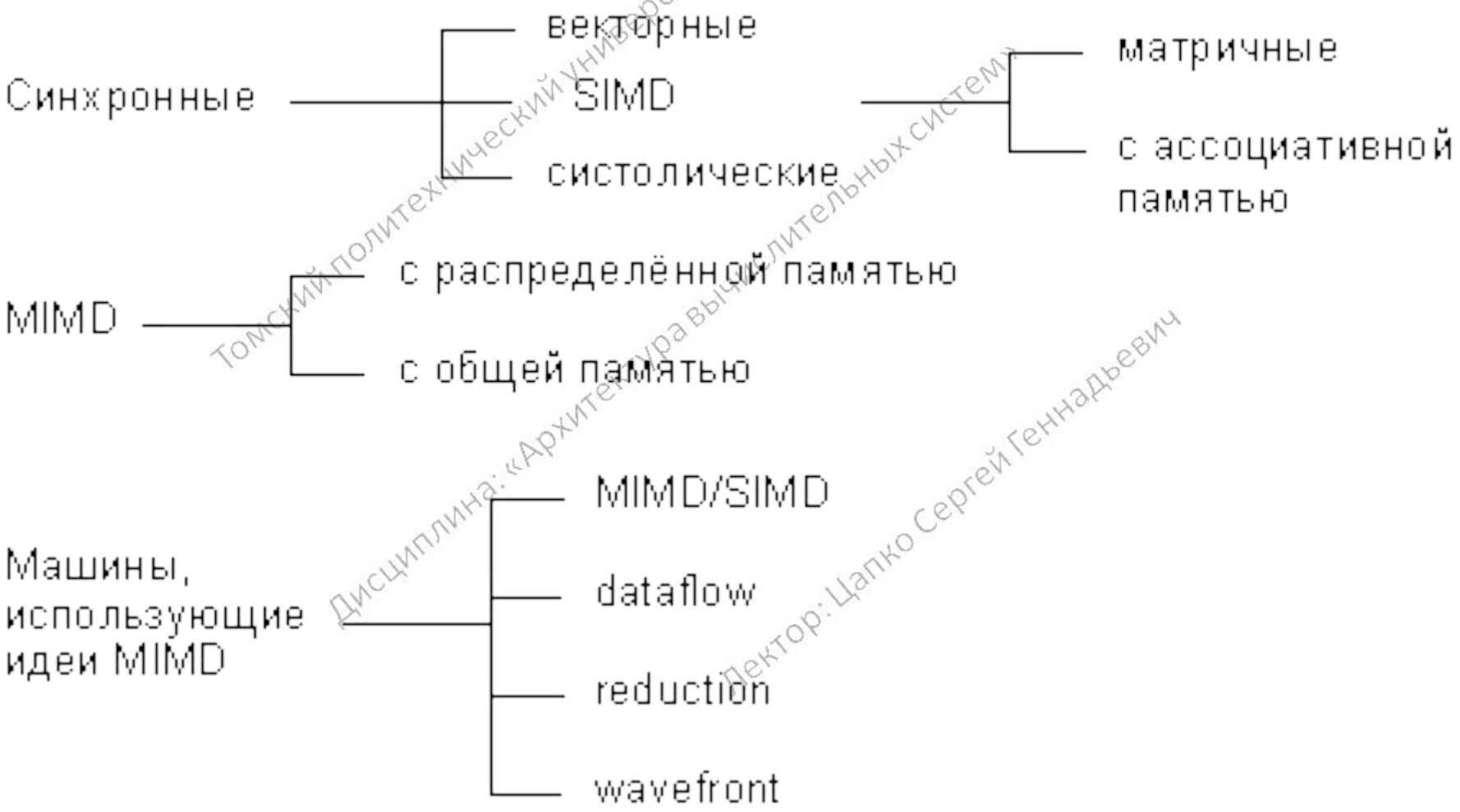
- не делает никакого различия между процессорными матрицами, векторно-конвейерными и многопроцессорными системами;
- отсутствует акцент на том, за счет чего компьютер может одновременно обрабатывать более одного слова.

Классификация Дункана

Дункан определил набор требований для создания своей классификации.

- Из классификации должны быть исключены машины, параллелизм в которых заложен на самом низком уровне:
  - конвейеризация на этапе подготовки и выполнения команды;
  - наличие в архитектуре нескольких функциональных устройств, работающих независимо;
  - наличие отдельных процессоров ввода/вывода
- Классификация должна быть согласованной с классификацией Флинна.
- Классификация должна описывать архитектуры, которые однозначно не укладываются в систематику Флинна

Классификация Дункана



Основные архитектуры, представленные на рисунке рисунка

- Систолические архитектуры представляют собой множество процессоров, объединенных регулярным образом. Обращение к памяти может осуществляться только через определенные процессоры на границе массива. Выборка операндов из памяти и передача данных по массиву осуществляется в одном и том же темпе. Направление передачи данных между процессорами фиксировано. Каждый процессор за интервал времени выполняет небольшую инвариантную последовательность действий.

  Гибридные MIMD/SIMD архитектуры, dataflow, reduction и wavefront вычислительные системы осуществляют параллельную обработку информации на основе асинхронного управления.
- MIMD/SIMD - типично гибридная архитектура. Она предполагает, что в MIMD системе можно выделить группу процессоров, представляющую собой подсистему, работающую в режиме SIMD;
- Dataflow используют модель, в которой команда может выполнятся сразу же, как только вычислены необходимые операнды;
- Модель вычислений, применяемая в reduction машинах иная и состоит в следующем: команда становится доступной для выполнения тогда и только тогда, когда результат ее работы требуется другой, доступной для выполнения, команде в качестве операнда;
- Wavefront array архитектура. В данной архитектуре процессоры объединяются в модули и фиксируются связи, по которым процессоры могут взаимодействовать друг с другом. Данная архитектура использует асинхронный механизм связи с подтверждением

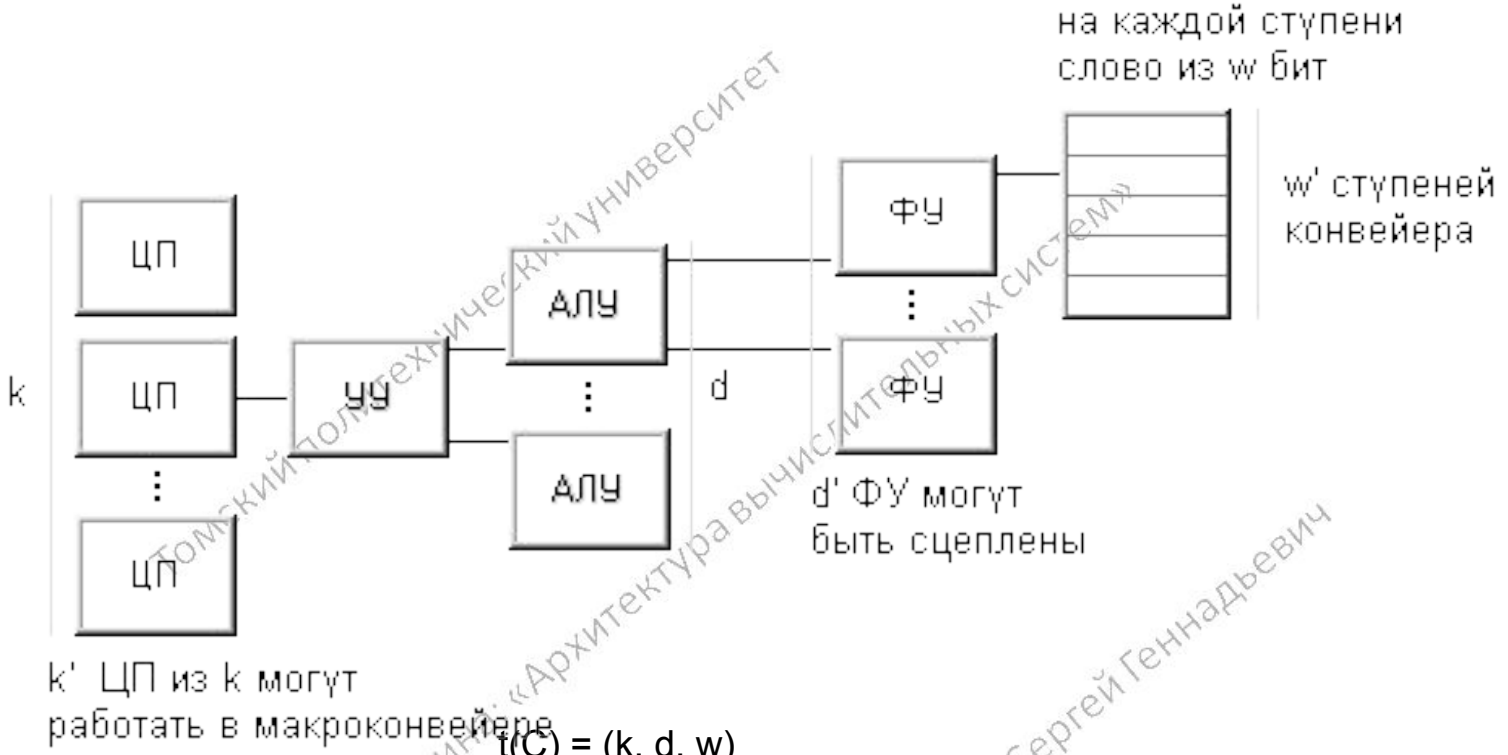
Классификация Хендлера

В основу классификации В.Хендлер закладывает явное описание возможностей параллельной и конвейерной обработки информации вычислительной системой. При этом он намеренно не рассматривает различные способы связи между процессорами и блоками памяти и считает, что коммуникационная сеть может быть нужным образом сконфигурирована и будет способна выдержать предполагаемую нагрузку.

Предложенная классификация базируется на различии между тремя уровнями обработки данных в процессе выполнения программ:

- уровень выполнения программы
- уровень выполнения команд
- уровень битовой обработки

Классификация Хендлера



t(C) = (k, d, w)

t(PEPE) = (k×k',d×d',w×w')

где:

k - число процессоров (каждый со своим УУ), работающих параллельно

k' - глубина макроконвейера из отдельных процессоров

d - число АЛУ в каждом процессоре, работающих параллельно

d' - число функциональных устройств АЛУ в цепочке

w - число разрядов в слове, обрабатываемых в АЛУ параллельно

w' - число ступеней в конвейере функциональных устройств АЛУ

$$t(n,d,w)=[(1,d,w)+...+(1,d,w)]\{n\ раз\}$$

Дополнения к классификации Хендлера

Для описания:

✔сложных структур с подсистемами ввода-вывода

✔возможных режимов функционирования вычислительных систем, поддерживаемых для оптимального соответствия структуре программ.

Хендлер предлагает использовать три операции:

- Первая операция (×) отражает конвейерный принцип обработки и предполагает последовательное прохождение данных сначала через первый ее аргумент-подсистему, а затем через второй
- *Вторая операция параллельного исполнения* (+), фиксирует возможность независимого использования процессоров разными задачами
- Третья операция - *операция альтернативы* (V), показывает возможные альтернативные режимы функционирования вычислительной системы

Примеры классификации Хендлера

- t(MINIMA) = (1,1,1);
  t(IBM 701) = (1,1,36);
  t(SOLOMON) = (1,1024,1);
  t(ILLIAC IV) = (1,64,64);
  t(STARAN) = (1,8192,1) - в полной конфигурации;
  t(C.mmp) = (16,1,16) - основной режим работы;
  t(PRIME) = (5,1,16);
  t(BBN Butterfly GP1000) = (256,~1,~32).
- t(TI ASC) = (1,4,64×8)
- t(CDC 6600) = (1,1×10,~64)
- t(PEPE) = (1×3,288,32)
- t(CDC 6600) = (10,1,12) × (1,1×10,64),
- t(PEPE) = t(CDC 7600) × (1×3, 288, 32) = = (15, 1, 12) × (1, 1×9, 60) × (1×3, 288, 32)
- (15, 1, 12) × (1, 1×9, 60) = [(1, 1, 12) + ... +(1, 1, 12)]} *{15 раз}* × (1, 1×9, 60)
- t(C.mmp) = (16, 1, 16) V (1×16, 1,1 6) V (1, 16, 16).

Классификация Шнайдера

Основная идея заключается в выделении этапов выборки и непосредственно исполнения в потоках команд и данных. Именно разделение потоков на адреса и их содержимое позволяет описать такие ранее "неудобные" для классификации архитектуры, как компьютеры с длинным командным словом, систолические массивы и целый ряд других архитектур.

Под *потоком ссылок* (reference stream) *S* некоторой вычислительной системы понимается конечное множество бесконечных последовательностей пар:

$$S = \{ (a_1 < t_1 >) (a_2 < t_2 >)...,$$
$$(b_1 < u_1 >) (b_2 < u_2 >)...,$$
$$(c_1 < v_1 >)(c_2 < v_2 >)...\},$$

Первый компонент каждой пары - это неотрицательное целое число, называемое *адресом*, второй компонент - это набор из n неотрицательных целых чисел, называемых *значениями*, причем n одинаково для всех наборов всех последовательностей. Например, пара $(b_2 < u_2 >)$ определяет адрес b_2 и значение $< u_2 >$. Если значения рассматривать как команды, то из потока ссылок получается *поток команд I*; если же значения интерпретировать как данные, то соответствующий поток - это *поток данных D*.

Классификация Шнайдера

Пусть *S* произвольный поток ссылок. *Последовательность адресов* потока *S*, обозначаемая S_a, - это последовательность, чей i-й элемент - набор, сформированный из адресов i-х элементов каждой последовательности из *S*:

$$S_a = < a_1\ b_1\ ...c_1 > , < a_2\ b_2\ ...c_2 > ,...$$

Последовательность данных потока *S*, обозначаемая S_v, - это последовательность, чей i-й элемент - набор, образованный слиянием наборов значений i-х элементов каждой последовательности из *S*:

$$S_v = < t_1\ u_1\ ...v_1 > , < t_2\ u_2 ...v_2 > ,...$$

Если S_x - последовательность элементов, где каждый элемент - набор из n чисел, то для обозначения "ширины" последовательности обозначается как: $w(S_x) = n$.

Из определений S_a, S_v и w сразу следует утверждение: если *S* - это поток ссылок со значениями из n чисел, то

$$w(S_a) = |S| \text{ и}$$
$$w(S_v) = n|S|,$$

где |S| обозначает мощность множества *S*.

Каждая пара *(I, D)* с потоком команд *I* и потоком данных *D* называется *вычислительным шаблоном*, а все компьютеры разбиваются на классы в зависимости от того, какой шаблон они могут исполнить.

Если компьютер может исполнить шаблон *(I, D)*, если он в состоянии:

- выдать $w(I_a)$ адресов команд для одновременной выборки из памяти;
- декодировать и проинтерпретировать одновременно $w(I_v)$ команд;
- выдать одновременно $w(D_a)$ адресов операндов и
- выполнить одновременно $w(D_v)$ операций над различными данными.

Если все эти условия выполнены, то компьютер может быть описан следующим образом:

$$I_{w(Ia)w(Iv)}D_{w(Da)w(Dv)}$$

Классификация Шнайдера (кратко)

Поток ссылок:
S = { (a1 < t1 >) (a2 < t2 >)..., (b1 < u1 >) (b2 < u2 >)..., (c1 < v1 >)(c2 < v2 >)...}
I – поток команд; D – поток данных; (I, D) – вычислительный шаблон;
Sa = < a1 b1 ...c1 > , < a2 b2 ...c2 >,...
Sv = < t1 u1 ...v1 > , < t2 u2 ...v2 > ,...
w(Sx) = n
w(Sa) = |S|
w(Sv) = n|S|,

Компьютер может быть описан следующим образом:

$$I_{w(Ia)w(Iv)}D_{w(Da)w(Dv)}$$

Примеры
I1,1D1,1
I1,1D1,16384
I1,1D64,64

Каждую пару *(I, D)* с потоком команд *I* и потоком данных *D* будем называть *вычислительным шаблоном*, а все компьютеры будем разбивать на классы в зависимости от того, какой шаблон они могут исполнить. В самом деле, компьютер может исполнить шаблон *(I, D)*, если он в состоянии:

- выдать w(Ia) адресов команд для одновременной выборки из памяти;
- декодировать и проинтерпретировать одновременно w(Iv) команд;
- выдать одновременно w(Da) адресов операндов и
- выполнить одновременно w(Dv) операций над различными данными.

Классы компьютеров в соответствии с классификацией Шнайдера

- IssDss - фон-неймановские машины;
- IssDsc - фон-неймановские машины, в которых заложена возможность выбирать данные, расположенные с разным смещением относительно одного и того же адреса, над которыми будет выполнена одна и та же операция. Примером могут служить компьютеры, имеющие команды, типа одновременного выполнения двух операций сложения над данными в формате полуслова, расположенными по указанному адресу.
- IssDsm - SIMD компьютеры без возможности получения уникального адреса для данных в каждом процессорном элементе, включающие MPP, Connection Machine 1 так же, как и систолические массивы.
- IssDcc - многомерные SIMD машины - фон-неймановские машины, способные расщеплять поток данных на независимые потоки операндов;
- IssDmm - это SIMD компьютеры, имеющие возможность независимой модификации адресов операндов в каждом процессорном элементе, например, ILLIAC IV и Connection Machine 2.
- IscDcc - вычислительные системы, выбирающие и исполняющие одновременно несколько команд, для доступа к которым используется один адрес. Типичным примером являются компьютеры с длинным командным словом (VLIW).
- IccDcc - многомерные MIMD машины. Фон-неймановские машины, которые могут расщеплять свой цикл выборки/выполнения с целью обработки параллельно нескольких независимых команд.
- ImmDmm - к этому классу относятся все компьютеры типа MIMD.

Классификация Скилликорна

Архитектура любого компьютера состоит из:

- процессора команд (IP –Instruction Processor);
- процессора данных (DP – Data Processor);
- иерархии памяти (IP – Instruction Memory, DM – Data Memory);
- переключатели.

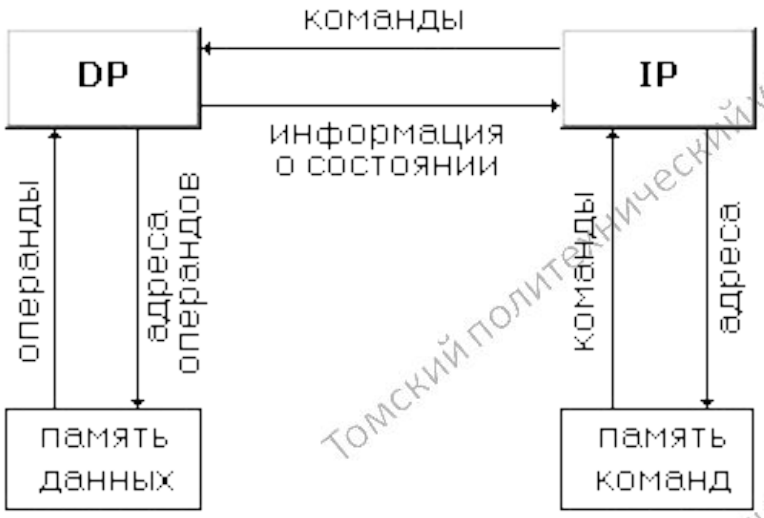
Четыре типа переключателей:

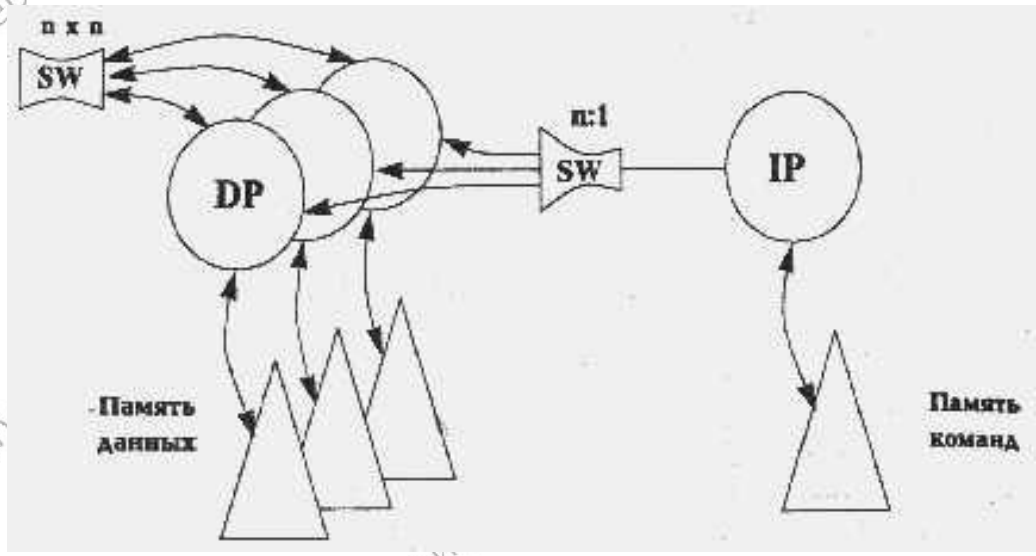
- 1-1 - переключатель такого типа связывает пару функциональных устройств;
- n-n - переключатель связывает i-е устройство из одного множества устройств с i-м устройством из другого множества, т.е. фиксирует попарную связь;
- 1-n - переключатель соединяет одно выделенное устройство со всеми функциональными устройствами из некоторого набора;
- nxn - каждое функциональное устройство одного множества может быть связано с любым устройством другого множества, и наоборот.

Классификация Скилликорна проводится на основе восьми характеристик:

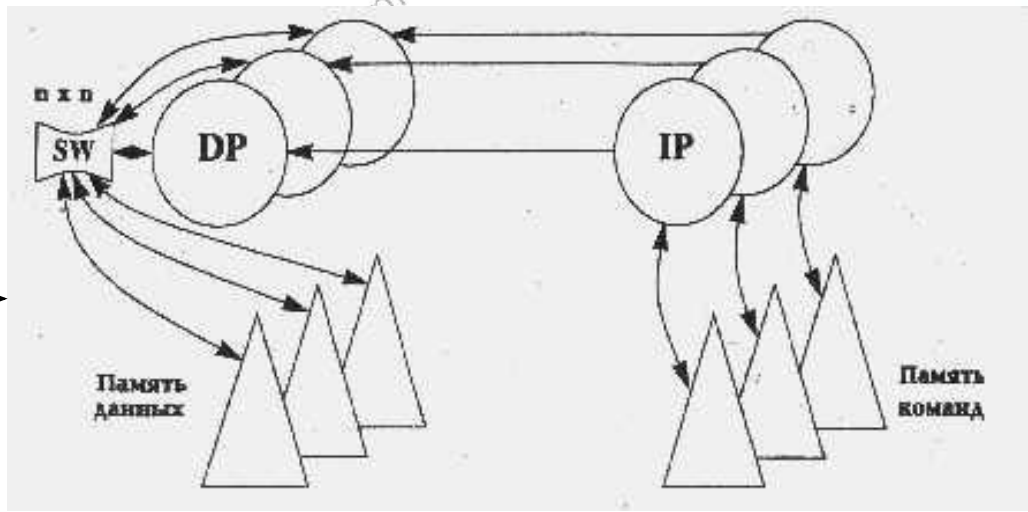
- количество процессоров команд (IP);
- число запоминающих устройств (модулей памяти) команд (IM);
- тип переключателя между IP и IM;
- количество процессоров данных (DP);
- число запоминающих устройств (модулей памяти) данных (DM);
- тип переключателя между DP и DM;
- тип переключателя между IP и DP;
- тип переключателя между DP и DP.

Примеры классификации Скилликорна





Connection Machine 2
(1, 1, 1-1, n, n, n-n, 1-n, nxn)



BBN Butterfly, C.mmp
(n, n, n-n, n, n, nxn, n-n, нет),

Когерентность памяти. Коммутаторы ВС.

1. Организация когерентности многоуровневой иерархической памяти
   a) классифицировать по способу размещения данных в иерархической памяти и способу доступа к этим данным
   b) Однопроцессорный подход к организация механизма неявной реализации когерентности
   c) Многопроцессорный подход к организация механизма неявной реализации когерентности в системах с сосредоточенной памятью
   d) Многопроцессорный подход к организация механизма неявной реализации когерентности в системах физически распределенной памятью
   e) Механизм реализации когерентности и его особенности
   f) Реализация коммутационной среды
2. Коммутаторы вычислительных систем
   a) Общие сведения о коммутаторах. Различия между коммутаторами ВС.
   b) Простые коммутаторы с временным разделением
   c) Алгоритмы арбитража простых коммутаторов с временным разделением.
   d) Особенности реализации шин. Недостатки шинных структур.
   e) Простые коммутаторы с пространственным разделением.
   f) Составные коммутаторы: коммутаторы 2x2, коммутаторы Клоза, распределенные составные коммутаторы.

Многопроцессорную ВС можно рассматривать как совокупность процессоров, подсоединенных к многоуровневой иерархической памяти. При таком представлении коммуникационная среда, объединяющая процессоры и блоки памяти, составляет неотъемлемую часть иерархической памяти.

Классифицировать по способу размещения данных в иерархической памяти и способу доступа к этим данным

- **Явное размещение данных; явное указание доступа к данным** (send, receive).

  Программист явно задает действия по поддержке когерентности памяти посредством передачи данных, программируемой с использованием специальных команд "послать" (send) и "принять" (receive). Каждый процессор имеет свое собственное адресное пространство (память ВС распределена), а согласованность элементов данных выполняется путем установления соответствия между областью памяти, предназначенной для передачи командой send, и областью памяти, предназначенной для приема данных командой receive, в другом блоке памяти.

- **Неявное размещение данных; неявное указание доступа к данным** (load, store).

  В ВС **с** разделяемой памятью механизм реализации когерентности прозрачен для прикладного программиста, и в программах отсутствуют какие-либо другие команды обращения к памяти, кроме команд "чтение" (load) и "запись" (store). Используется единое физическое или виртуальное адресное пространство.

  Преимущества:

- однородность адресного пространства памяти, позволяющая при создании приложений не учитывать временные соотношения между обращениями к разным блокам иерархической памяти;
- создание приложений в привычных программных средах;
- легкое масштабирование приложений для исполнения на разном числе процессоров и разных ресурсах памяти.

Классифицировать по способу размещения данных в иерархической памяти и способу доступа к этим данным

- **Неявное размещение данных как страниц памяти; явное указание доступа к данным.**

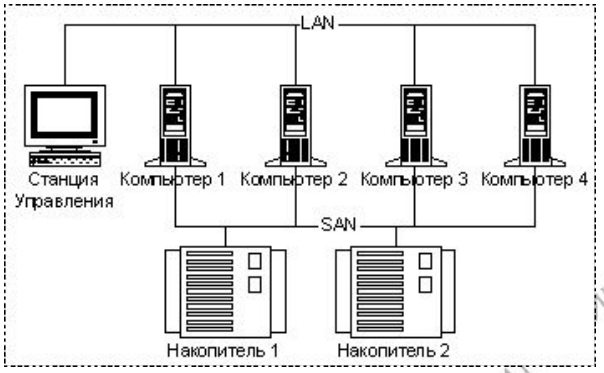
  В этой архитектуре используется разделяемое множество страниц памяти, которые размещаются на внешних устройствах. При явном запросе страницы автоматически обеспечивается когерентность путем пересылки уже запрошенных ранее страниц не из внешней памяти, а из памяти модулей, имеющих эти страницы.

- **Явное размещение данных с указанием разделяемых модулями страниц; неявное указание доступа к данным посредством команд load, store.**

  В каждом компьютере кластера предполагается организация памяти на основе механизма виртуальной адресации. Адрес при этом состоит из двух частей: группы битов, служащих для определения номера страницы, и адреса внутри страницы. В каждом компьютере в ходе инициализации выделяется предписанное, возможно разное, вплоть до полного отсутствия, количество физических страниц памяти, разделяемых этим компьютером с другими компьютерами кластера.

  После установления во всех компьютерах отображения страниц памяти, доступ к удаленным страницам памяти выполняется посредством обычных команд чтения (load) и записи (store).

Кластерные системы



LAN – Local Area Network, локальная сеть
SAN – Storage Area Network, сеть хранения данных

Впервые в классификации вычислительных систем термин "кластер" определила компания Digital Equipment Corporation (DEC).

По определению DEC, кластер - это группа вычислительных машин, которые связаны между собою и функционируют как один узел обработки информации.

Каждый компьютер кластера имеет встраиваемую в него интерфейсную плату-адаптер "шина компьютера — входной и выходной каналы (линки) некоторой среды передачи, данных". В области адресов устройств ввода/вывода шины размещаются две таблицы управления страницами памяти:

1. для выдачи обращений в удаленные разделяемые (общие) страницы памяти других компьютеров;
2. для приема обращений из других компьютеров в локальные разделяемые страницы рассматриваемого компьютера.

Каждый элемент таблицы, используемый при выдаче обращений, содержит:

1) данные, необходимые для доставки сообщения в другой компьютер кластера (например, ID компьютера, в памяти которого находится разделяемая страница);

2) данные, необходимые для точного указания места в странице, к которому должен быть осуществлен доступ по чтению или записи;

3) служебные данные, указывающие на состоятельность рассматриваемого элемента, особенности маршрутизации и т.д.

На основе этих данных адаптер формирует сообщения (пакеты), которые передаются через выходной линк в сеть передачи данных.

Кластерные системы

Сообщение, доставленное в компьютер-адресат, воспринимается через входной линк адаптером этого компьютера.

Сообщение содержит один, из перечисленных ниже, видов информации.

1. команду чтения в совокупности с адресом блока данных, который необходимо прочитать и передать в компьютер, выполняющий команду чтение
2. команду записи в совокупности с адресом, указывающим на место записи данных, и сами записываемые данные.
3. Сообщение, содержащее информационные сигналы прерываний для удаленной шины и синхронизирующих примитивов, необходимых для взаимного исключения одновременного доступа совместно протекающих процессов к областям разделяемой памяти.

Фирма Encore Computer Corporation запатентовала технологию MEMORY CHANNEL, используемой для эффективной организации кластерных систем на базе модели разделяемой памяти.

Кластерная система ASCI-Red. частотой 200 МГц. Быстродействие 1,3 терафлопс. В 1997 году возглавляла список TOP500 и была самая быстрая в мире. Сейчас устарела и снята с эксплуатации в 2006 году.

Blue Gene/L, собран на основе 65536 процессоров, а его производительность оценивают 280,6 терафлопсами

Blue Gene/L

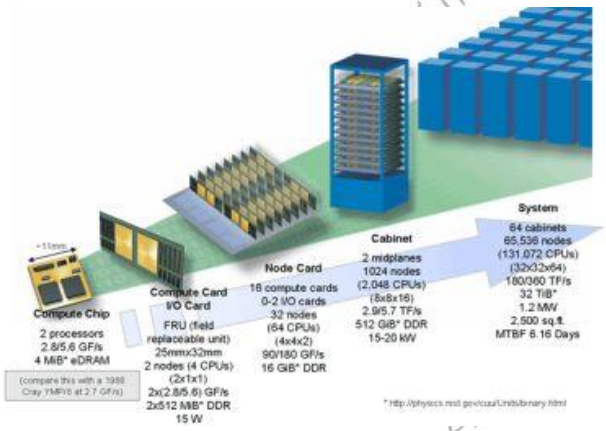
Расположение:
Ливерморская национальная лаборатория имени Лоуренса

Общее число процессоров 65536 штук

Состоит из 64 стоек

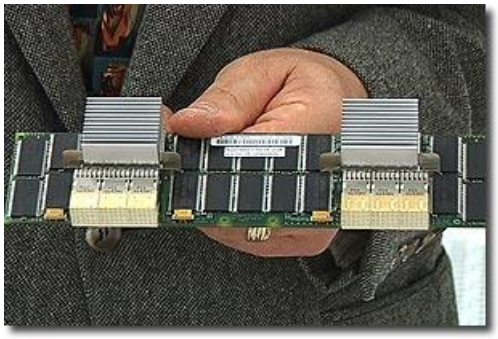
Производительность 280,6 терафлопс

В штате лаборатории - порядка 8000 сотрудников, из которых - более 3500 ученых и инженеров.



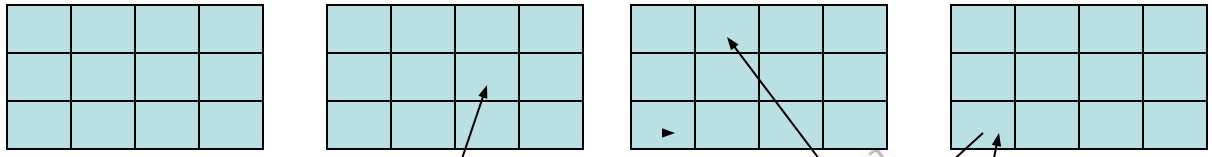
Машина построена по сотовой архитектуре, то есть, из однотипных блоков, что предотвращает появление "узких мест" при расширении системы.

Стандартный модуль BlueGene/L - "compute card" - состоит из двух блоков-узлов (node), модули группируются в модульную карту по 16 штук, по 16 модульных карт устанавливаются на объединительной панели (midplane) размером 43,18 x 60,96 x 86,36 см, при этом каждая такая панель объединяет 512 узлов. Две объединительные панели монтируются в серверную стойку, в которой уже насчитывается 1024 базовых блоков-узлов.



На каждом вычислительном блоке (compute card) установлено по два центральных процессора и по четыре мегабайта выделенной памяти

Процессор PowerPC 440 способен выполнять за такт четыре операции с плавающей запятой, что для заданной тактовой частоты соответствует пиковой производительности в 1,4 терафлопс для одной объединительной панели (midplane), если считать, что на одном узле установлено по одному процессору. Однако на каждом блоке-узле имеется еще один процессор, идентичный первому, но он призван выполнять телекоммуникационные функции.



Адрес	Модуль	Адрес в модуле
0000		
…		
0675	028	002
…		

Разместить переменную А по адресу 007 модуля 27

Разместить переменную А по адресу 0675

Данные из ОП модуля N

Считать В по адресу 009 модуля 29

Механизм неявной реализации когерентности

Реализация механизма когерентности в ВС с разделяемой памятью требует аппаратурно-временных затрат. Уменьшить временную составляющую затрат можно за счет увеличения аппаратурной составляющей и наоборот.

При организации механизма неявной когерентности требуется рассматривать два подхода:

Однопроцессорный.

Организация когерентности кэш-памяти: кэш-память с прямым доступом, частично-ассоциативная кэш-память и ассоциативная память.

Многопроцессорный.

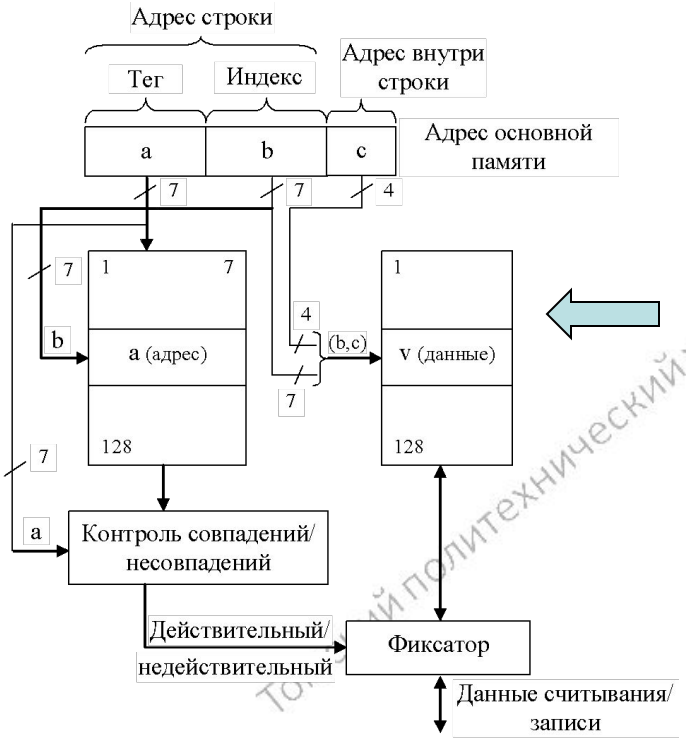
Организация когерентности между кэш-памятью процессорных модулей, либо оперативной памятью различных вычислительных модулей. Существует методы поддержки когерентности для сосредоточенной памяти и физически распределенной памяти.

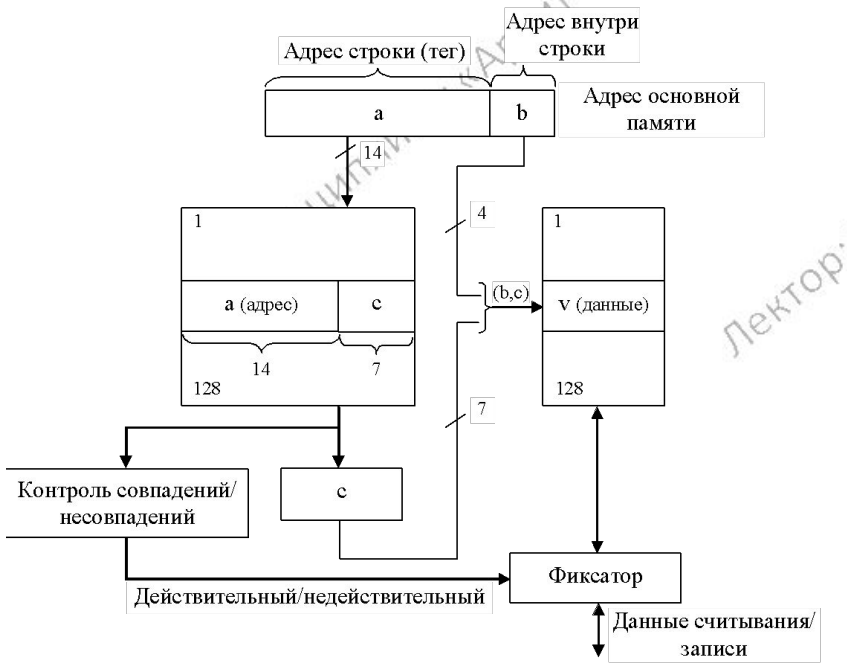
Способы организации кэш-памяти при однопроцессорном подходе

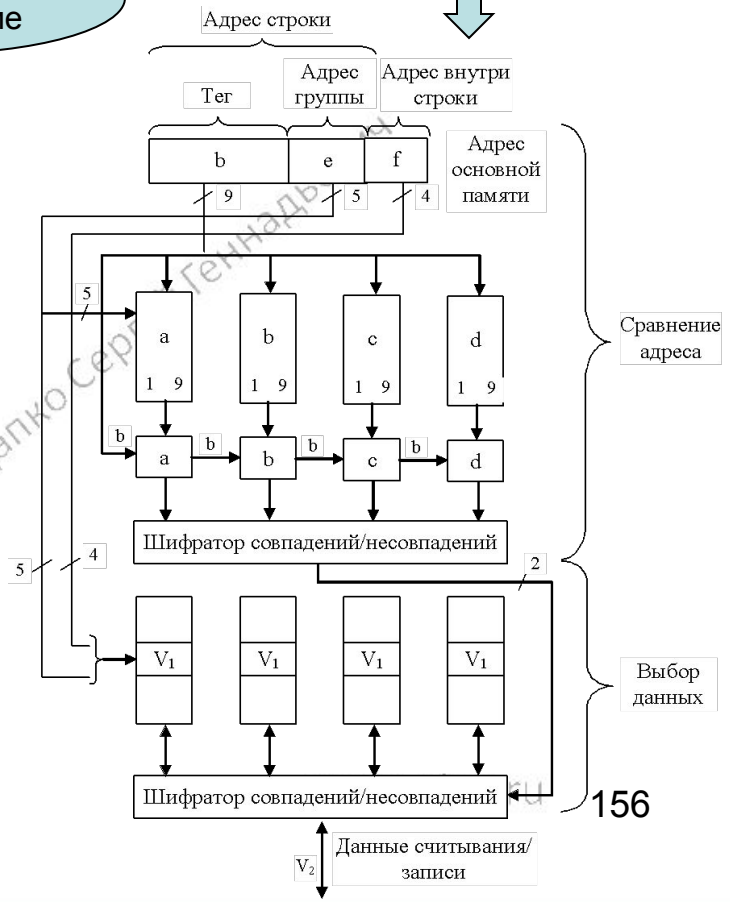
Прямое отображение

Ассоциативное отображение

Частично ассоциативное отображение







Методы обновления ОП при однопроцессорном подходе к организация механизма неявной реализации когерентности (организация когерентности при однопроцессорном подходе)

- *Первый способ* предполагает внесение изменений в оперативную память сразу после их возникновения в кэше. Кэш-память, работающая в таком режиме, называется памятью со сквозной записью.
- *Второй способ* предполагает отображение изменений в основной памяти только в момент вытеснения строки данных из кэша. Кэш-память при таком способе обновления называется кэш-памятью с обратной записью.

Сосредоточенная память

Каждый ВМ имеет собственную локальную кэш-память, имеется общая разделяемая основная память, все ВМ подсоединены к основной памяти посредством шины.

К шине подключены также внешние устройства.

Все действия с использованием транзакций шины, производимые ВМ и внешними устройствами, с копиями строк, как в каждой кэш-памяти, так и в основной памяти, доступны для отслеживания всем ВМ.

Многопроцессорный подход к организация механизма неявной реализации когерентности в системах с сосредоточенной памятью

Алгоритм поддержки когерентности кэшей – MESI (Modified, Exclusive, Shared, Invalid), представляет собой организацию когерентности кэш-памяти с обратной записью.

Каждая строка кэш-памяти ВМ может находиться в одном из следующих состояний:
M – строка модифицирована (доступна по чтению и записи только в этом ВМ, потому что модифицирована командой записи по сравнению со строкой основной памяти);
E – строка монопольно копированная (доступна по чтению и записи в этом ВМ и в основной памяти);
S – строка множественно копированная или разделяемая (доступна по чтению и записи в этом ВМ, в основной памяти и в кэш-памятях других ВМ, в которых содержится ее копия);
I – строка, невозможная к использованию (строка не доступна ни по чтению, ни по записи).

Состояние строки используется, во-первых, для определения процессором ВМ возможности локального, без выхода на шину, доступа к данным в кэш-памяти, а, во-вторых, — для управления механизмом когерентности.

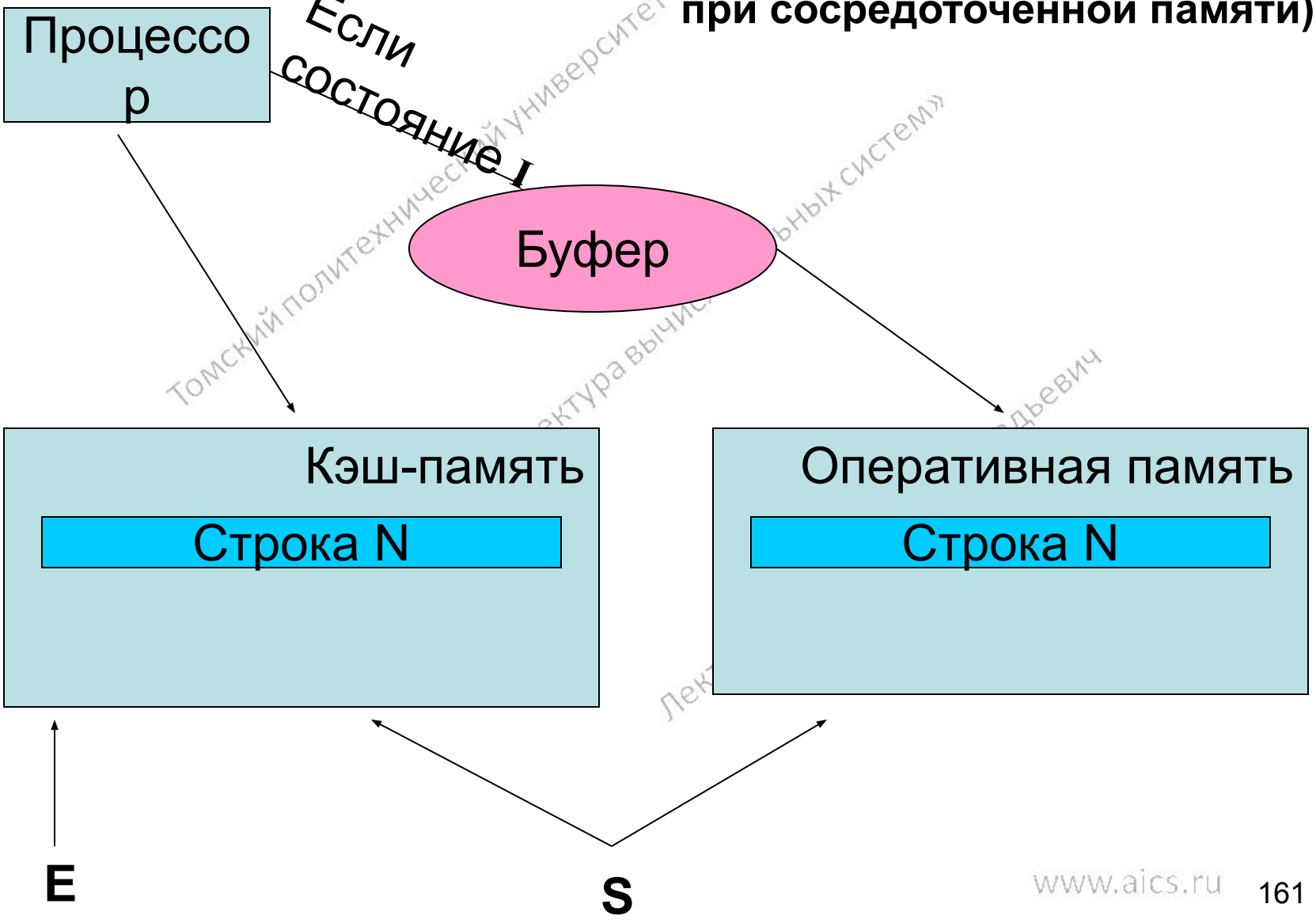
Многопроцессорный подход к организация механизма неявной реализации когерентности в системах с сосредоточенной памятью

Исх. состояние строки	Состояние после чтения	Состояние после записи
I	Если WT = 1, тогда **E**, иначе **S**; Обновление строки путем ее чтения из основной памяти	Сквозная запись в основную память; **I**
S	**S**	Если WT = 1 тогда **E**, иначе **S**
E	**E**	**M**
M	**M**	**M**

Для управления режимом работы механизма поддержки когерентности используется бит WT, состояние 1 которого задает режим сквозной записи, а состояние 0 – режим обратной записи в кэш-память.

Кэш-память заполняется только при промахах чтения. При промахе записи транзакция записи помещается в буфер и посылается в основную память при предоставлении шины.

Реализация когерентности (многопроцессорный подход при сосредоточенной памяти)



Многопроцессорный подход к организация механизма неявной реализации когерентности в системах физически распределенной памятью

Прямолинейный подход к поддержанию когерентности кэшей в мультипроцессорной системе, основная память которой распределена по ВМ, заключается в том, что при каждом промахе в кэш в любом процессоре инициируется запрос требуемой строки из того блока памяти, в котором эта строка размещена. Этот блок памяти называться ***резидентным***.

Запрос передается через коммутатор в модуль с резидентным для строки блоком памяти, из которого затем необходимая строка через коммутатор пересылается в модуль, в котором произошел промах.

При этом в каждом модуле для каждой резидентной строки ведется список модулей, в кэшах которых эта строка размещается, либо организуется распределенный по ВМ список этих строк. Строка, размещенная в кэше более чем одного модуля, называться ***разделяемой***.

Многопроцессорный подход к организация механизма неявной реализации когерентности в системах физически распределенной памятью

Когерентность кэшей обеспечивается следующим. При обращении к кэш-памяти в ходе операции записи данных, после самой записи, процессор приостанавливается до тех пор пока не выполнится последовательность, как минимум, из трех действий: измененная строка кэша пересылается в резидентную память модуля, затем, если строка была разделяемой, она пересылается из резидентной памяти во все модули, указанные в списке разделяющих эту строку. После получения подтверждений, что все копии изменены, резидентный модуль пересылает в процессор, приостановленный после записи, разрешение продолжать вычисления.

Для обеспечения наименьших простоев процессоров можно использовать алгоритм DASH.

Алгоритм DASH

Каждый модуль памяти имеет для каждой строки, резидентной в модуле, список модулей, в кэшах которых размещены копии строк.

С каждой строкой в резидентном для нее модуле связаны три ее возможных глобальных состояния:

1) "некэшированная", если копия строки не находится в кэше какого-либо другого модуля, кроме, возможно, резидентного для этой строки;

2) "удаленно-разделенная", если копии строки размещены в кэшах других модулей;

3) "удаленно-измененная", если строка изменена операцией записи

в каком-либо модуле.

Кроме этого, каждая строка кэша находится в одном из трех локальных состояний:

1) "невозможная к использованию";

2) "разделяемая", если есть неизмененная копия, которая, возможно, размещается также в других кэшах;

3) "измененная", если копия изменена операцией записи.

Алгоритм DASH

Каждый процессор может читать из своего кэша, если состояние читаемой строки "разделяемая" или "измененная". Если строка отсутствует в кэше или находится в состоянии "невозможная к использованию", то посылается запрос "промах чтения", который направляется в модуль, резидентный для требуемой строки.

Если глобальное состояние строки в резидентном модуле "некэшированная" или "удаленно-разделенная", то копия строки посылается в запросивший модуль и в список модулей, содержащих копии рассматриваемой строки, вносится модуль, запросивший копию.

Если состояние строки "удаленно-измененная", то запрос "промах чтения" перенаправляется в модуль, содержащий измененную строку. Этот модуль пересылает требуемую строку в запросивший модуль и в модуль, резидентный для этой строки, и устанавливает в резидентном модуле для этой строки состояние "удаленно-разделенная".

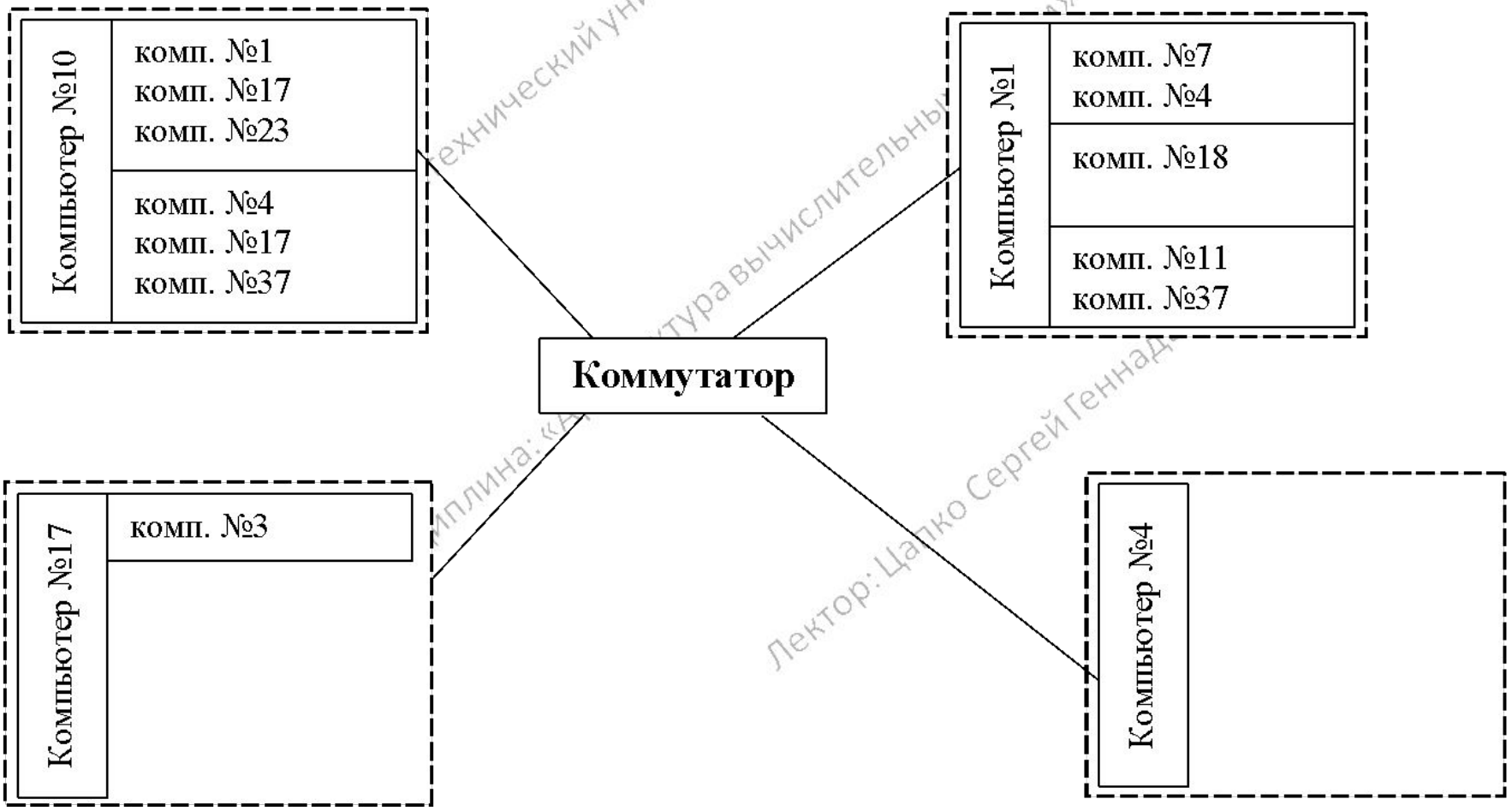
Алгоритм DASH

Если процессор выполняет операцию записи и состояние строки, в которую производится запись "измененная", то запись выполняется и вычисления продолжаются. Если состояние строки "невозможная к использованию" или "разделяемая", то модуль посылает в резидентный для строки модуль запрос на захват в исключительное использование этой строки и приостанавливает выполнение записи до получения подтверждений, что все остальные модули, разделяющие с ним рассматриваемую строку, перевели ее копии в состояние "невозможная к использованию«.

Если глобальное состояние строки в резидентном модуле "некэшированная", то строка отсылается запросившему модулю, и этот модуль продолжает приостановленные вычисления.

Если глобальное состояние строки "удаленно-разделенная", то резидентный модуль рассылает по списку всем модулям, имеющим копию строки, запрос на переход этих строк в состояние "невозможная к использованию". По получении этого запроса каждый из модулей изменяет состояние своей копии строки на "невозможная к использованию" и посылает подтверждение исполнения в модуль, инициировавший операцию записи. При этом в приостановленном модуле строка после исполнения записи переходит в состояние "удаленно-измененная".

Пример обеспечения когерентности памятей ВМ



Механизм явной реализации когерентности

При явной реализации когерентности используются отдельные наборы команд типа load, store для работы с локальной памятью ВМ и специальные команды (вызовы процедур) типа send, receive для управления адаптерами каналов ввода/вывода. Задача программиста – эффективно запрограммировать передачи данных, совмещая их по возможности с вычислениями и минимизируя объем передаваемых данных.

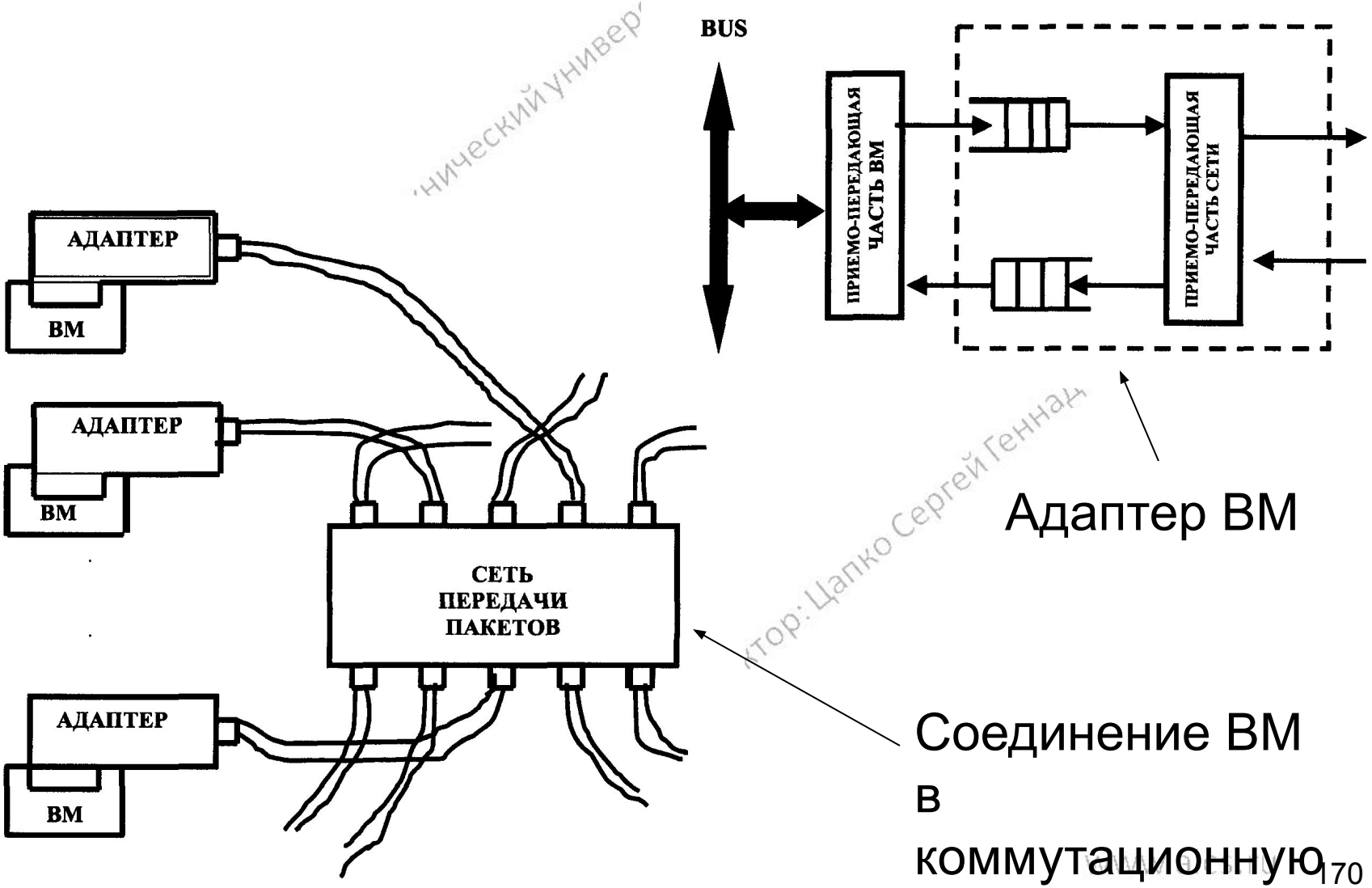
Использование явной реализации когерентности обусловлено недопустимо большими затратами аппаратуры или времени на реализацию неявного механизма когерентности в создаваемой ВС.

Реализация коммутационной среды

Процесс реализации коммутационной среды можно разделить на три этапа.

1. На структурном уровне коммуникационная среда состоит из трех компонентов:
   a. адаптеров, осуществляющих интерфейс между ВМ и сетью передачи пакетов;
   b. коммутаторов сети передачи пакетов;
   c. кабелей, служащих для подсоединения входных и выходных каналов (линков) адаптеров к портам коммутатора.
2. Адаптеры состоят из двух частей: приемопередающей части ВМ и приемопередающей части сети, между которыми, как правило, имеется согласующий буфер.
3. Для маршрутизации пакетов по сети необходимо принять соглашение об идентификации ВМ системы. Механизмы реализации это – соглашение об отображении адресов и элементов распределенной иерархической многоуровневой памяти.

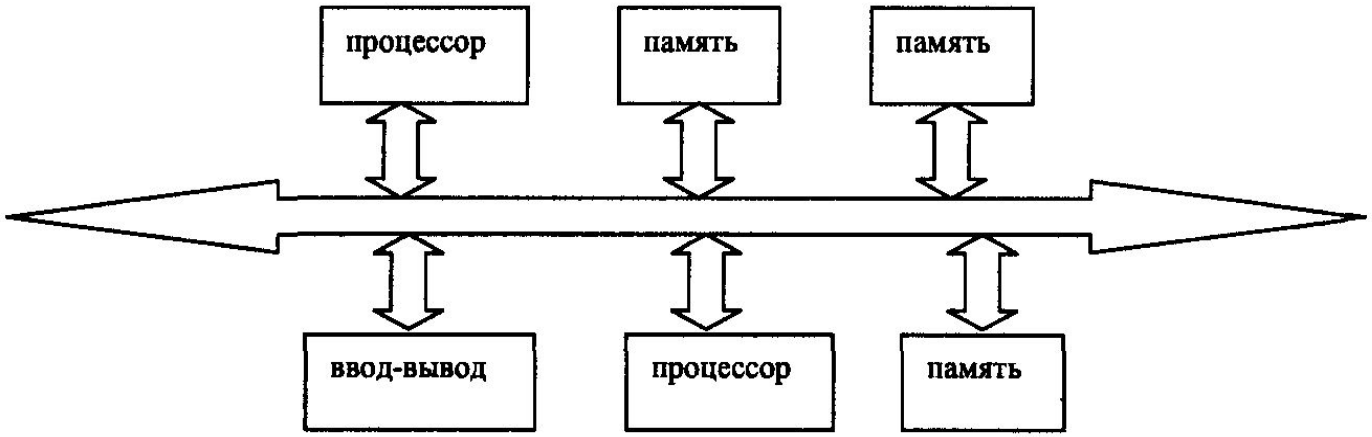
Реализация коммутационной среды



Простые коммутаторы с временным разделением

Простые коммутаторы бывают с временным и пространственным разделением. Достоинства простых коммутаторов: простота управления и высокое быстродействие.

Особенность заключается в использовании общей информационной магистрали для передачи данных между устройствами, подключенными к шине

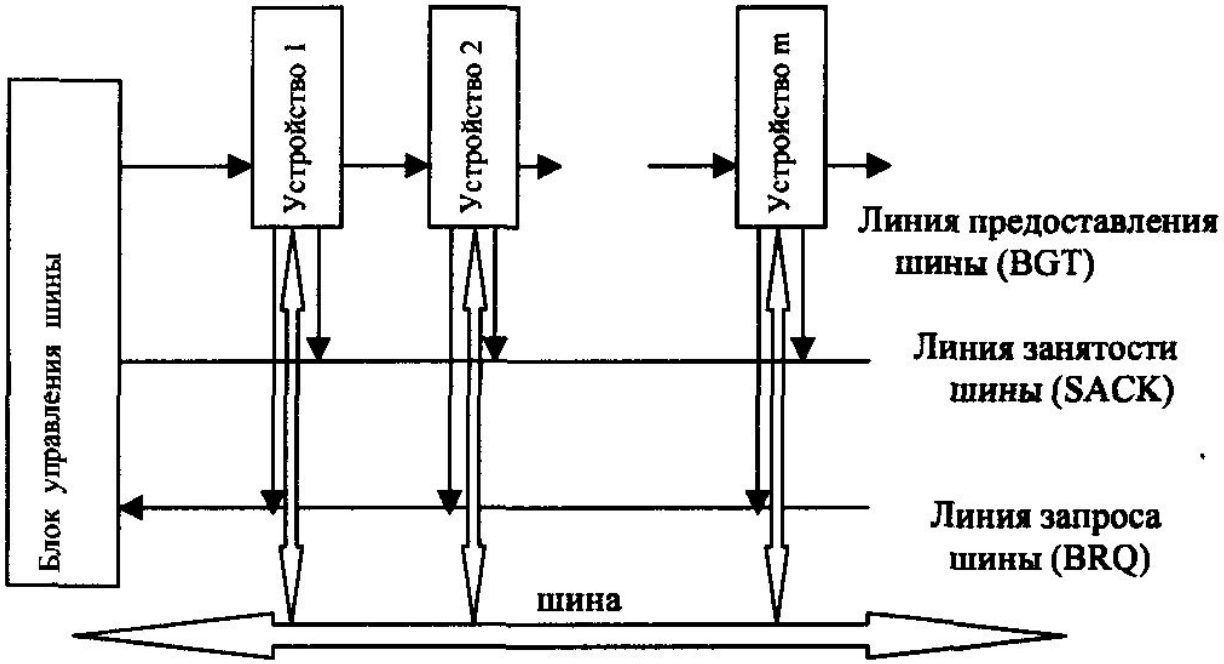


Как правило, шины состоят только из пассивных элементов, и все управление передачами выполняется передающим и принимающим устройствами.

Для разрешения конфликтов при одновременном запросе на передачу от нескольких устройств используются алгоритмы арбитража

Алгоритмы арбитража. Статические приоритеты

Каждому устройству приписывается уникальный приоритет. Когда несколько устройств одновременно запрашивают шину для передачи, то шина предоставляется устройству с наивысшим приоритетом из числа запросивших.



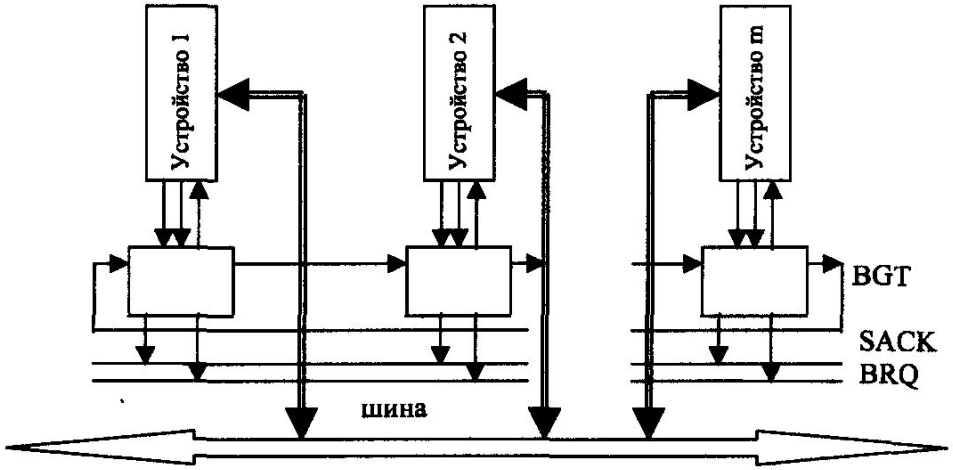
Устройство, расположенное ближе к централизованному блоку управления шиной, имеет больший приоритет.

Алгоритмы арбитража. Фиксированные временные интервалы

Алгоритм предоставляет каждому устройству одинаковый временной интервал по циклической дисциплине. Если устройство получило временной интервал и не имеет данных для передачи, т. е. не использует временной интервал, то этот интервал не предоставляется другому устройству. Алгоритм используется в синхронных шинах, в которых применяется один тактовый генератор для всех устройств..

Алгоритмы арбитража. Динамические приоритеты

Устройствам приписываются уникальные приоритеты, но приоритеты динамически изменяются, предоставляя каждому устройству возможность доступа к шине. Применяются в основном два механизма изменения приоритетов: наивысший приоритет предоставляется устройству, наиболее долго не использовавшему шину - LRU, и циклической сменой приоритетов RDC. Первый механизм переприсваивает приоритет каждого устройства после очередного цикла работы шины.

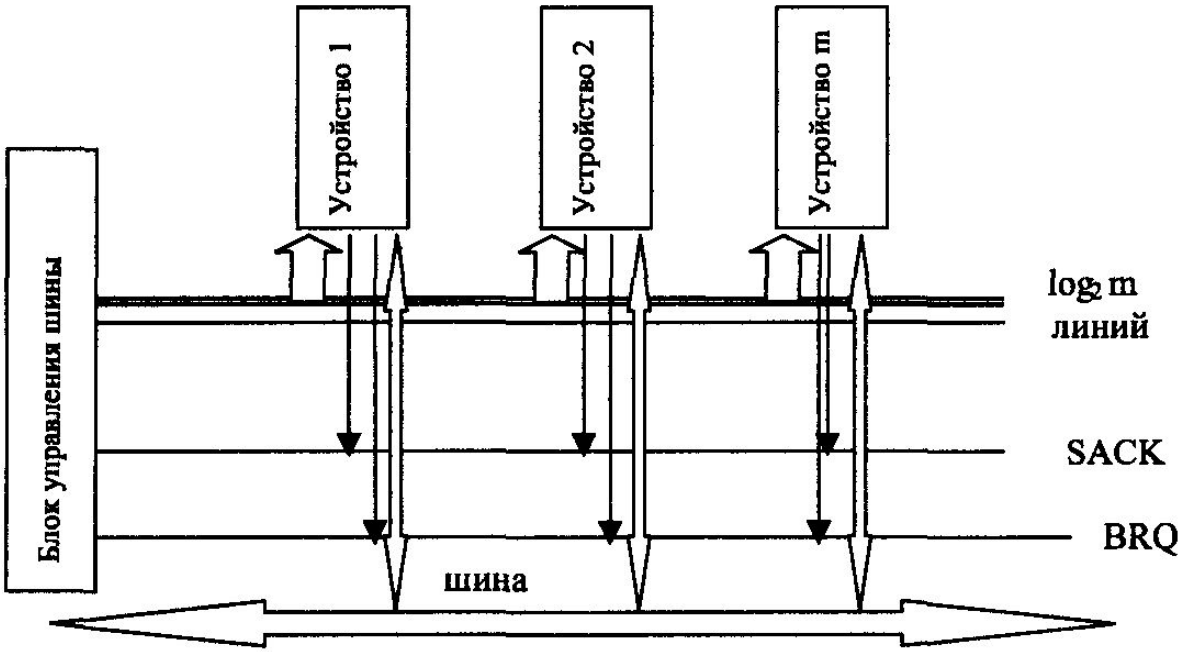


При RDC механизме реализуется распределенный блок управления шиной. Линия BGT предоставления шины циклически соединяет все устройства шины

Устройство, получившее доступ к шине, выступает как контроллер шины при следующем цикле арбитража. Приоритет каждого устройства определяется его расстоянием до устройства, выполняющего в текущий момент роль контроллера и имеющего при этом минимальный приоритет.

Алгоритмы арбитража. Голосование

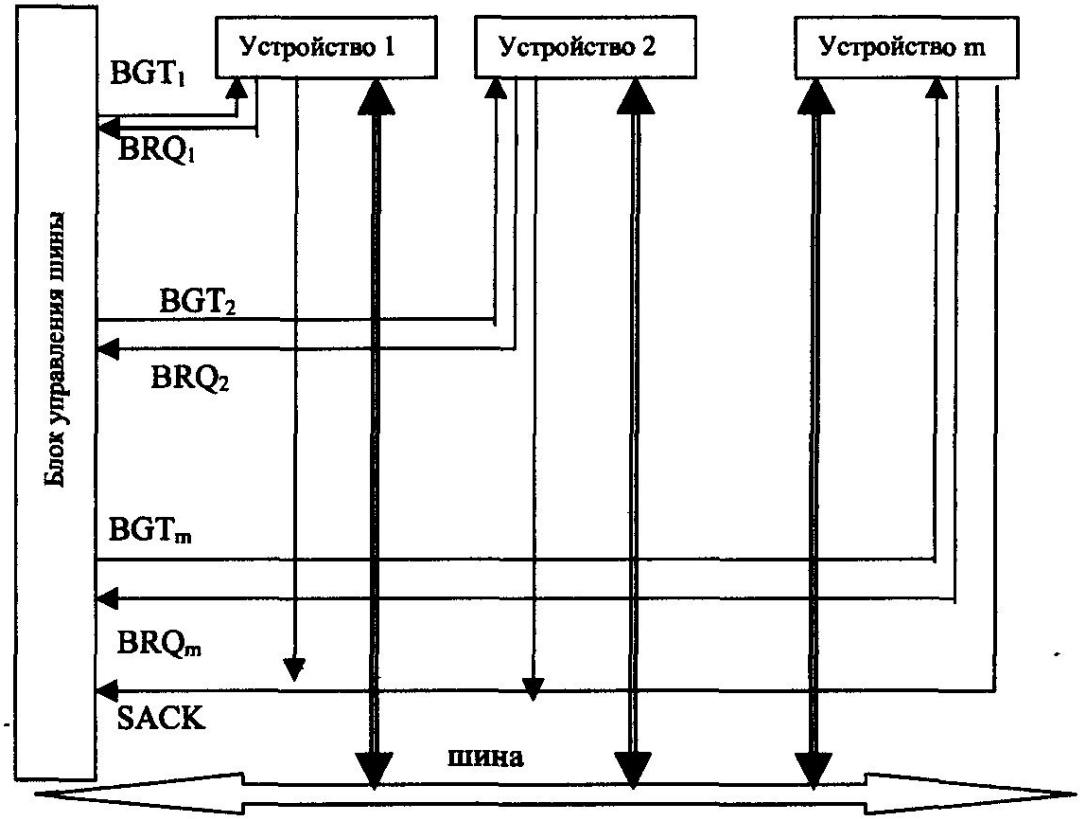
При этом механизме линия BGT предоставления шины представляется совокупностью [$\log_2$ m] линий голосования, где m – число устройств шины, а [x] – ближайшее целое число, большее или равное x.



При запросе шины контроллер начинает выдавать на линии голосования адреса устройств. При обнаружении устройством своего адреса оно выставляет сигнал на линию занятости. Голосование прекращается с тем, чтобы возобновиться после освобождения шины. Приоритет устройств задается порядком выдачи адресов при голосовании.

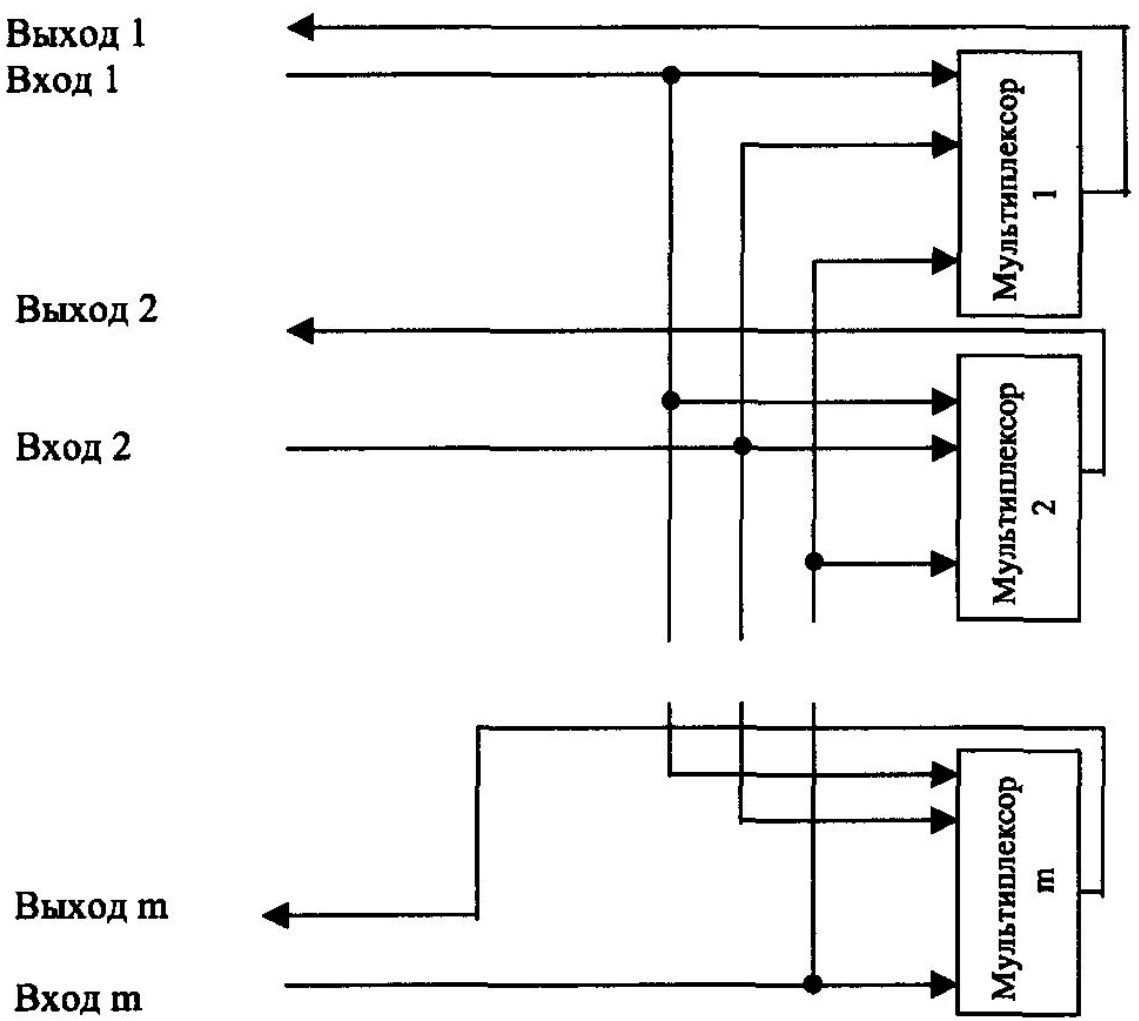
Алгоритмы арбитража. Независимые запросы

Каждое устройство имеет индивидуальные линии запроса и предоставления шины.

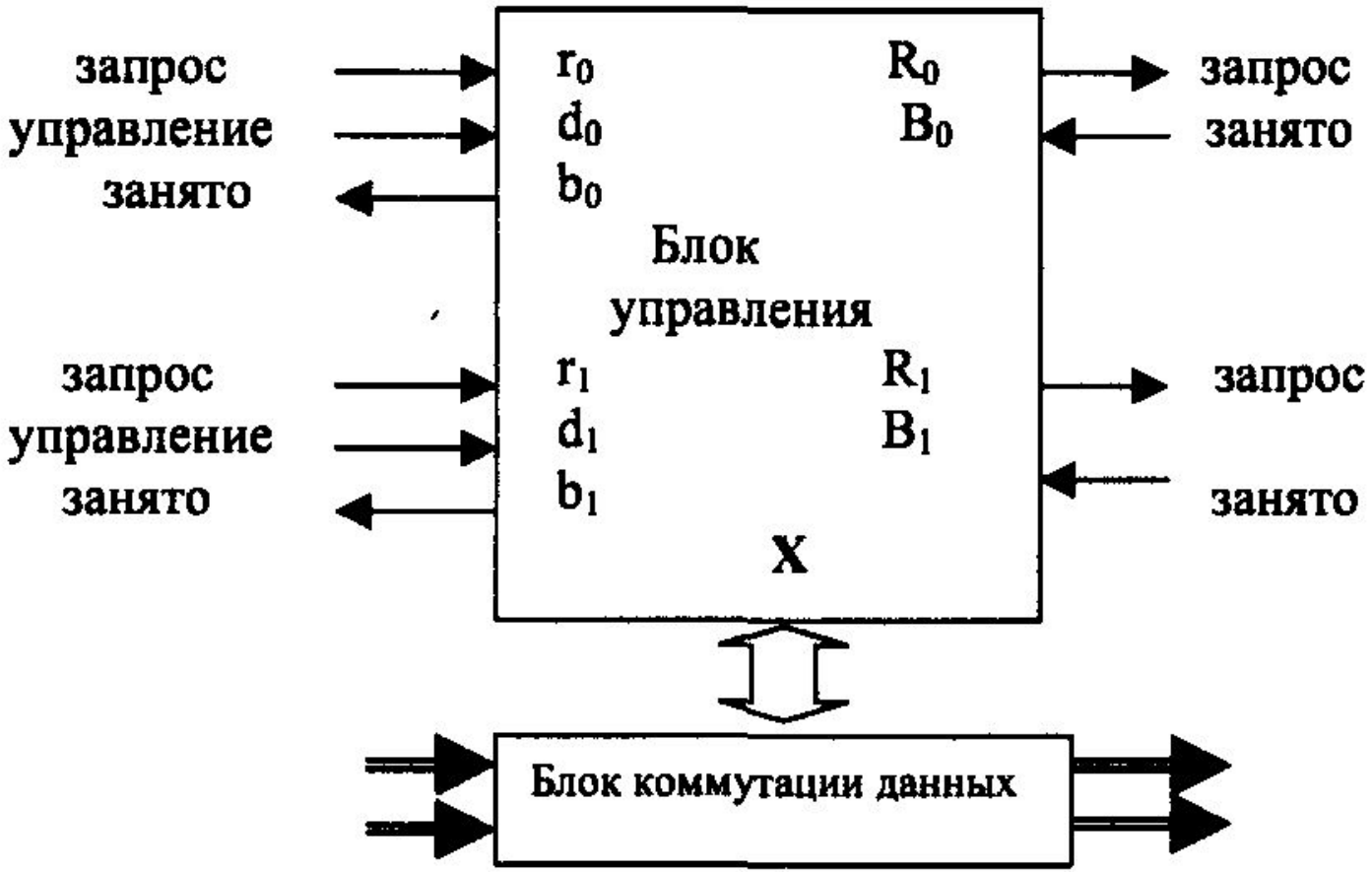


Примером такой шины может служить шина PCI

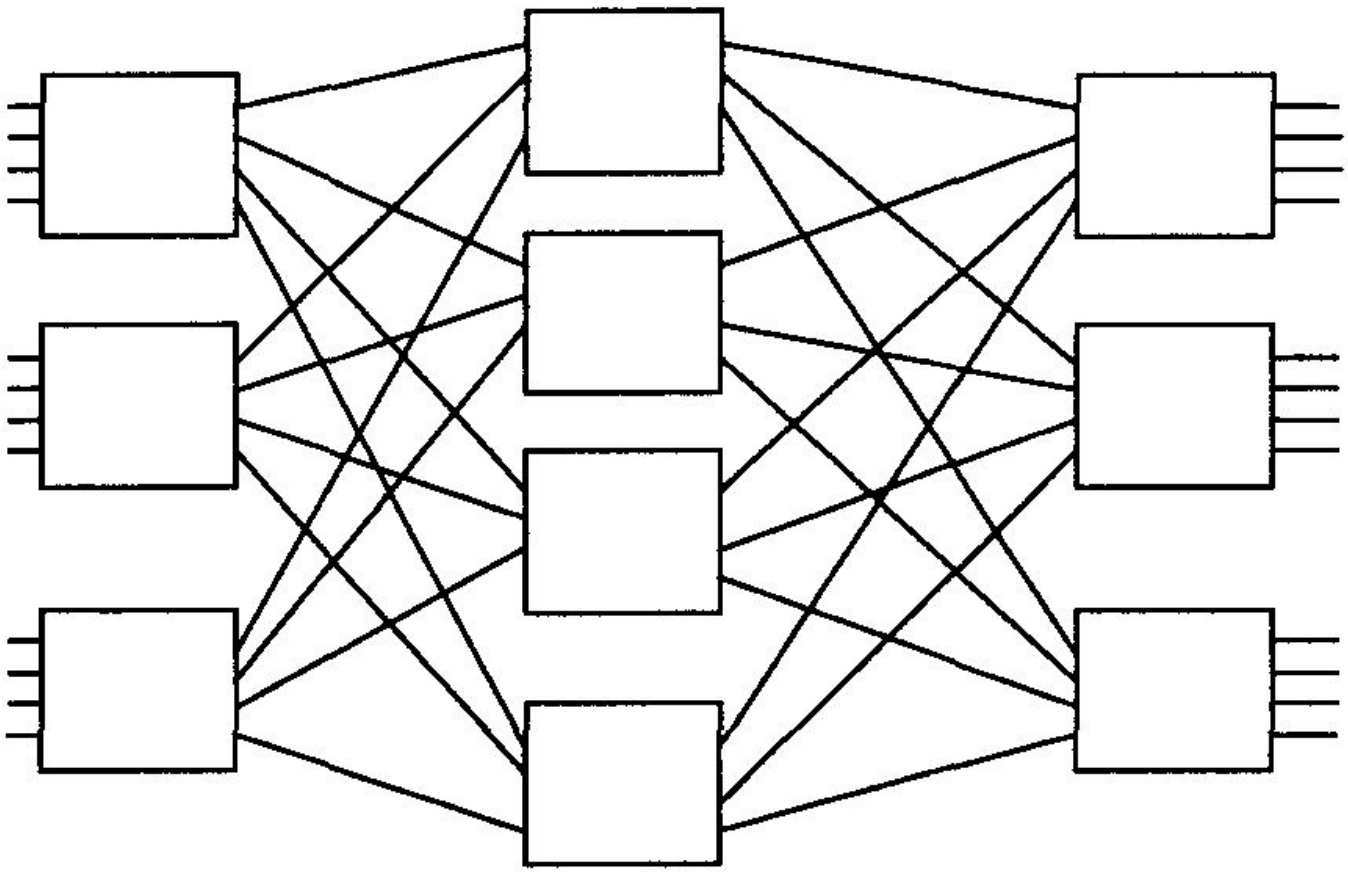
Простые коммутаторы с пространственным разделением



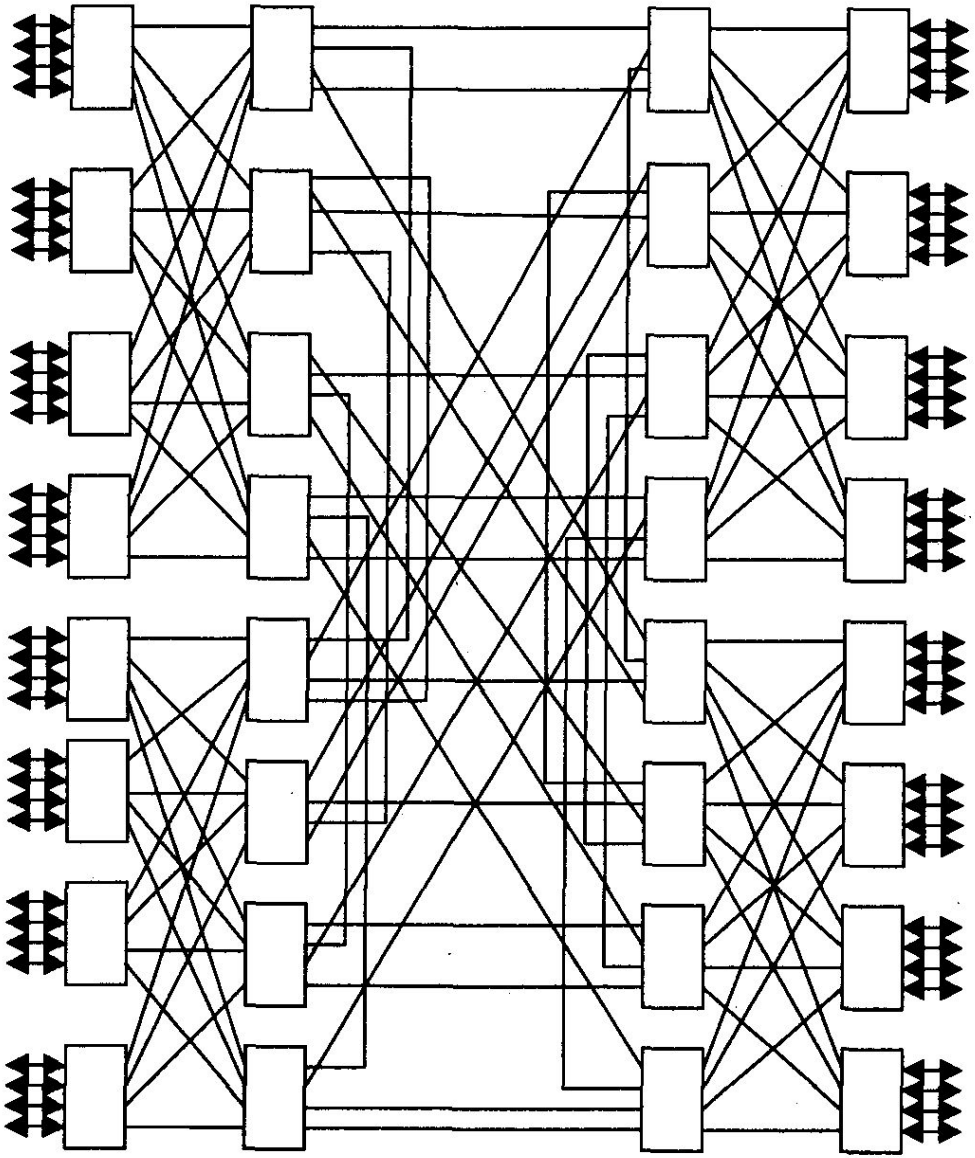
Прямоугольные коммутаторы 2x2



Коммутатор Клоза



Распределенные составные коммутаторы



Список литературы

Иерархическая память многопроцессорных ВС
- http://www.uran.donetsk.ua/~masters/2001/fvti/prokopenko/diss/ch03.htm

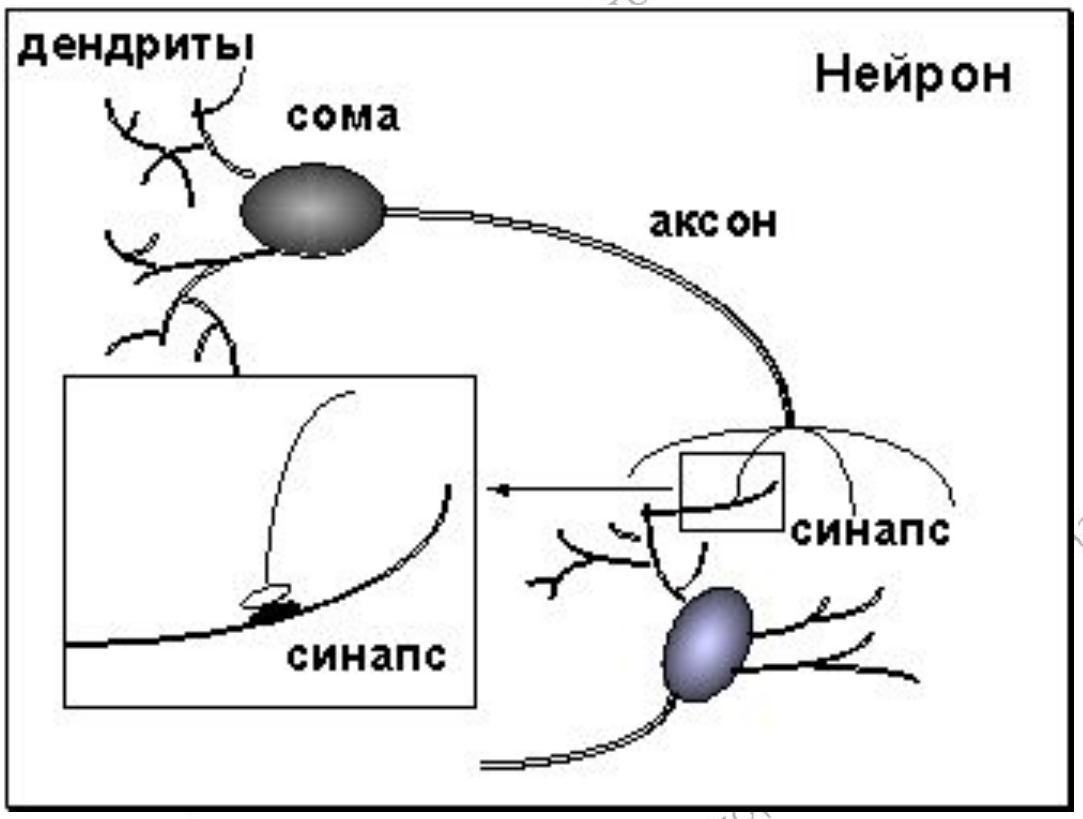
Анализ мультипроцессорных систем с иерархической памятью
- http://masters.donntu.edu.ua/2001/fvti/prokopenko/diss/index.htm
- Многопроцессорные системы
- http://www.dvo.ru/bbc/reff/referat.html

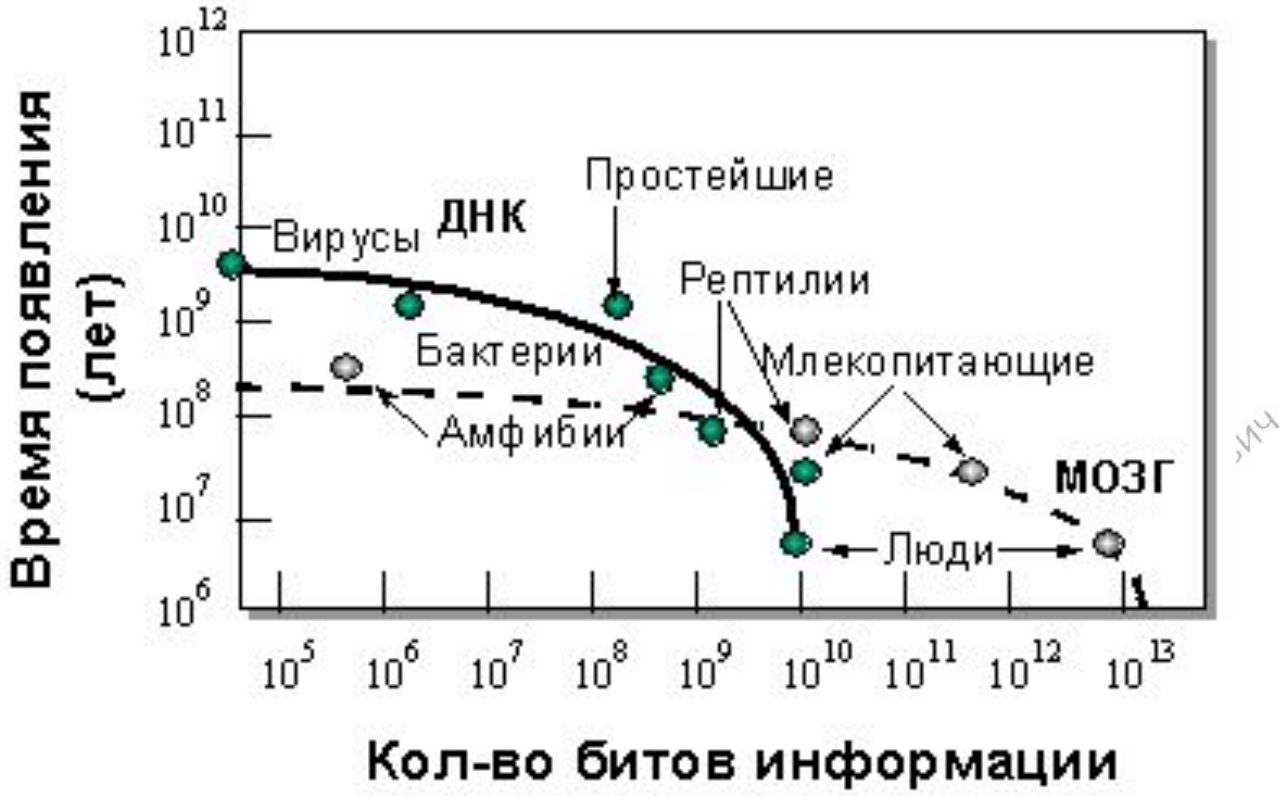
Распределенная общая память
- http://www2.sscc.ru/Litera/krukov/lec6.html

Механизм когерентности обобщенного кольцевого гиперкуба
- http://www.radioland.net.ua/contentid-149.html

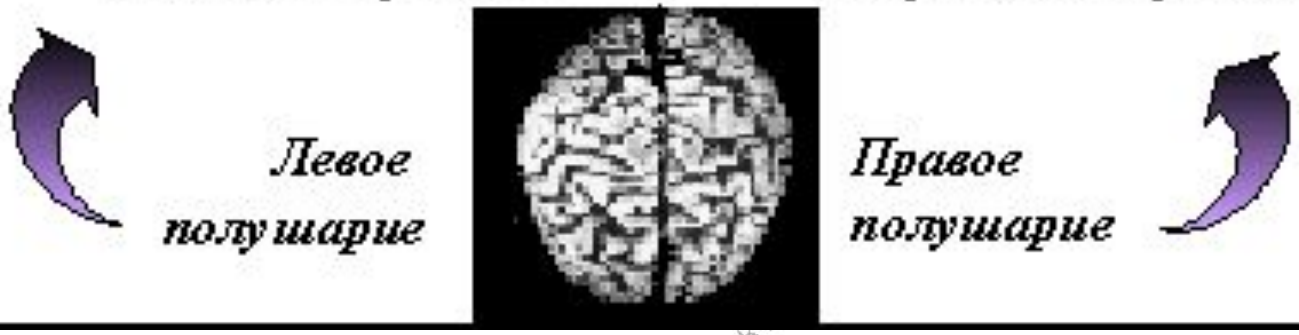
Коммутаторы для многопроцессорных вычислительных систем
- http://informika.ru/text/teach/topolog/5.htm
- Архитектуры с распределенной разделяемой памятью
- http://www.osp.ru/os/2001/03/015.htm
- Коммутаторы для кластеров
- http://kis.pcweek.ru/Year2004/N20/CP1251/Srv_Storage/chapt7.htm

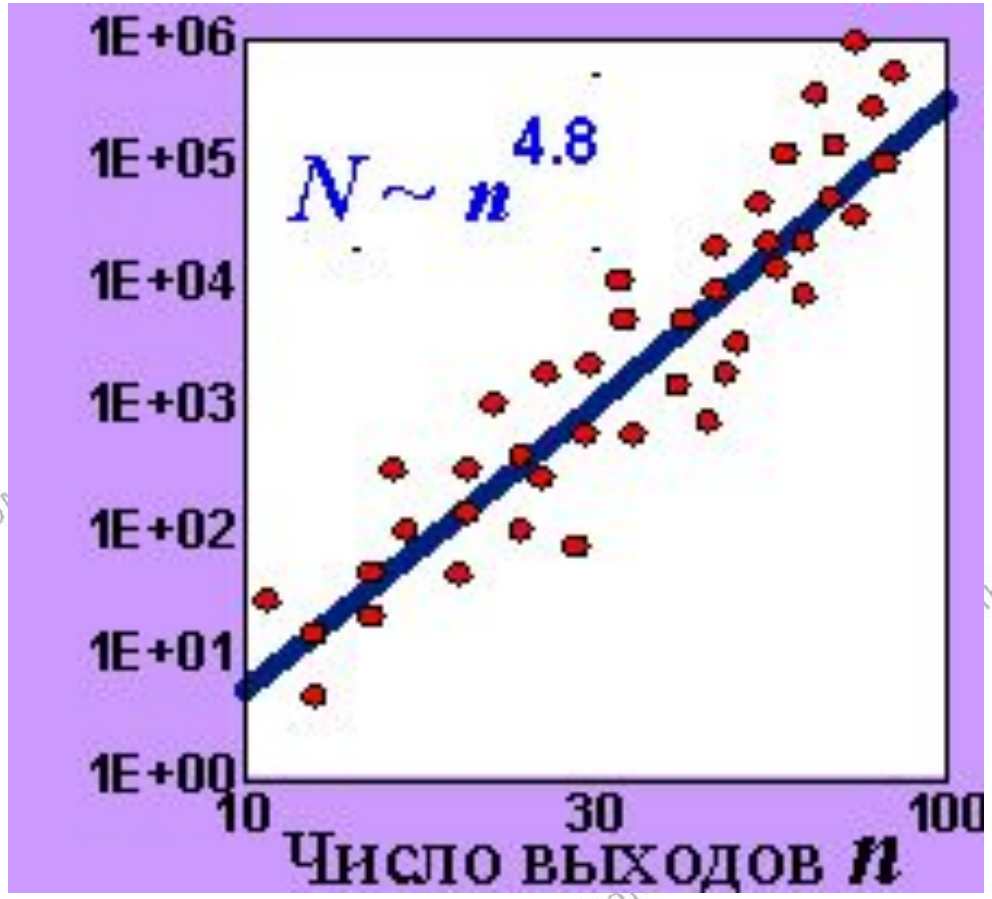
Архитектура высокопроизводительного коммутатора
- http://www.parallel.ru/computers/reviews/sp2_overview.html#switch_arch




Время появления (лет)
10^{12}
10^{11}
10^{10}
10^{9}
10^{8}
10^{7}
10^{6}
Вирусы
ДНК
Простейшие
Рептилии
Бактерии
Млекопитающие
Амфибии
МОЗГ
Люди
10^{5} 10^{6} 10^{7} 10^{8} 10^{9} 10^{10} 10^{11} 10^{12} 10^{13}
Кол-во битов информации

- **Традиционные ЭВМ**
  - Последовательные
  - Заданный алгоритм
  - Иерархическая структура алгоритмов, разбиение сложной задачи на простые
- **Нейрокомпьютеры**
  - Параллельные
  - Алгоритм формируется путем обучения на примерах
  - Непосредственные операции с образами




1E+06
1E+05
1E+04
1E+03
1E+02
1E+01
1E+00
$N \sim n^{4.8}$
10
30
100
Число выходов n

Нейропроцессор – это кристалл, который обеспечивает выполнение нейросетевых алгоритмов в реальном масштабе времени.

Среди разновидностей кристаллов, используемых в качестве нейропроцессоров выделим следующие (рис. 4.1):

- специализированные нейрочипы;
- заказные кристаллы (ASIC);
- встраиваемые микроконтроллеры (mC);
- процессоры общего назначения (GPP);
- перепрограммируемые логические интегральные схемы (FPGA, ПЛИС);
- процессоры цифровой обработки сигналов (ПЦОС);
- транспьютеры.

В свою очередь, для оценки производительности нейропроцессоров и нейрокомпьютеров применяется ряд специальных показателей (параметров):

- ***MMAC*** – миллионов умножений с накоплением в секунду;
- ***CUPS (Connections Update per Second)*** – число измененных значений весов в секунду (оценивает скорость обучения);
- ***CPS (Connections per Second)*** – число соединений (умножений с накоплением) в секунду (оценивает производительность);


Специализированные нейрочипы
Перепрограммируемая логика (FPGA, ПЛИС)
Процессоры цифровой обработки сигналов (ПЦОС)
Нейро-процессоры
Процессоры общего назначения (GPP)
Заказные кристаллы (ASIC)
Встраиваемые микроконтроллеры (μC)