2. Процессоры

2.1. Базовая структура ЭВМ

Базовая структура ЭВМ может быть представлена состоя-щей из шести основных частей (Рис.1): средства ввода, средства вывода, памяти, арифметико-логического устройства, устройства управления и интерфейсного блока

В данной структуре под средствами ввода подразумеваются клавиатура, мышка, джойстик, сканер; средства ввода звуковой, видео информации и т. п.

Под средствами вывода подразумеваются принтеры, звуковые и видео карты и т.д.

В блок памяти включаются как оперативная память, так и внешние запоминающие устройства, такие как магнитные диски, оптические диски и магнитные ленты. Сюда же относится и флэш-память.

Важнейшим компонентом ЭВМ является центральный процессор. Упрощено он может быть представлен состоящим из трёх частей: операционного устройства (ОУ) или операционного блока (ОБ), устройства управления (УУ) и интерфейсного блока.

2.2. Основные характеристики ЭВМ

В качестве основных характеристик ЭВМ обычно рассматривают: быстродействие и производительность емкость памяти, стоимость и надежность.

Для пользователей обычно представляют основной интерес быстродействие и производительность.

Быстродействие оценивают либо как количество стандартных операций в единицу времени, либо как скорость вычислений при выполнении эталонного алго-ритма или некоторого класса алгоритмов.

В качестве стандартных операций обычно выбирают либо короткую операцию сложения,

Производительность является более универ-сальным показателем, чем быстродействие, пос-кольку явно зависит от порядка прохождения за-дач в ЭВМ и оценивается количеством стандарти-зованных тестовых программ выполняемых в единицу времени.

В настоящее время общепринятой практикой является использование набора специально подобранных прикладных программ.

Подбором таких приложений занимается орга-низация System Performance Evaluation Corpora-tion (SPEC). Она публикует списки программ для разных областей применения и результаты тести-рования компьютеров.

В список входят самые разнообразные программы, от игр, компиляторов и приложений баз данных до программ, осуществляющих вычисления в области астрофизики и квантовой механики.

Коэффициент производительности определяется как:

$$SPEC - коэффициент = \frac{T_{эталон}}{T_{тестир}}$$

где $T_{эталон}$ - время выполнения на эталонном компьютере, а $T_{тестир.}$ - время выполнения на тестируемом компьютере.

В каждом случае программа компилируется для тестируемого компьютера и измеряется реальное время её выполнения на тестируемом компьютере.

Никакая эмуляция не допускается.
После этого та же самая программа компилируется и выполняется на эталонном компьютере.

Набор тестовых приложений подвергался модификациям в 2000, 2006 и 2013 годах.

Например, эталонным компьютером для теста SPEC2000 выбрана рабочая станция UltraSPARC10 с процессором UltraSPARC-III, с

Полное тестирование производится по всему списку тестовых приложений, а затем вычисляется среднее геометрическое резуль-татов по отдельным тестам.

Итоговый SPEC - коэффициент рассчитывает-ся по формуле:

$$SPEC - коэффициент = \left(\prod_{i=1}^{n} SPEC_i \right)^{1/n}$$

где n – **количество тестовых программ**.

На быстродействие и производительность влияет целая масса факторов, таких как тактовая частота процессора, пропускная способность интерфейсов, структура процессора, способы представления дан-ных, последовательность исполнения команд, свой-ства программного обеспечения и ещё множество других.

Одним из компонентов общего времени выпол-нения программы является процессорное время.

Допустим, что для выполнения программы пот-ребуется время $T_{проц}$ секунд процессорного времени и выполнить N машинных команд.

N – это не количество команд (кодов) програм-мы, а именно количество выполняемых процессо-ром команд.

Допустим, что для выполнения одной машинной команды требуется в среднем *S* шагов, а каждый шаг производится за один такт процессора. Если тактовая частота равна *R* тактам в секунду, то процессорное время выполнения программы составит

$$T_{проц} = \frac{N \times S}{R}$$

Для пользователя ЭВМ параметр $T_{проц}$ имеет гораздо большее значение, чем *N*, *S*, и *R*.

Очевидно, что $T_{проц}$ уменьшается с уменьшением *N* и *S* и увеличением *R*.

Количество команд объектной программы зависит от свойств компилятора. Величина *S* уменьшается, когда процесс выполнения команды состоит из меньшего количества шагов или некоторые шаги выполняются одновременно. Значение *R* растет с увеличением такто-вой частоты.

Важно отметить, что параметры *N, S,* и *R* зависят друг от друга, изменение одного из них может пов-лиять на величину другого.

Совсем не обязательно, что процессор с тактовой частотой 900 МГц будет иметь $T_{проц.}$ больше, чем процессор с тактовой частотой 1.2 ГГц, поскольку у последнего, например, для выполнения операции требуется большее количество тактов.

На быстродействие и производительность кроме *N, S,* и *R* влияют такие факторы, как, форма предс-тавления данных, пропускная способность интер-фей-сов, структура процессора, последовательность исполнения команд, свойства программного обеспе-чения и ещё множество других.

2.2. Представление данных

Данные, с которыми оперирует ЭВМ – быва-ют следующих типов:

числа, символы или строки символов, логические значения.

Числа могут быть представлены в форме

с фиксированной точкой (целые без знака, целые со знаком, правильные дроби со знаком или без знака) и

с плавающей точкой.

2.2.1. Числа в форме с фиксированной точкой.

Представление правильных дробей и целых чисел показано на рис. 2 и 3. Как правило, числа для представления чисел со знаком используют дополнительный код.

Основным недостатком формата с фиксированной точкой является небольшой диапазон представления чисел

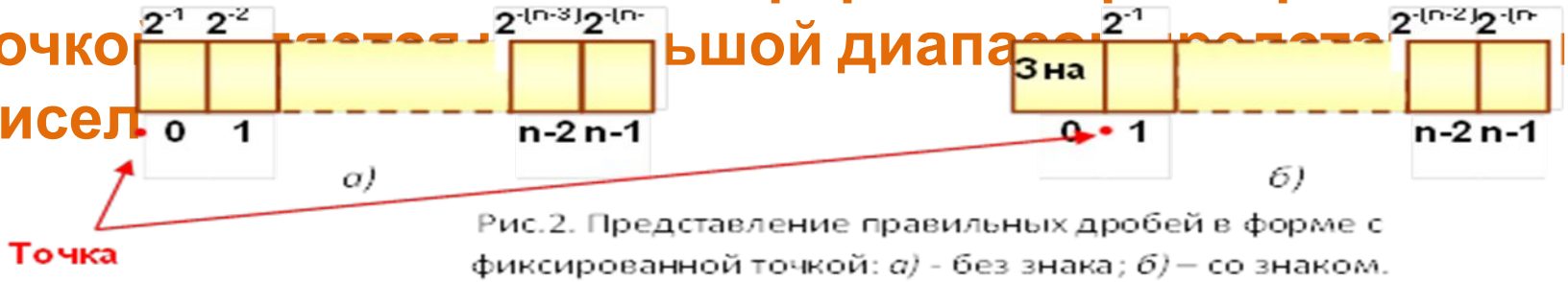


Рис.2. Представление правильных дробей в форме с фиксированной точкой: *а)* - без знака; *б)* – со знаком.

Рис.3. Представление целых чисел в форме с фиксированной точкой: *а)* - без знака; *б)* – со знаком.

2.2.2. Числа в форме с плавающей точкой

Форма представления с плавающей точкой, ещё называемая полулогарифмической, обеспе-чивает существенно больший диапазон предс-тавления.

Число представляется в виде произведения X = +m*q^±p, где m - мантисса числа X, p - порядок числа, q - основание системы счисления, а хранится в виде двух групп цифр - мантиссы и порядка.

На рис. 4 показаны диапазоны представления 32-раз-рядных целых чисел и с плавающей точ-кой (мантисса - 24

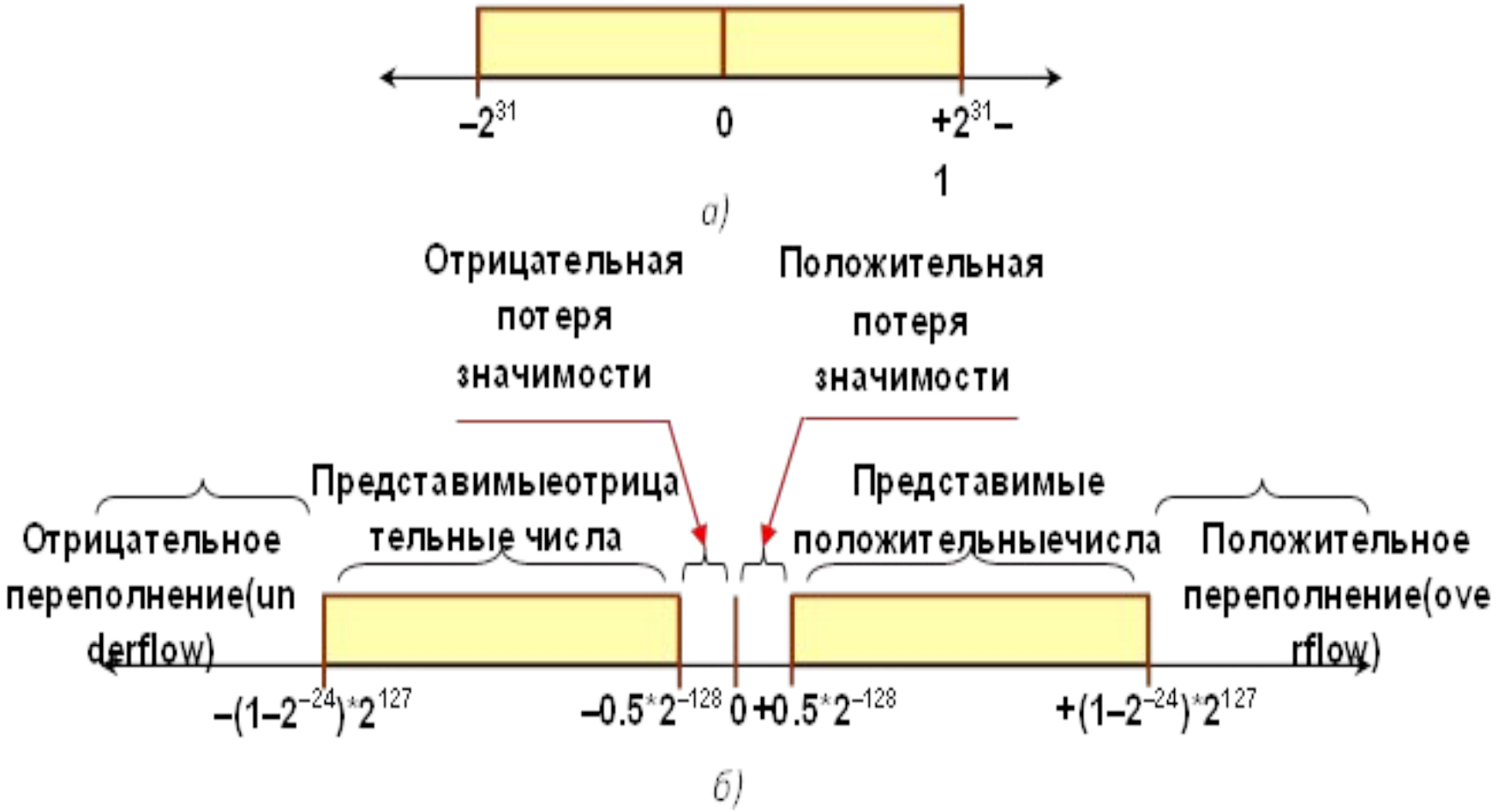


Рис.4. Диапазоны представления чисел в 32-разрядных форматах: *а)* целые числа с фиксированной точкой; *б)* с плавающей точкой.

Стандарт IEEE 754 для представления чисел с пла-вающей точкой в 32 – разрядном формате разработан и детально специфицирован Институтом инженеров по электротехнике и электронике (Institute of Electrical and Electronics Engineers, IEEE).

Стандарт определяет 32-битовый (с одинарной точ-ностью) и 64-битовый (с двойной точностью) форматы (рис. 5) с 8- и 11-разрядным порядком соответственно. Основанием системы счисления является 2.

В дополнение, стандарт предусматривает два рас-ширенных формата, одинарный и двойной, фактичес-кий вид которых зависит от конкретной реализации.

Расширенные форматы предусматривают дополни-тельные биты для порядка

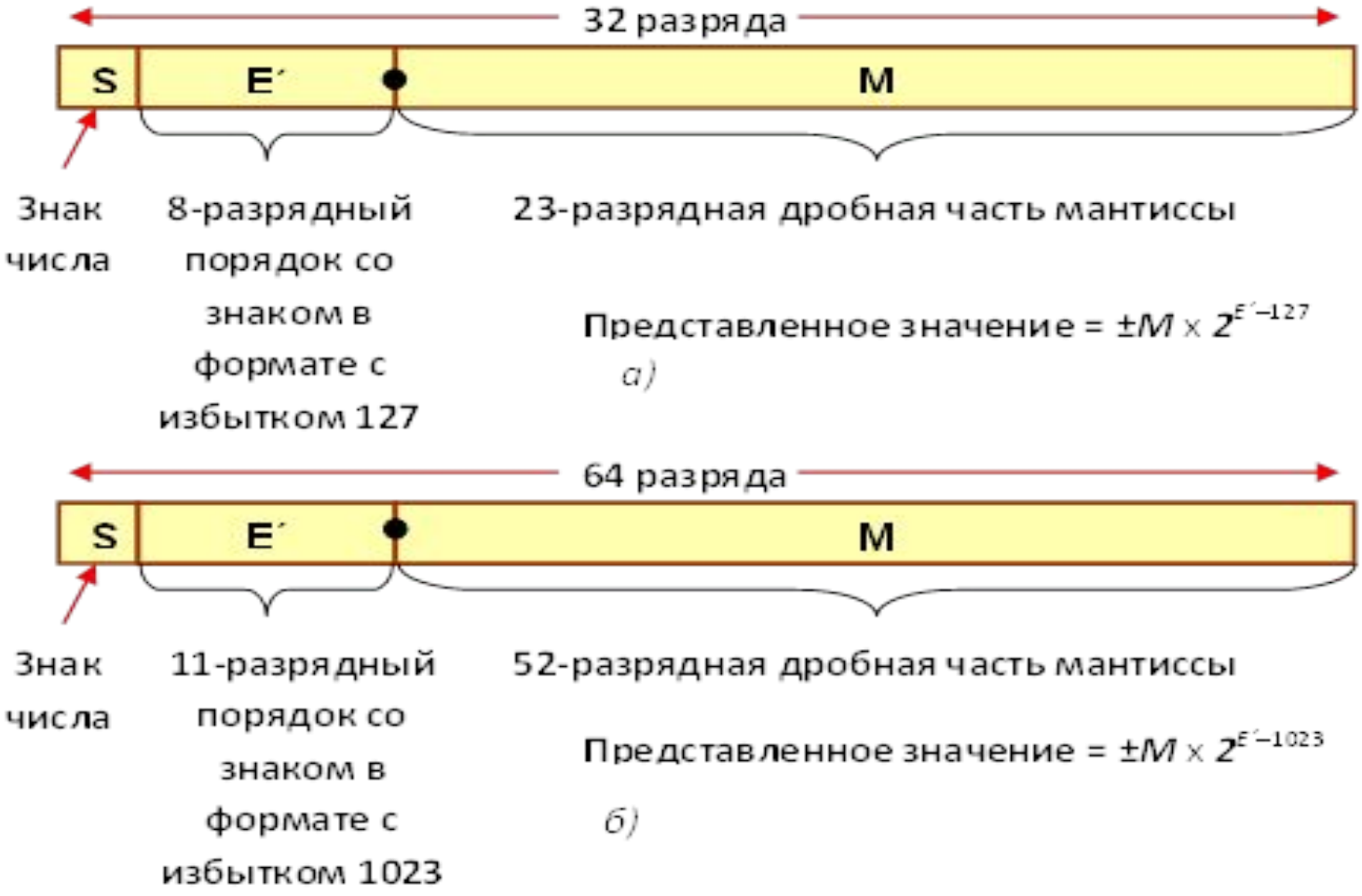


Рис.5 Представление чисел с плавающей точкой в соответствии со стандартом IEEE 754: *а)* с одинарной точностью; *б)* с двойной точностью.

Особенностью представления чисел в формате IEEE является следующее. Порядки смещены в область положительных чисел, могут принимать значения в диапазоне от 1 до 254 для одинарного формата и от 1 до 2036 - для двойного формата, и используются для представления ненулевых нормализованных чисел. (Это значит, что порядки – целые числа без знака)

Представляемые числа хранятся в нормализованном виде. Нормализованное число требует, чтобы слева от двоичной точки был единичный бит. Поскольку этот бит всегда равен 1, то он в явном виде не присутствует, а подразумевается.

Благодаря этому обеспечивается эффективная ширина ман-тиссы, равная 24 битам для одинарного и 53 битам – для двойного форматов.

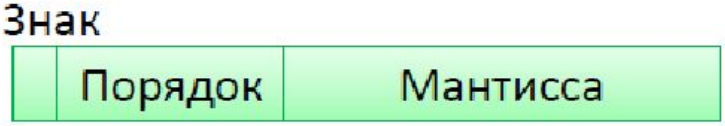
Сказанное выше можно проиллюстрировать следующим при-мером.

Ненормализованное число +0.0010110…x29 в нормализован-ном виде выглядит как +1.0110…x26, а нормализованное и в формате IEEE как 010000101.0110.

В примере первый 0 – это знак; далее 8 цифр (до точки) – смещённый порядок; после точки – само число (точнее мантисса) с учётом подразумеваемой единицы. Точка

Представление чисел в IEEE 754

- Стандарт определяет 4 формата представления чисел



Single precision	32 бита	
Double precision	64 бита	
Single Extended precision	≥ 43 бит	Редко используется
Double Extended Precision	≥ 79 бит	Обычно 80

Знак	Знак числа 0 – “+”, 1 – “-”
Порядок	Порядок числа плюс смещение (константа). Прибавление смещения позволяет записывать положительные и отрицательные порядки в виде положительных чисел.
Мантисса	Мантисса числа. При этом число, по-возможности, нормализуется, т.е. мантисса приводится к диапазону [1,2[. Тогда старший бит не пишется.

Пример записи числа с плавающей точкой

- Рассмотрим число +178.25
- В двоичной системе +1011 0010.01 = +1.**0110 0100 1**×2^{111}

Single Precision (32 бита, смещение порядка: $127_{(10)}$=111 $1111_{(2)}$)		
Знак	1 бит	**0**
Порядок	8 бит	**111**+111 1111=**1000 0110**
Мантисса	23 бита	**011 0010 0100 0000 0000 0000**
Итого	32 бита	**0100 0011 0**011 0010 0100 0000 0000 0000
В 16-ричном формате		**43324000**
Double Precision (64 бита, смещение порядка: $1023_{(10)}$ = 11 $1111\ 1111_{(2)}$)		
Знак	1 бит	**0**
Порядок	11 бит	**111**+11 1111 1111=**100 0000 0110**
Мантисса	52 бита	**0110 0100 1000 0000 0000 0000 000 …**
Итого	64 бита	**0100 0000 0110** 0110 0100 1000 0000 … 0000
В 16-ричном формате		**40664800 00000000**

Благодаря этому обеспечивается эффективная ши-рина мантиссы, равная 24 битам для одинарного и 53 битам – для двойного форматов.

Сказанное выше можно проиллюстрировать следующим примером.

Ненормализованное число +0.0010110…x29 в нормализованном виде выглядит как +1.0110…x26,

а нормализованное и в формате IEEE как 010000101.0110.

В примере первый 0 – это знак; далее 8 цифр (до точки) – смещённый порядок; после точки – само число (точнее мантисса) с учётом подразумеваемой единицы. Точка поставлена

2.2.3. Символы

ЭВМ способны обрабатывать не только числа, но и текстовую информацию, состоящую из сим-волов.

Под термином символы подразумеваются буквы латинские, греческие, кириллицы, деся-тичные цифры, знаки препинания, иероглифы, символы математических операций и так далее.

Каждому символу ставится в соответствие определенная двоичная комбинация. Совокуп-ность возможных символов и назначенных им двоичных

Наиболее распространенными являются кодо-вые таблицы, в которых символы кодируются с помощью восьмиразрядных двоичных комби-наций (байтов), позволяющих представить 256 различных символов:

1. **Расширенный двоично-кодированный код EBCDIC (Extended Binary Coded Decimal Interchange Code); известный ещё под названием ДКОИ**

(Двоичный Код для Обработки Информации)

2. **Американский стандартный код для обмена информацией ASCII (American Standard Code for Information Interchange).**

Стандартный код ASCII – 7-разрядный. В более поздней, европейской модификации ASCII (стан-дарт ISO 8859-1) используются все 8 разрядов.

Дополнительные комбинации (коды 128-255) в новом варианте отводятся для представления специфических букв алфавитов западно-европей-ских языков, символов псевдографики, некото-рых букв греческого алфавита, а также ряда мате-матических и финансовых символов.

Именно эти кодовые таблицы считаются миро-вым стандартом де-факто, который в различных модификациях применяется во всех странах.

В зависимости от использования кодов 128-255 различают несколько вариантов

2.2.4. Логические значения

Элементом логических данных является логическая (булева) переменная, которая принимает значения: «истина» или «ложь».

Единицей кодируют истинное значение, нулем — ложное. Как правило, в ЭВМ опери-руют наборами логических переменных дли-ной в машинное слово.

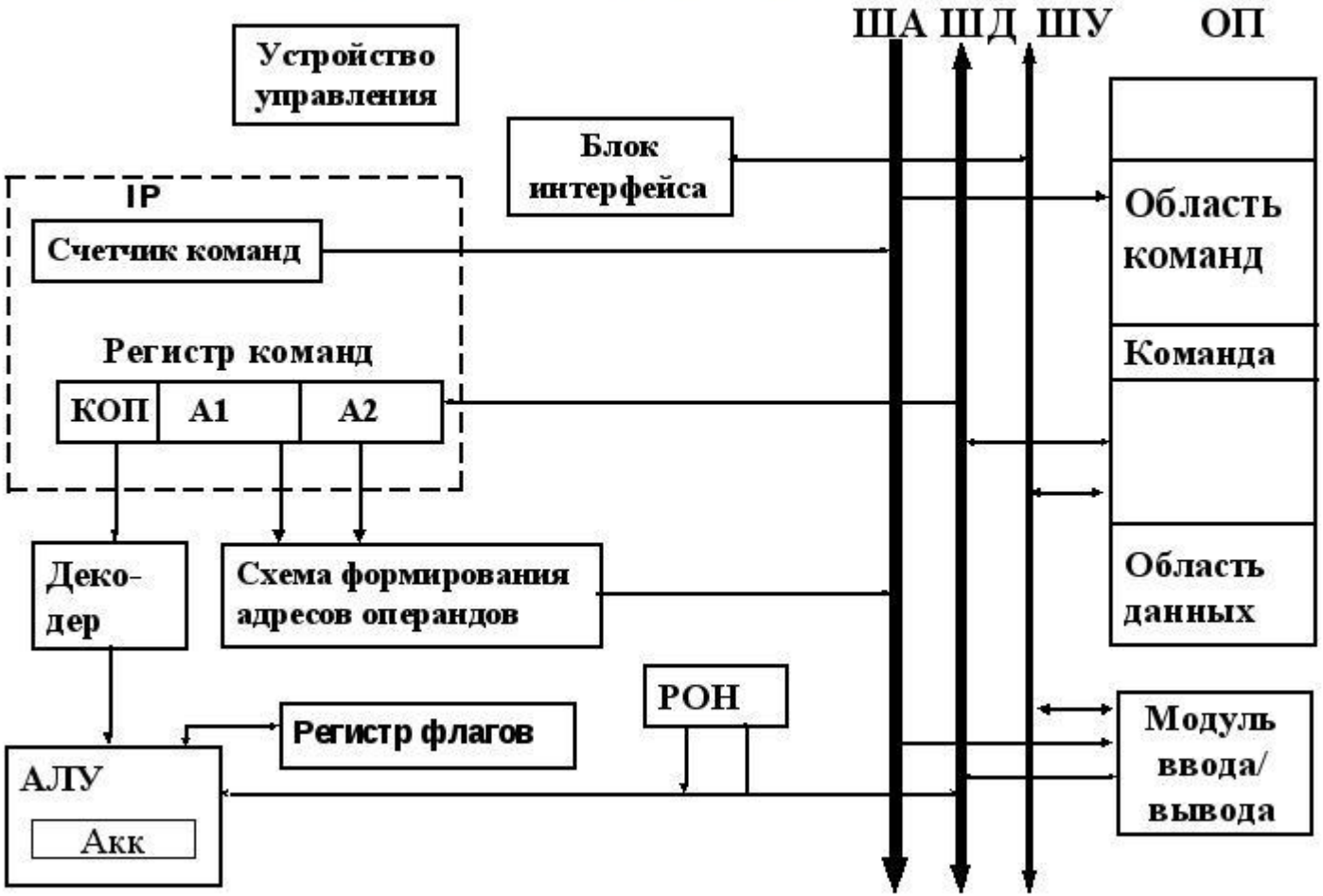
Обрабатываются такие слова с помощью команд логических операций (И, ИЛИ, НЕ и т. д.), при этом все биты обрабатываются одина-ково, но независимо друг от друга, то есть ника-ких переносов между разрядами не возникает.

2.3. Основные концепции функционирования

На примере простейшего гипотетического компьютера (Рис.6) рассмотрим его состав и не-которые аспекты функционирования.

В состав любого процессора входят указатель (счётчик) команд (IP), регистр команд (РК), регистр адреса (РгАП), арифметико – логичес-кое устройство (АЛУ), набор регистров в нашем случае это набор регистров общего назначения (РОН 0 - РОНn).

Обобщенная структура процессора



Цикл процессора

Цикл процессора - **процесс обработки каждой команды, состоящий из двух этапов: выборка и исполнение.**

Выборка команды.

Когда устройство управления завершит выполнение текущей команды, оно должно выбрать следующую команду из памяти в Регистр команд (РгК).

Адрес следующей команды содержится в специальном регистре, называемом Счетчиком Команд (СчК).

Всякий раз при выборке команды устройство управления одновременно увеличивает содержимое СчК на единицу, чтобы после выполнения текущей команды можно было произвести выборку следующей.

Таким образом, устройство управления работает с коман-дами в порядке, в котором они помещены в ОЗУ. Подобная ситуация иллюстрируется на рис. 5.8.

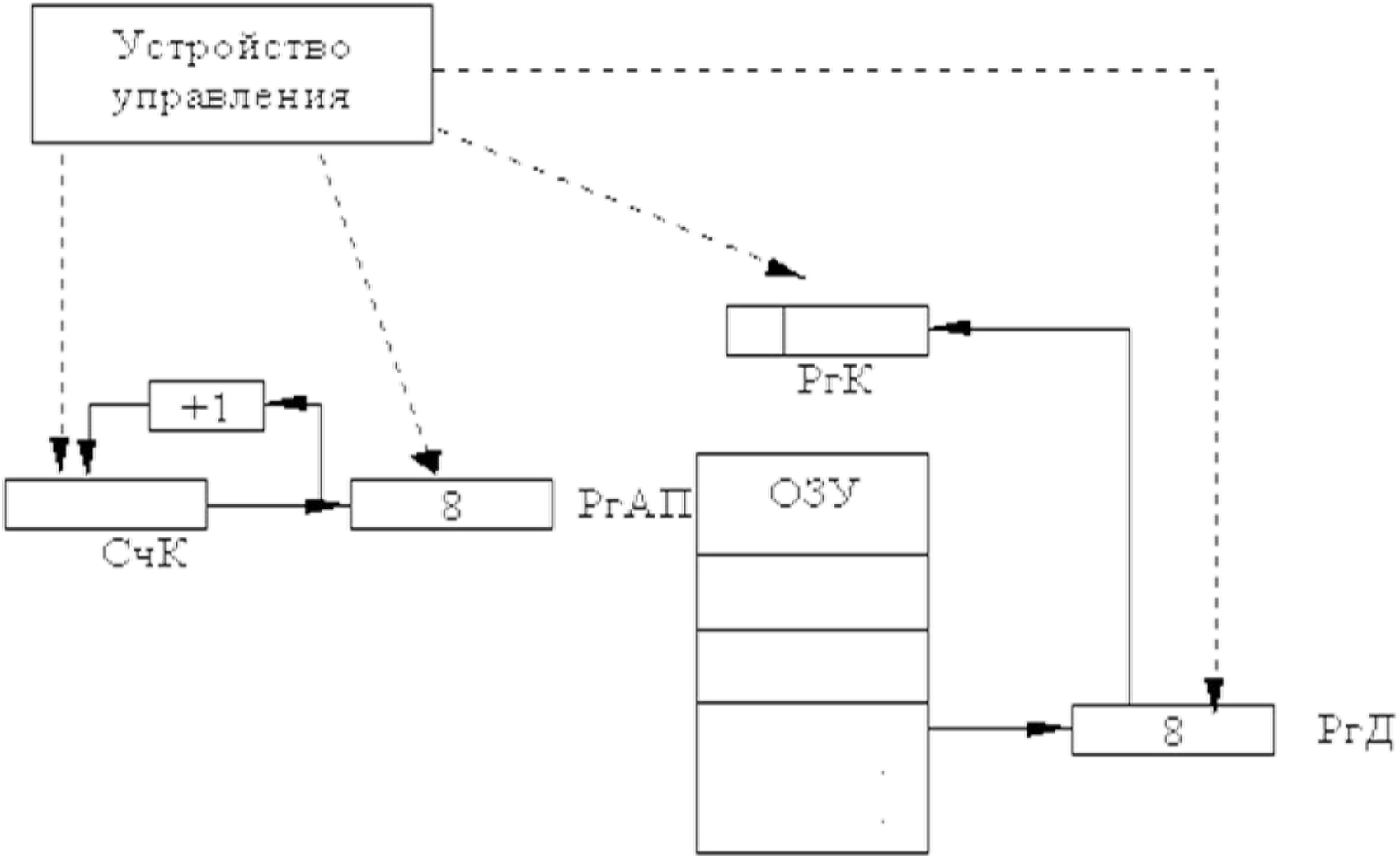


Рис. 5.8. Структура цепей выборки команды

Последовательность выборки команды:

- а) копируется адрес следующей команды из СчК в Регистр адреса памяти (РгАП) (адрес 8 в примере на рис.5.8);
- б) прибавляется 1 к содержимому СчК СчК:=СчК+1 (9);
- в) содержимое ячейки (команда), которая адресуется РгАП, пересылается в Регистр данных (РгД);
- г) содержимое РгД копируется в Регистр команд (РгК).

Этим выборка завершается.

Исполнение команды

Взаимодействие блоков процессора при выполнении команды (например: СЛОЖЕНИЕ) схематично показано на рис. 5.7.

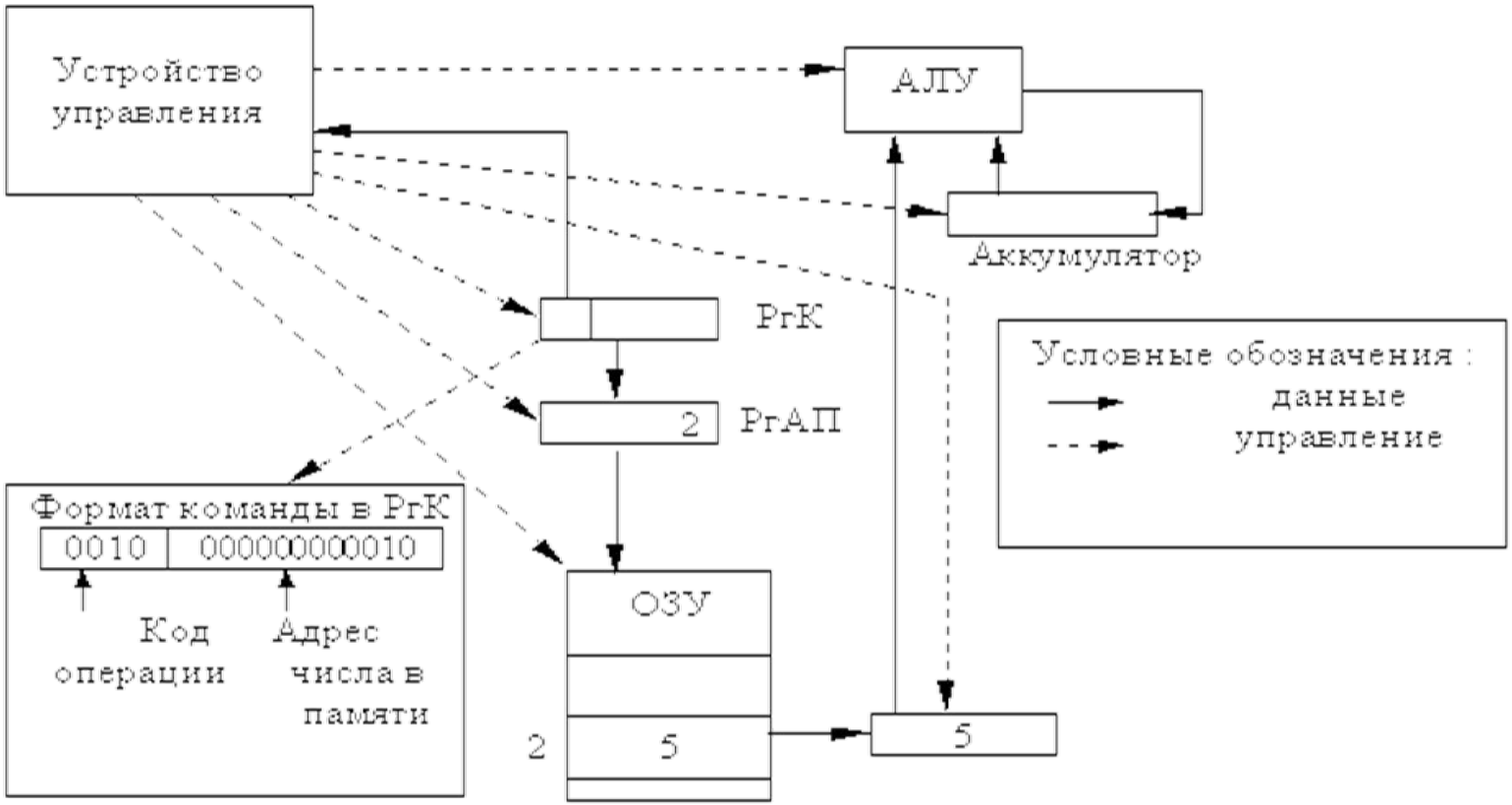


Рис. 5.7. Взаимодействие блоков процессора при выполнении команды СЛОЖЕНИЕ.

Для временного хранения машинной команды используется специальный регистр, содержимое которого интерпретируется как команда - Регистр Команды (РгК).

Команда, записанная в РгК, имеет две части: функциональную и адресную

Последовательность выполнения команды:

- **а) устройство управления декодирует функциональную часть команды, интерпретируя ее как операцию сложения;**
- **б) адрес операнда из адресной части РгК пересылается в РгАП;**

- в) устройство управления инициирует чте-ние операнда из ячейки, адрес которой находит-ся в РгАП, и загрузку операнда в РгД.

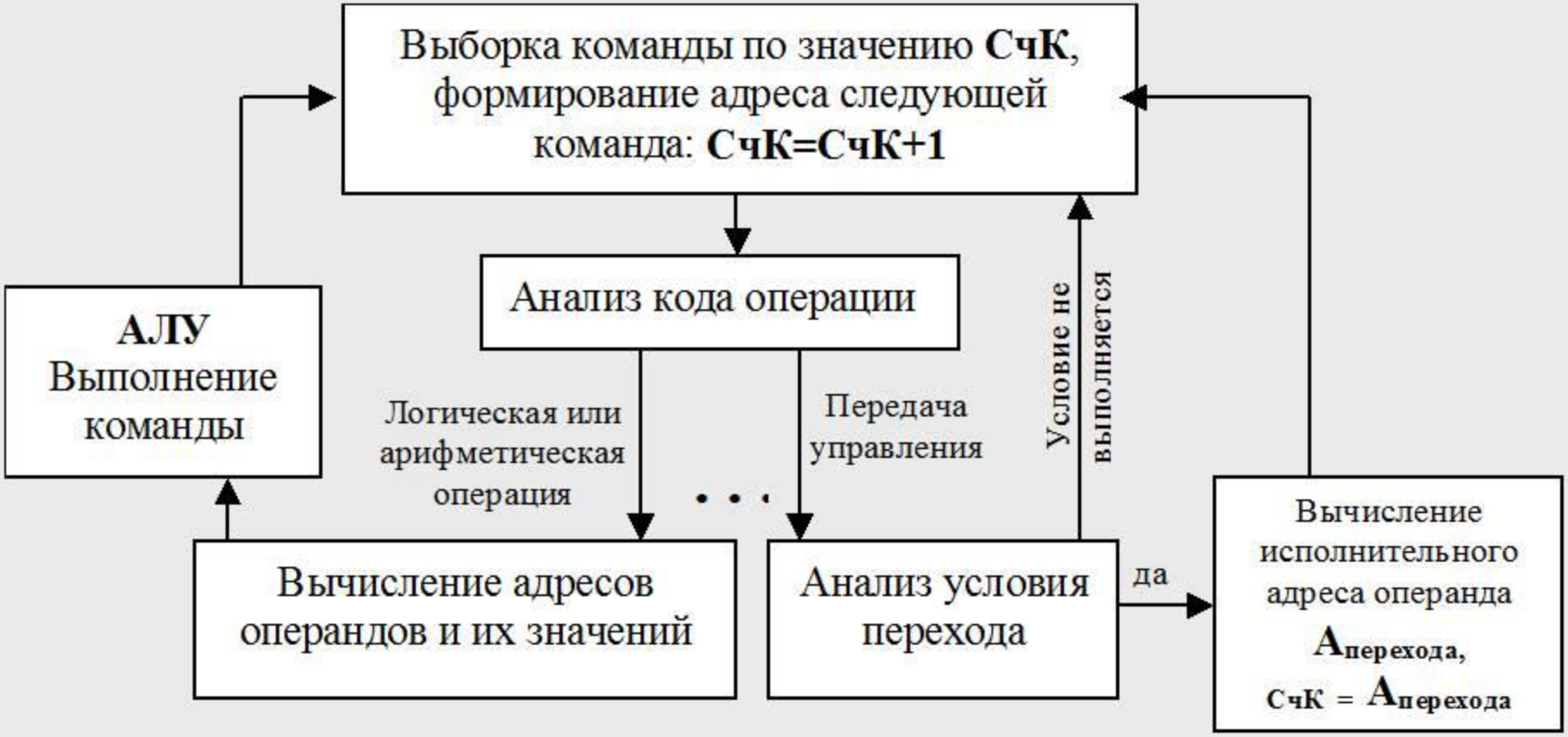
Таким образом, 5 из второй ячейки оказывает-ся в регистре данных;

- г) АЛУ выполняет операцию (сложение) над содержимым РгД и аккумулятора. А результат по сигналу устройства управления будет занесен в аккумулятор. Таким образом, к концу команды сложения содержимое аккумулятора равно 11.

Этим завершается выполнение

Центральный процессор

Рабочий цикл процессора



В рассмотренном примере предполагается последовательное исполнение команд и пос-ледовательное исполнение отдельных этапов команд.

Это так называемая последовательная или фон-неймановская архитектура.

Однако процесс вычислений можно орга-ни-зовать с той или иной степенью паралле-лизма.

В зависимости от степени распараллелива-ния процессов различают четыре типа архи-тектур (так

1. SISD - Single Instruction Single Data (ОКОД - Одиночный поток Команд Одиночный поток Данных); без использования параллелизма.

2. SIMD - Single Instruction Multiple Data (ОКМД - Одиночный поток Команд Множест-венный поток Данных); несколько процессо-ров по одному алгоритму (одной команде) обрабатывают одновременно несколько пото-ков данных.

Это класс, так называемых, потоковых процессоров.

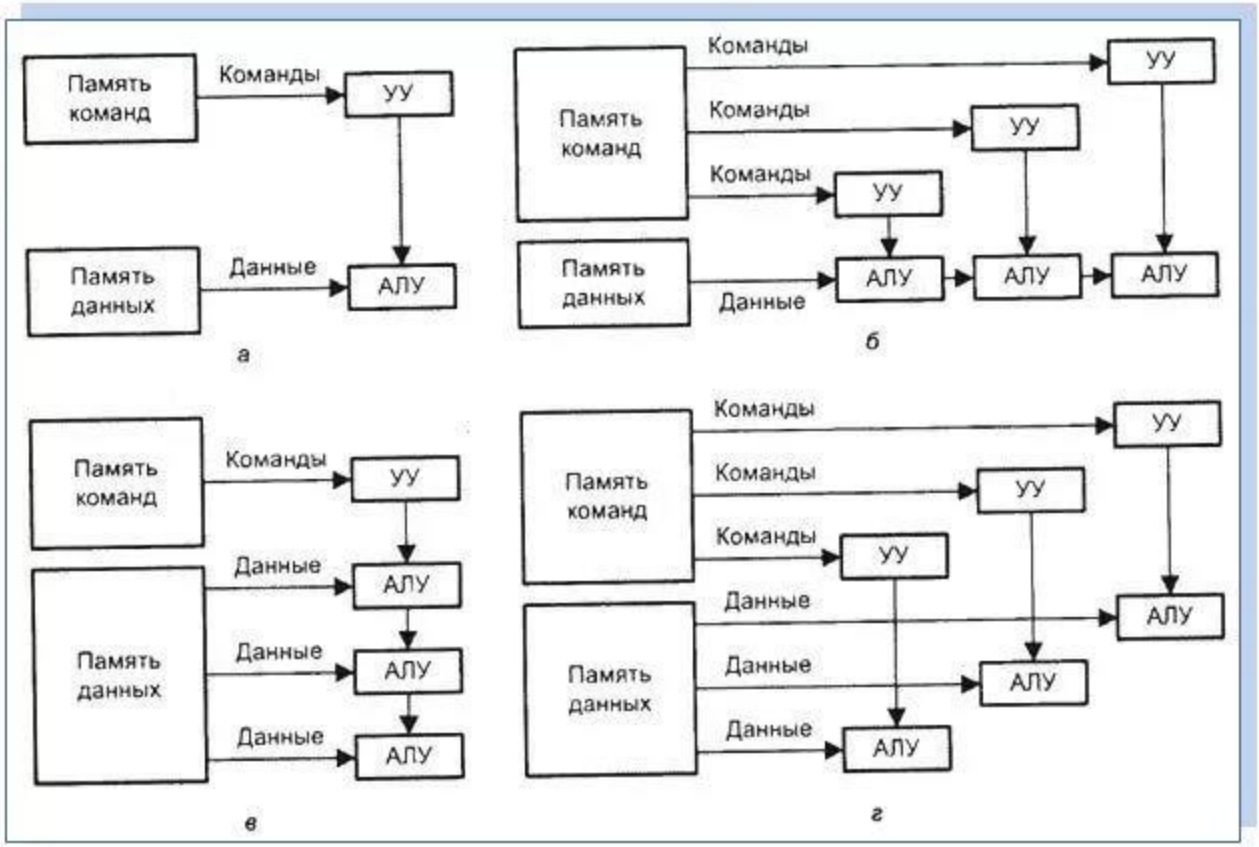
3. MISD - Multiple Instruction Single Data (МКОД - Множественный поток Команд Одиночный поток Данных); конвейерная обработка, когда одновременно на вход поступает один поток дан-ных (одно данное), но он последовательно обра-батывается большим количеством процессоров различного функционального назначения.

4. MIMD - Multiple Instruction Multiple Data (МКМД - Множественный поток Команд Мно-жественный поток Данных); самый сложный случай, когда используется и конвейерная, и па-раллельная обработки.

Параллелизм как основа высокопроизводительных вычислений

Классификация параллельных вычислительных систем

Классификация Флинна. MISD



Архитектура вычислительных систем по Флинну: а — SISD; б —MISD; в-SIMD; г-MIMD

2.4. Структуры АЛУ

Арифметическая и логическая обработка дан-ных в ЭВМ возлагается на операционный блок, а точнее на арифметико-логическое устройство (АЛУ).

Количество различных операций и типов обра-батываемых данных достаточно велико, поэтому АЛУ можно построить либо как одно универсаль-ное устройство, либо как набор специализирован-ных устройств.

Каждое из устройств реализует определенное подмножество арифметических или логических операций, предусмотренных системой команд ЭВМ.

Операционный блок, в свою очередь, может быть построен по схеме с закреплением микро-операций по регистрам либо с магистральной структурой.

В первом случае за каждым регистром опера-ционного блока жёстко закреплены определён-ные функции. Например, регистр результата или аккумулятор, регистр множителя – частного.

Достаточно всего трёх регистров, чтобы пост-роить такой операционный блок.

Вариант ОБ с магистральной структурой пока-зан на рис. 7. Возможны различные варианты организации подобных блоков: 3-х шинные, 2-х шинные, с одной шиной.

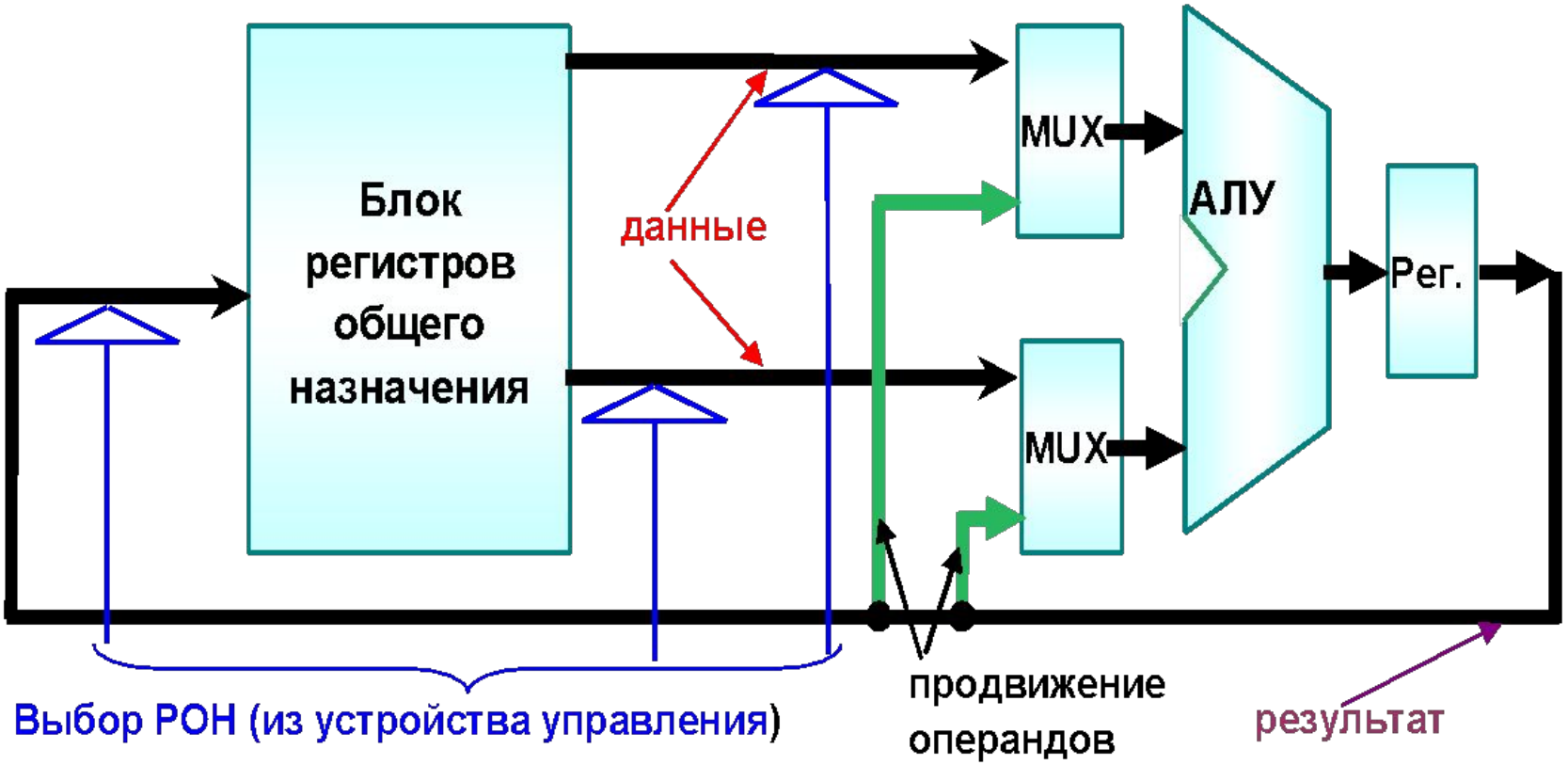


Рис.7.Операционный блок с 3-х шинной магистралью.

2.5. CISC и RISC и другие процессоры

В зависимости от набора и порядка выполнения команд процессоры можно разделить на:

1. Complex Instruction Set Comand (CISC) – процессор со сложным (полным) набором команд.

2. Reduced Instruction Set Comand (RISC) – процессор с сокращенным набором команд.

3. Minimum Instruction Set Comand (MISC) – процессор с минимальным набором команд.
Архитектура MISC строится на стековой вычислительной модели с ограниченным числом команд (примерно 20–30 команд).

4. Very long instruction word (VLIW) – процессор с очень длинным командным словом — архитектура с нес-колькими АЛУ. В одной инструкции процессора задаётся несколько операций, которые должны вы-полняться параллельно.

Перед разработчиками системного програм-много обеспечения и создателями аппаратуры компьютеров всегда стояла

проблема определения количества и перечня инструкций процессора.

Программистам необходимо иметь в своем распоряжении как можно больше разнообраз-ных команд, чтобы повысить эффективность компиляторов и операционных систем,

а для Разработчиков аппаратуры каждая новая инструкция – это лишняя проблема.

В то же время известно, что наиболее простой способ достижения высокой скорости выполнения программ заключается

в переносе наиболее частых алгоритмических действий в аппаратуру процессора.

Поэтому по мере развития технологии интегральных схем

разработчики аппаратуры

постепенно уступали давлению программистов, добавляя новые инструкции в систему команд.

В итоге список команд типичного компьютера расширился от нескольких десятков до нескольких сотен.

Благодаря этому удалось максимально упрос-тить компиляцию программ и заодно минимизи-ровать размер исполняемого модуля – а это еще один эффективный способ увеличения производи-тельности, поскольку компактную программу про-ще разместить в кэше инструкций и за счет этого уменьшить количество обращений к оперативной памяти.

Так сформировалась стратегия архитектуры CISC (Complex Instruction Set Comand – компьютер с комплексным набором команд), которую образно можно представить как перенос "центра тяжести" обработки с программного уровня на

Однако для микропроцессоров идеология CISC стала серьезным препятствием в повышении их быстродействия.

Наиболее критическим фактором для сверх-больших интегральных схем, которыми по су-ществу и являются микропроцессоры, является площадь кристалла.

Расширенный набор команд требует значи-тельного объема оборудования устройства уп-равления и, при ограниченной площади крис-талла, не хватает места для размещения арифме-тических устройств, необходимых для

Проведённые в конце 70–х годов исследования были обобщены в виде правила «80/20», которое гласит, что в типовом случае 80% кода программы использует всего 20% простейших команд формата «регистр–регистр» полного набора инструкций CISC.

При этом значительно снижается время реше-ния задач вычислительного типа и сильно упрощается устройство управления процессора, что приводит к заметному уменьшению площади кристалла.

Так возникла стратегия RISC: – обеспечить рост производительности с помощью высокой скорости выполнения большого числа простых операций («длинная программа – короткие команды»), а не путем уменьшения количества команд CISC–програм-мы («короткая программа –

Основные принципы RISC заключаются в следую-щем:

1. **Любая операция, вне зависимости от ее типа, должна выполняться за один такт.**
2. **Система команд должна содержать минималь-ное количество наиболее часто используемых прос-тейших инструкций одинаковой длины.**
3. **Операции обработки данных реализуются только в формате «регистр–регистр».**

Обмен между оперативными регистрами и па-мятью (модификация переменных в памяти) выпол-няется только с помощью команд загрузки/записи.

4. **Состав системы команд должен быть «удобен» для компиляции операторов языков**

RISC–процессоры обязательно должны иметь конвейеризованные арифметические устройства.

Современные технологические возможности в сфере проектирования и производства БИС поз-волили существенно смягчить ограничение соста-ва команд. Вместо нескольких десятков инструк-ций, использовавшихся в приборах первого поко-ления, современные RISC–процессоры реализуют более сотни инструкций.

Однако основной закон RISC был и остается незыблемым: обработка данных выполняется только в рамках регистровой структуры процес-сора без обращения к

Основные особенности современных RISC - процессоров:

1. Сокращенный набор команд (от 80 до 150 команд).
2. Большинство команд выполняется за 1 такт.
3. Большое количество регистров общего назначе-ния.
4. Наличие жестких многоступенчатых конвейеров.
5. Все команды имеют простой формат, и использу-ют небольшое количество способов адресации.
6. Наличие вместительной раздельной кэш-памяти.
7. Применение оптимизирующих

Практически все современные RISC – процессоры

- **Являются 64-х разрядными и суперскаляр-ными (запускаются не менее 4-х команд за такт).**
- **Имеют встроенные конвейерные блоки арифметики с плавающей точкой.**
- **Имеют многоуровневую кэш-память.**

Большинство RISC–процессоров кэшируют предварительно

Одной из причин появления архитектуры RISC является относительная простота устройства управ-ления процессора.

Однако по мере развития некоторые микро-процессоры с RISC – архитектурой по сложности догнали и перегнали CISC - процессоры.

Принцип простоты, изначальный для RISC - процессоров сохраняется в архитектуре MISC (Minimum Instruction Set Comand), процессор с минимальным набором команд.

Архитектура MISC строится на стековой вычис-лительной модели с ограниченным количеством команд (примерно 20–30 команд).

Другой ветвью развития архитектуры RISC является архитектура VLIW (Very long instruction word, очень длинная машинная команда) – архитектура процессоров с несколь-кими вычислительными устройствами или линиями кон-вейера.

Характеризуется тем, что одна инструкция процессора содержит несколько операций, которые должны выпол-няться параллельно.

Так же, как в RISC, в инструкции VLIW явно указывается, что именно должен делать каждый модуль процессора.

Из–за этого длина инструкции может достигать 128 или даже 256 бит.

В ранних моделях суперскалярных процессоров также есть несколько вычислительных модулей, но в ранних моделях таких процессоров задача распределения работы между модулями решалась аппаратно.

Это сильно усложняет структуру процессора, и может быть чревато ошибками.

2.6. Матричные процессоры

Наиболее распространенными из систем, клас-са: один поток команд - множество - потоков данных (SIMD), являются матричные системы, которые лучше всего приспособлены для реше-ния задач, характеризующихся параллелизмом независимых объектов или данных.

Организация систем подобного типа на первый взгляд достаточно проста.

Они имеют общее управляющее устройство, генерирующее поток команд и большое число процессорных элементов, работающих парал-лельно и обрабатывающих каждая свой поток данных.

Таким образом, производительность сис-темы оказывается равной

сумме производительностей всех процессорных элементов.

Однако на практике, чтобы обеспечить достаточную эффективность системы при решении широкого круга задач необходимо организовать связи между процессорными элементами с тем, чтобы наиболее полно загрузить их работой.

Именно характер связей между процес-сорными элементами и

Одним из первых матричных процессоров был SOLOMON (60-е годы).

Рис. 2.1 Структура матричной вычислительной системы "SOLOMON"

Система SOLOMON содержит 1024 процессорных элемента, соединены в виде матрицы: 32x32. Каждый процессорный элемент матрицы включает в себя про-цессор, обеспечивающий выполнение последовательных поразрядных арифметических и логических операций, а также оперативное ЗУ, емкостью 16 Кбайт.

Длина слова - переменная от 1 до 128 разрядов. Разрядность слов устанавливается программно.

По каналам связи от устройства управления передают-ся команды и общие константы.

В процессорном элементе используется, так называе-мая, многомодальная логика, которая позволяет каждому процессорному элементу выполнять или не выполнять общую операцию в зависимости от значений обрабатыва-емых данных.

В каждый момент все активные процессорные элемен-ты выполняют одну и ту же операцию над данными, хра-нящимися в собственной памяти и

Идея многомодальности заключается в том, что в каждом процессорном элементе имеется специальный регистр на 4 состояния - регистр моды.

1. Мода (модальность) заносится в этот регистр от устройства управления.

2. При выполнении последовательности ко-манд модальность передается в коде операции и сравнивается с содержимым регистра моды.

3. Если есть совпадения, то операция выполня-ется.

В других случаях процессорный элемент не выполняет операцию, но может, в зависимос-ти от кода, пересылать свои операнды соседнему процессорному элементу.

Такой механизм позволяет выделить стро-ку или **столбец процессорных элементов, что очень полезно при операциях над матрицами.**

Взаимодействуют процессорные элементы с периферийным оборудованием через внеш-ний процессор.

2.7. Многоядерные процессоры

Общее понятие о ядре процессора

Если сам процессор — это мозг компьютера, то его ядро — это мозг самого процессора.

Ядро процессора выполняет все арифметические и логические операции, а также содержит все необходи-мые функциональные блоки, среди которых:

- **Блок работы с прерываниями — это, попросту говоря, возможность быстро и часто переключаться с выполнения одной задачи на другую.**
- **Блок выборки инструкций — получает и направ-ляет на дальнейшую обработку сигналы команд.**

- **Блок декодирования** — обрабатывает сигналы команд, определяет, что нужно сделать в данный момент, и нужны ли для этого дополнительные дей-ствия.
- **Управляющий блок** — передает декодированные инструкции для дальнейшего вы-полнения в дру-гие блоки, координирует нагрузку, подаваемую на них.
- **Блоки выполнения и сохранения результатов** со-ответственно выполняют полученную команду и сох-раняют в нужном месте результат.

Это краткое описание структуры ядра.

В разных процессорах может быть разное количес-тво ядер. Это делается для того, чтобы компьютер мог выполнять параллельно несколько однотипных или напротив, разноплановых задач, увеличивая скорость их обработки и, соответственно, скорость их

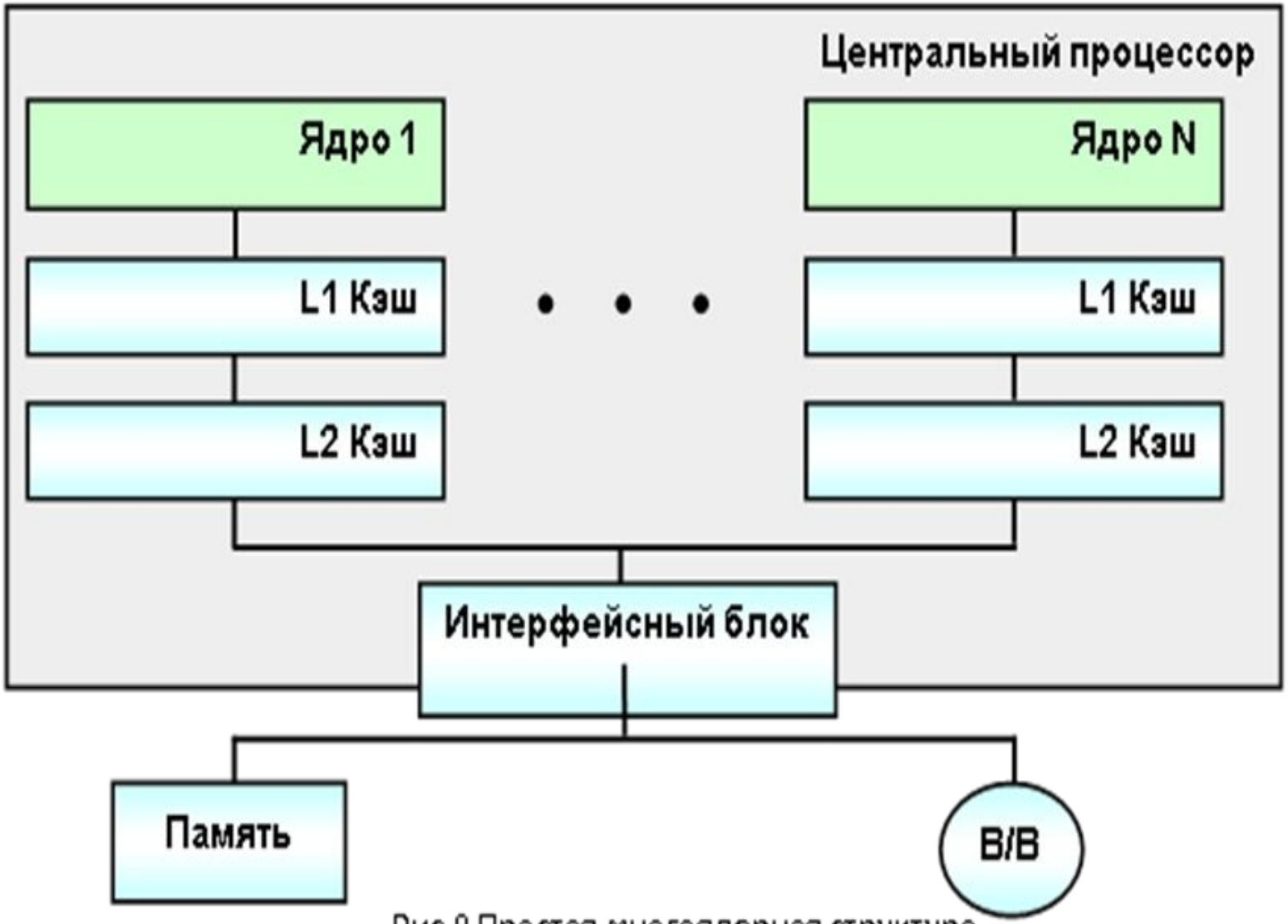


Рис.8 Простая многоядерная структура.

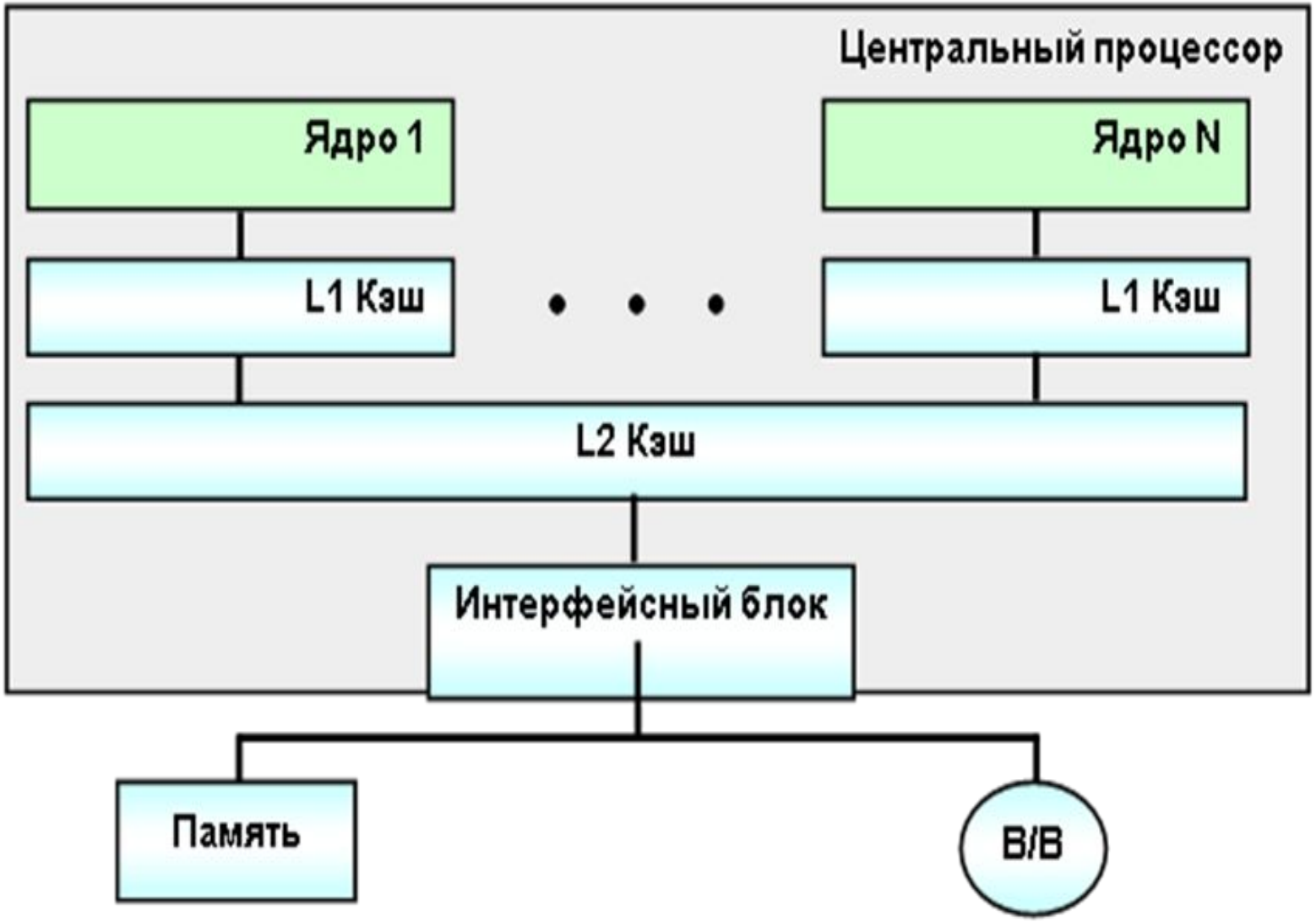


Рис. 9 Многоядерная структура с общей кэш-памятью.

Преимущества многоядерных процессоров состоят в следующем.

1. **Простота (естественно относительная) проектирова-ния и производства. Разработав одно эффективное ядро, его можно тиражировать в кристалле, дополняя архитекту-ру нужными системными компонентами.**
2. **Заметно уменьшается энергопотребление. Если, к примеру, на кристалле разместить два ядра и заставить их работать на тактовой частоте, обеспечивающей производи-тельность равную производительности, одноядерного «собрата», а потом сравнить энергопотребление обоих, то обнаружится, что энергопотребление уменьшается в нес-колько раз, поскольку оно растет почти пропорционально квадрату частоты.**

В целом же, если внимательно посмотреть на рисунки 8 и 9, можно увидеть, что принципиальной разницы между, скажем, 2-х процессорной системой и ЭВМ на 2-х ядерном процессоре нет. Проблемы одинаковые. И одна из первых – соответствующая