Организация компьютерных систем

Процессоры

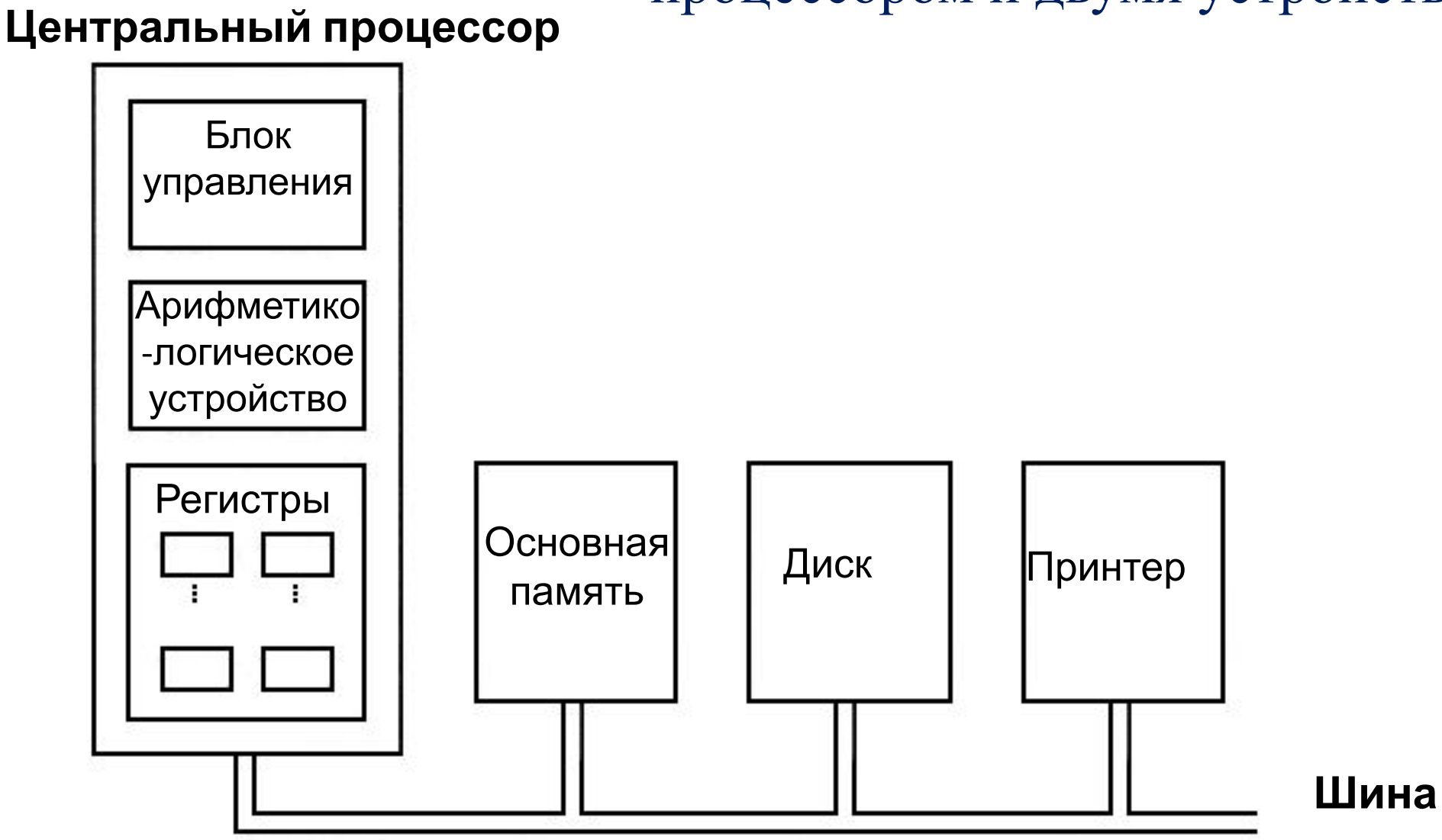
- **Центральный процессор** — это мозг компьютера.

Основная задача: выполнять программы, находящиеся в основной памяти.

Для этого ЦП вызывает команды из памяти, определяет их тип, а затем выполняет одну за другой.

- Компоненты ЦП соединены **шиной,** представляющей собой набор параллельно связанных проводов для передачи адресов, данных и управляющих сигналов.
- Шины могут быть внешними (связывающими процессор с памятью и устройствами ввода-вывода) и внутренними. Современный компьютер использует несколько шин.

Схема компьютера с одним центральным процессором и двумя устройствами ввода-вывода



Центральный процессор

- **Блок управления** отвечает за вызов команд из памяти и определение их типа.
- **Арифметико-логическое устройство** выполняет арифметические операции (например, сложение) и логические операции (например, логическое И).
- **Быстрая память** небольшого объема служит для хранения промежуточных результатов и некоторых команд управления, состоит из нескольких регистров, каждый из которых выполняет определенную функцию.

Регистры ЦП

- Обычно имеют одинаковый размер.
- Операции чтения и записи с регистрами выполняются очень быстро, поскольку они находятся внутри центрального процессора.
- Самый важный регистр — **счетчик команд,** который указывает, какую команду нужно выполнять следующей.
- В **регистре команд** находится выполняемая в данный момент команда.
- У большинства компьютеров имеются и другие регистры, одни из них многофункциональны, другие служат лишь какие-либо конкретным целям. Третьи регистры используются операционной системой для управления компьютером.

Устройство центрального процессора

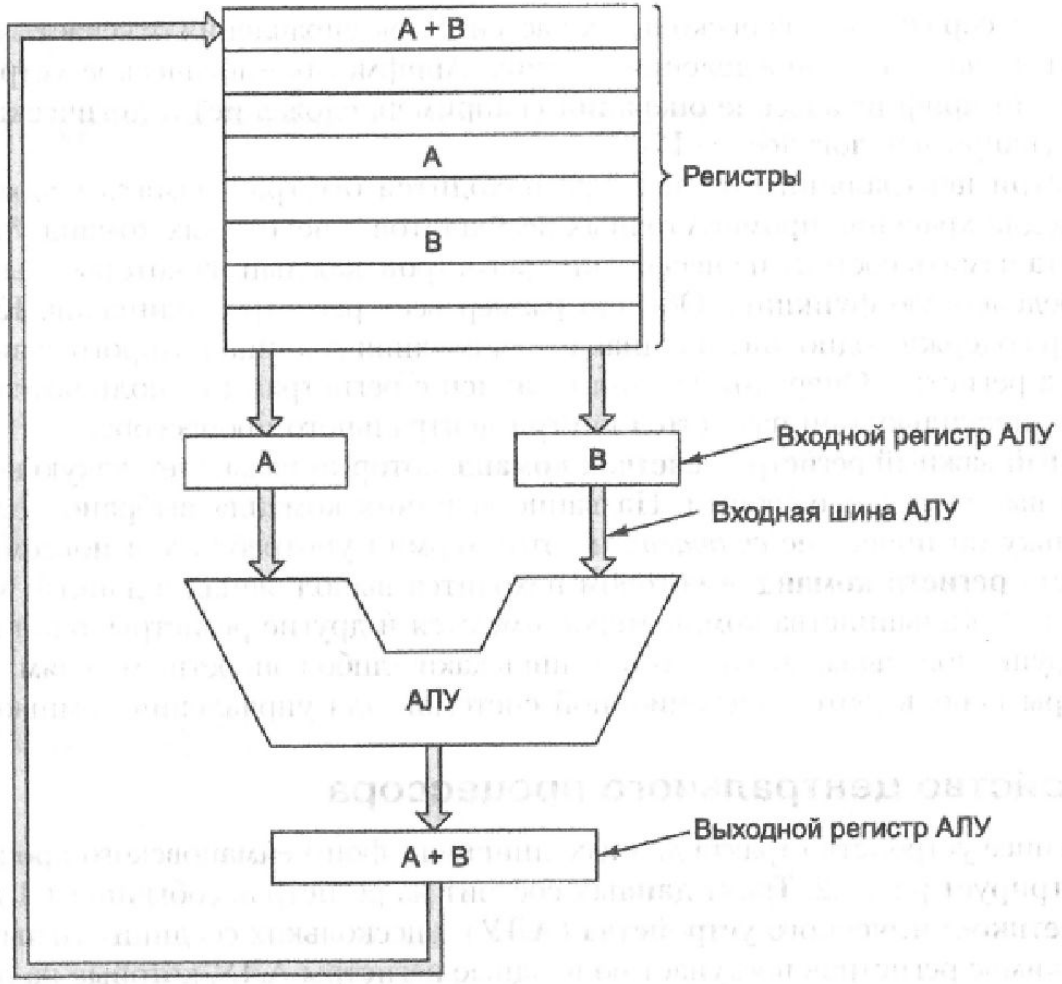


Рис. 2.2. Тракт данных обычной фон-неймановской машины

Тракт данных состоит из:

- ✔ регистров (обычно от 1 до 32)
- ✔ **арифметико-логического устройства (АЛУ)**
- ✔ нескольких соединительных шин.

Содержимое регистров поступает во входные регистры АЛУ, обозначены буквами A и B (в них находятся входные данные АЛУ, пока АЛУ производит вычисления).

АЛУ выполняет сложение, вычитание и другие простые операции над входными данными и помещает результат в выходной регистр.

Содержимое этого выходного регистра может записываться обратно в один из регистров или сохраняться в памяти, если это необходимо.

Не во всех архитектурах есть регистры A, B и выходные регистры.

Цикл тракта данных

- **Команды:**

✔ *тип регистр-память*: вызывают слова из памяти, помещают их в регистры, где они используются в качестве входных данных АЛУ (слова — это такие элементы данных, которые перемещаются между памятью и регистрами). Другие команды этого типа помещают слова обратно в память.

✔ *тип регистр-регистр:* вызывают два операнда из регистров, помещают их во входные регистры АЛУ, выполняют над ними какую-нибудь арифметиче скую или логическую операцию и переносят результат обратно в один из регистров.

Данные процессы называются **циклом тракта данных**.

Выполнение команд

ЦП выполняет каждую команду за несколько шагов:

1. Вызывает следующую команду из памяти и переносит ее в регистр команд.
2. Меняет положение счетчика команд, который после этого указывает на сле дующую команду.
3. Определяет тип вызванной команды.
4. Если команда использует слово из памяти, определяет, где находится это слово.
5. Переносит слово, если это необходимо, в регистр центрального процессора.
6. Выполняет команду.
7. Переходит к шагу 1, чтобы начать выполнение следующей команды.

Такая последовательность шагов (выборка – декодирование – исполнение) является основой работы всех компьютеров.

Интерпретатор для простого компьютера (на языке Java)

```
public class Interp{
static int PC;              // PC содержит адрес следующей команды
static int AC;              // Аккумулятор, регистр для арифметики
static int instr;           // Регистр для текущей команды
static int instretype;      // Тип команды (код операции)
static int data_loc;        // Адрес данных или -1, если его нет
static int data;            // Текущий операнд
static boolean run_bit = true;    // Бит, который можно сбросить,
                                  // чтобы остановить машину

public static void interpret(int memory[], int starting_address{
// Эта процедура интерпретирует программы для простой машины,
// которая содержит команды только с одним операндом из
// памяти. Машина имеет регистр AC (аккумулятор). Он
// используется для арифметических действий - например,
// команда ADD суммирует число из памяти с AC. Интерпретатор
// работает до тех пор, пока не будет выполнена команда
// HALT, вследствие чего бит run_bit поменяет значение на
// false. Машина состоит из блока памяти, счетчика команд, бита
// run bit и аккумулятора AC. Входные параметры представляют собой
// копию содержимого памяти и начальный адрес.

PC=starting_address;
while (run_bit) {
instr=memory[PC];          // Вызывает следующую команду в instr
PC=PC+1;                   // Увеличивает значение счетчика команд
instrjtype=get_instrjtype(instr);        // Определяет тип команды
data_loc=find_data(instr, instrjtype);          //   Находит данные (-1,
                                                // если данных нет)
if(data_loc>=0)            // Если data_loc=-l, значит, операнда нет
data=memory[data_loc];                          // Вызов данных
execute(instr_type,data);                 // Выполнение команды
private static int get_instr_type(int addr) {...}
private static int find_data(int instr, int type) {...}
private static void execute(int type, int data) {...}
}
```

Описание работы центрального процессора можно представить в виде программы.

Такая программа называется **интерпретатором.**

В описываемом компьютере есть два регистра:

- ✔ счетчик команд (PC) с адресом следующей команды
- ✔ аккумулятор (AC), в котором хранятся результаты арифметических операций.

Кроме того, имеются внутренние регистры, в которых хранится текущая команда (instr), тип текущей команды (instr_type), адрес операнда команды (data_loc) и сам операнд (data).

Каждая команда содержит один адрес ячейки памяти. В ячейке памяти хранится операнд – например, фрагмент данных, который нужно добавить в аккумулятор.

Эквивалентность аппаратных процессоров и интерпретаторов

Эквивалентность аппаратных процессоров и интерпретаторов имеет важные последствия для организации компьютера и проектирования компьютерных систем.

После того как разработчик выбрал машинный язык (Я) для нового компьютера, он должен решить:

- ✔ разрабатывать ли ему процессор, который будет выполнять программы на языке Я
- ✔ написать специальную программу для интерпретации программ на том же языке => разработать аппаратное обеспечение для исполнения этого интерпретатора.

Интерпретатор разбивает команды на более мелкие (элементарные) => машина, предназначенная для исполнения интерпретатора, может быть гораздо проще по строению и дешевле, чем процессор, выполняющий программы без интерпретации.

History. Интерпретация

- Первые компьютеры поддерживали небольшое количество команд, и эти команды были простыми.
- Разработка более мощных компьютеров привела к появлению сложных команд.
- У дорогих компьютеров было гораздо больше команд, чем у дешевых.
- Создание семейства компьютеров имеющих единую архитектуру и много разных моделей, отличающихся по цене и скорости, но «умеющих» выполнять одни и те же программы (50-е года).
- Интерпритация (Уилкс, 1951 год) позволяла разрабатывать простые дешевые ком пьютеры, которые, тем не менее, могли выполнять большое количество команд.

History. Интерпретация

Достоинства интерпретации:

- возможность исправлять неправильно реализованные команды «на месте» или даже компенсировать ошибки аппаратного обеспечения на уровне аппаратного обеспечения;
- возможность добавлять новые команды при минимальных затратах, причем при необходимости уже после покупки компьютера;
- возможность (благодаря структурированной организации) разработки, проверки и документирования сложных команд

History. Интерпретация

- К концу 70-х годов интерпретаторы стали применяться практически во всех моделях, кроме самых дорогих машин с очень высокой производительностью (на пример, Cray-1 и компьютеров серии Control Data Cyber).
- Эта тенденция достигла своего апогея в компьютере VAX, разработанном компанией DEC; у него было несколько сот команд и более 200 способов опре деления операндов в каждой команде.

Преимущества процессоров с интерпретацией:

можно разработать очень простой процессор, а вся самое сложное реализовать с помощью интерпретатора => вместо разработки сложной аппаратуры требо валась разработка сложного программного обеспечения.

Еще один фактор в пользу интерпретации — существование быстродействую щих постоянных запоминающих устройств для хранения интерпретаторов (так называемых командных ПЗУ).

Системы RISC и CISC

1980 г. разработка не ориентированных на интерпретацию процессоров VLSI (Дэвид Паттерсон (David Patterson) и Карло Секвин (Carlo Sequin))

.

Компьютер **RISC** (Reduced Instruction Set Computer – **компьютер с сокращенным набором команд)** противопоставлялся системе

CISC (Complex Instruction Set Computer — **компьютер с полным набором команд)**

Компьютеры **RISC** имели преимущества в плане производительности, но были несовместима с другими моделями.

Системы RISC и CISC

- По мнению сторонников **RISC**, наилучший способ разработки компьютеров – включение туда небольшого количества простых команд, каждая из которых выполняется за один цикл тракта данных, то есть производит над парой регистров какую-либо арифметическую или логическую операцию (например, сложение или операцию логического И) и помещает результат обратно в регистр.
- В качестве аргумента они утверждали, что даже если системе RISC приходится выполнять 4 или 5 команд вместо одной, которую выполняет CISC, RISC все равно выигрывает в скорости, так как RISC-команды выполняются в 10 раз быстрее (поскольку они не интерпретируются).

Системы RISC и CISC

Компания Intel сумела воплотить те же идеи в архитектуре CISC:

- Процессоры Intel, начиная с процессора 486, **содержат RISC-ядро**, которое выполняет самые простые (и обычно самые распространенные) команды за один цикл тракта данных, а по обычной технологии CISC интерпретируются более сложные команды.
- В результате обычные команды выполняются быстро, а более сложные и редкие — медленно. Хотя при таком «гибридном» подходе производительность ниже, чем в архитектуре RISC, новая архитектура CISC имеет ряд преимуществ, поскольку позволяет использовать старое программное обеспечение без изменений.

Принципы проектирования современных компьютеров. Принципы RISC

1. Все команды должны выполняться непосредственно аппаратным обеспече нием.

То есть обычные команды выполняются напрямую, без интерпретации микрокомандами. Устранение уровня интерпретации повышает скорость выполнения большинства команд. В компьютерах типа CISC более сложные команды могут разбиваться на несколько шагов, которые затем выполняются как последовательность микрокоманд. Эта дополнительная операция снижает быстродействие машины, но может использоваться для редко применяемых команд.

Принципы RISC

2. Компьютер должен запускать как можно больше команд в секунду.

Этот принцип предполагает, что **параллелизм** должен стать важным фактором повышения производительности, поскольку запустить на выполнение большое количество команд за короткий промежуток времени можно только в том случае, если есть возможность одновременного выполнения нескольких команд.

Принципы RISC

3. Команды должны легко декодироваться.

Предел количества запускаемых в секунду команд зависит от темпа декодирования отдельных команд.

Декодирование команд позволяет определить, какие ресурсы им необходимы и какие действия нужно выполнить.

Все, что способствует упрощению этого процесса, полезно.

Например, можно использовать единообразные команды с фиксированной длиной и с небольшим количеством полей. Чем меньше разных форматов команд, тем лучше.

Принципы RISC

4. К памяти должны обращаться только команды загрузки и сохранения.

Один из самых простых способов разбить операцию на отдельные шаги – сделать так, чтобы операнды большей части команд брались из регистров и возвращались туда же.

Операция перемещения операндов из памяти в регистры и обратно может осуществляться в разных командах. Поскольку доступ к памяти занимает много времени, длительность которой невозможно спрогнозировать, выполнение этих команд могут взять на себя другие команды, единственное назначение которых — перемещение операндов между регистрами и памятью.

То есть к памяти должны обращаться только команды загрузки и сохранения (LOAD и STORE).

Принципы RISC

5. Регистров должно быть много.

Поскольку доступ к памяти происходит относительно медленно, в компьютере должно быть много регистров (по крайней мере 32).

Если слово было однажды загружено из памяти, при наличии большого числа регистров оно может содержаться в регистре до тех пор, пока не потребуется.

Возвращение слова из регистра в память и новая загрузка этого же слова в регистр нежелательны. Лучший способ избежать излишних перемещений – наличие достаточного количества регистров.

Параллелизм

Один из способов заставить процессоры работать быстрее – повышение их тактовой частоты.

Большинство проектировщиков для повышения производительности при данной тактовой частоте процессора используют параллелизм (выполнение двух или более операций одновременно).

Существует две основные формы параллелизма:

- **Параллелизм на уровне команд** – реализуется за счет запуска большого количества команд каждую секунду.
- **Параллелизм на уровне процессоров**: над одним заданием работают одновременно несколько процессоров.

Параллелизм на уровне команд

- Главным препятствием высокой скорости выполнения команд является необходимость их загрузки из памяти.
- Для разрешения этой проблемы можно вызывать команды из памяти заранее и хранить в специальном наборе регистров.

Эта идея использовалась еще в 1959 году при разработке компьютера Stretch компании IBM, а набор регистров был назван **буфером выборки с упреждением**.

Таким образом, когда требовалась определенная команда, она вызывалась прямо из буфера, а обращения к памяти не происходило.

При выборке с упреждением команда обрабатывается за два шага:

✔ сначала происходит выборка команды, а затем ее выполнение.

Конвейеры

При использовании конвейера команда обрабатывается за большее количество шагов, каждый из которых реализуется определенным аппаратным компонентом, причем все эти компоненты могут работать параллельно.

Конвейер из 5 блоков

Первая ступень (блок C1) вызывает команду из памяти и помещает ее в буфер, где она хранится до тех пор, пока не потребуется.
Вторая ступень (блок C2) декодирует эту команду, определяя ее тип и тип ее операндов.
Третья ступень (блок C3) определяет местонахождение операндов и вызывает их из регистров или из памяти.
Четвертая ступень (блок C4) выполняет команду, обычно проводя операнды через тракт данных.
Пятая ступень (блок C5) записывает результат обратно в нужный регистр.

Состояние каждой ступени конвейера в зависимости от количества пройденных циклов

- **Цикл 1** блок C1 обрабатывает команду 1, вызывая ее из памяти.
- **Цикл 2** блок C2 декодирует команду 1, в то время как блок C1 вызывает из памяти команду 2.
- **Цикл 3** блок C3 вызывает операнды для команды 1, блок C2 декодирует команду 2, а блок C1 вызывает команду 3.
- **Цикл 4** блок C4 выполняет команду 1, C3 вызывает операнды для команды 2, C2 декодирует команду 3, а C1 вызывает команду 4.
- **Цикл 5** блок C5 записывает результат выполнения команды 1 обратно в регистр, тогда как другие ступени конвейера обрабатывают следующие команды.

Конвейеры

Конвейеры позволяют добиться компромисса между ***временем запаздывания*** (время выполнения одной команды) и ***пропускной способностью процессора*** (количество команд, выполняемых процессором в секунду).

- Если время обращения составляет T нс (2нс), а конвейер имеет n ступеней (5), время запаздывания составит nT нс (10нс).
- Поскольку одна команда выполняется за одно обращение, а за одну секунду таких обращений набирается $10^9/T$, количество команд в секунду также составляет $10^9/T$.

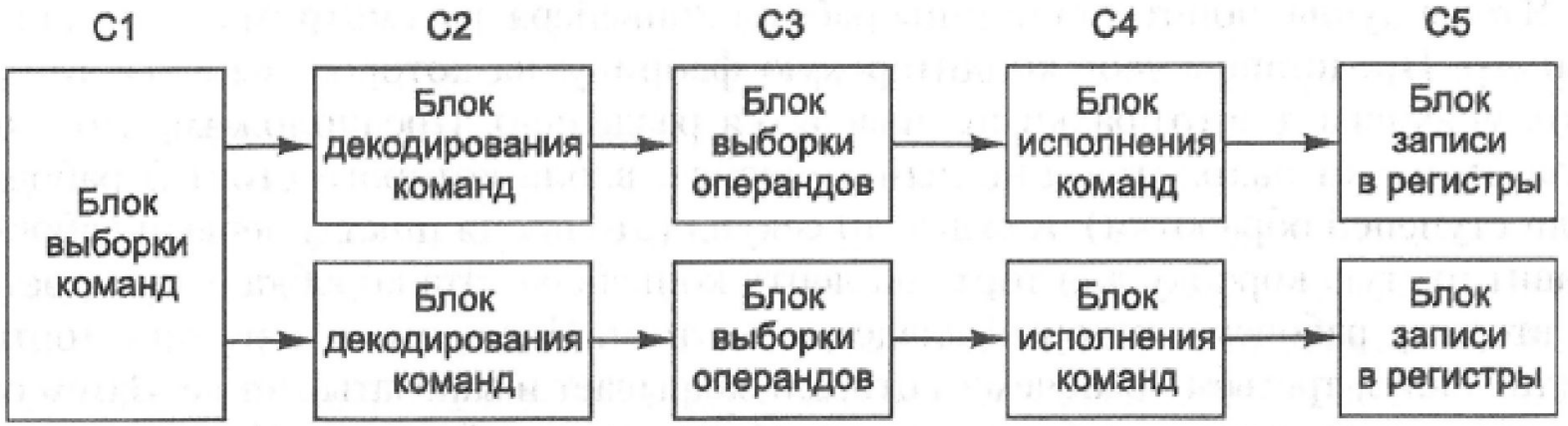
Если время цикла T = 2 нс, то каждую секунду выполняется 500 млн команд

$$(10^9/T)/10^6 = 1000/T \text{ MIPS}$$

(million instructions per second –миллион операций в секунду)

Сдвоенный пятиступенчатый конвейер с общим блоком выборки команд

Один конвейер – хорошо,
а два – еще лучше.



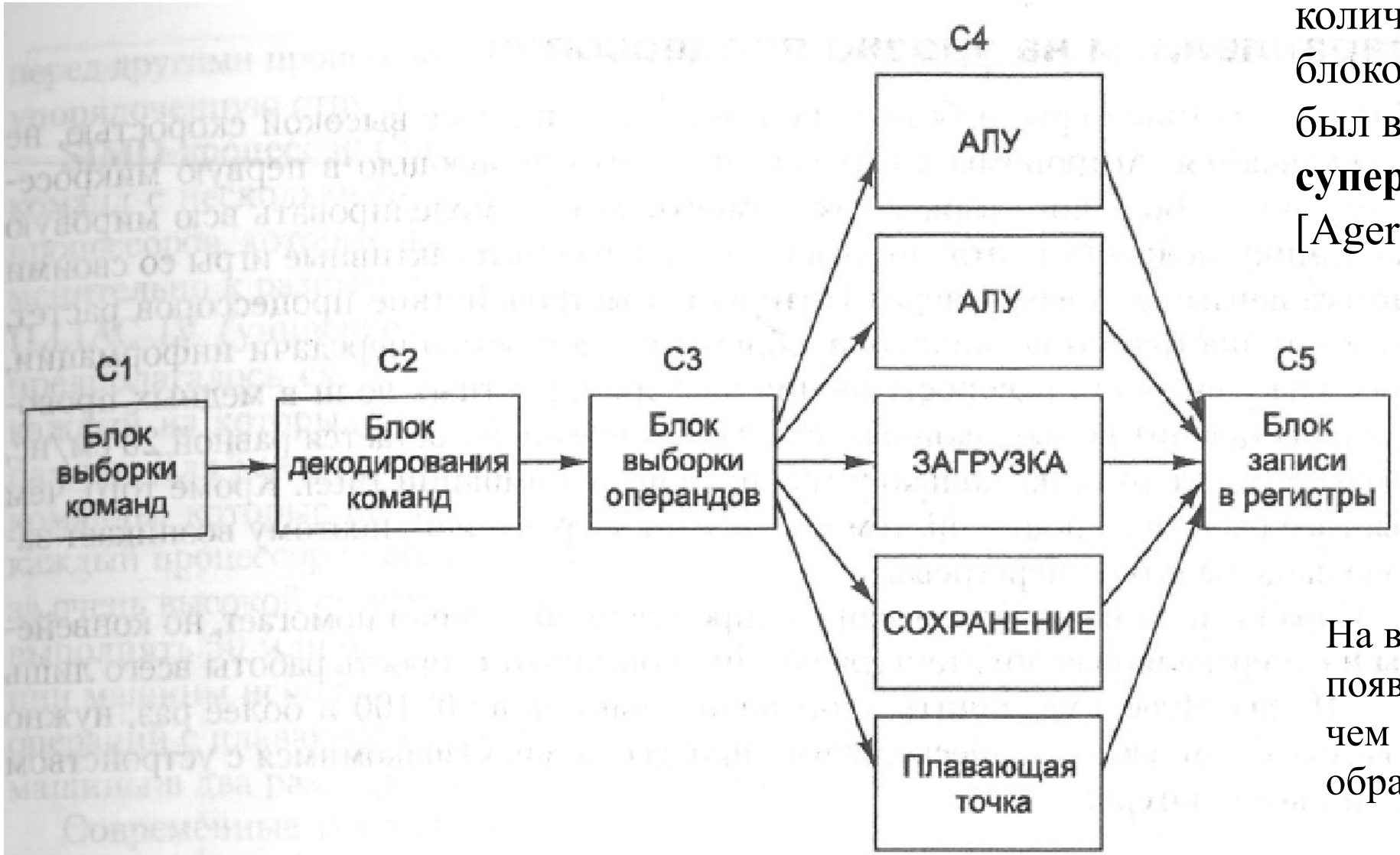
- Общий блок выборки команд вызывает из памяти сразу по две команды и помещает каждую из них в один из конвейеров.
- Каждый конвейер содержит АЛУ для параллельных операций.
- Чтобы выполняться параллельно, две команды не должны конфликтовать из-за ресурсов (например, регистров) и ни одна из них не должна зависеть от результата выполнения другой.
- Либо компилятор должен гарантировать отсутствие нештатных ситуаций (когда, например, аппаратура не обеспечивает проверку команд на несовместимость и при обработке таких команд выдает некорректный результат), либо конфликты должны выявляться и устраняться дополнительным оборудованием непосредственно в ходе выполнения команд.

Конвейеры

- Сначала конвейеры (как сдвоенные, так и обычные) использовались только в RISC-компьютерах.
- Конвейеры в процессорах компании Intel появились, только начиная с модели 486. Процессор 486 имел один пятиступенчатый конвейер, а Pentium – два таких конвейера.
- Главный конвейер **(u-конвейер)** мог выполнять произвольные команды. Второй конвейер **(v-конвейер)** мог выполнять только простые команды с целыми числами, а также одну простую команду с плавающей точкой **(FXCH).**

!!! Переход к четырем конвейерам возможен, но требует громоздкого аппаратного обеспечения

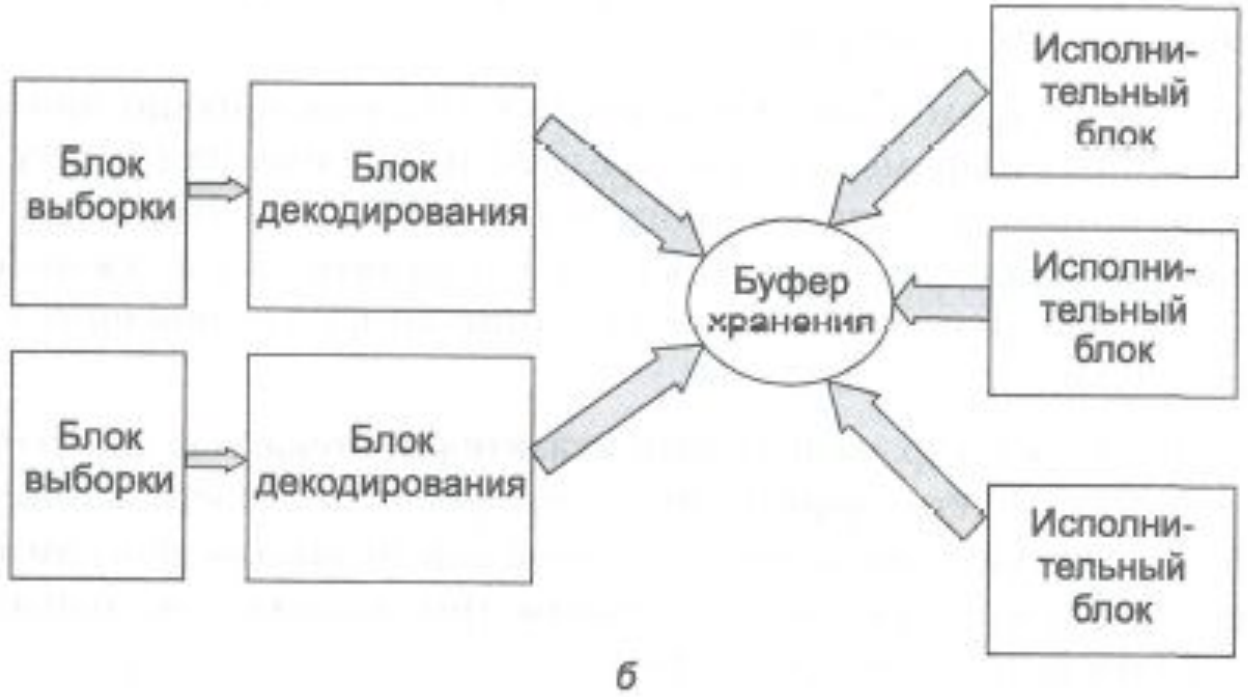
Суперскалярный процессор с пятью функциональными блоками



Один конвейер с большим количеством функциональных блоков – для данного подхода был введен термин **суперскалярная архитектура** [Agerwala and Cocke, 1987].

На выходе ступени 3 команды появляются зчательно быстрее, чем ступень 4 их способна обрабатывать.

Суперскалярный процессор



Имеет несколько исполнительных блоков, например:

- ✔ один — для целочисленной арифметики
- ✔ другой — для арифметики чисел с плавающей точкой
- ✔ третий — для логических операций.

Одновременно выбираются две и более команды, которые декодируются и помещаются в буфер хранения, в ко тором ожидают возможности своего выполнения.

Как только исполнительный блок становится доступен, он обращается к буферу хранения за командой, которую может выполнить, и если такая команда имеется, извлекает ее из буфера, а затем выполняет.

Режимы работы центральных процессоров

Большинство центральных процессоров, используемых во встраиваемых системах, имеют два режима работы: **режим ядра** и **пользовательский режим** (режим пользователя).

Обычно режимом управляет специальный бит в слове состояния программы - PSW.

При работе в режиме ядра процессор может выполнять любые команды из своего набора и использовать любые возможности аппаратуры.

Как правило, в пользовательском режиме запрещены все команды касающиеся операций ввода-вывода и защиты памяти.

Параллелизм на уровне процессоров

- Конвейеры и суперскалярная архитектура обычно повышают скорость работы всего лишь в 5-10 раз.
- Чтобы увеличить производительность в 50, 100 и более раз, нужно создавать компьютеры с несколькими процессорами.

Матричные компьютеры

Матричные компьютеры – используются для ускоренного выполнения больших научных программ:

- ✔ SIMD-процессоры.
- ✔ векторные процессоры.

Процессоры с распараллеливанием по данным:

- ✔ могут обеспечивать существенную вычислительную мощность с меньшим количеством транзисторов.
- ✔ так как все процессоры выполняют одну инструкцию, системе необходим только один «мозг», управляющий компьютером.
- ✔ соответственно процессору нужен одна ступень выборки команд, одна ступень декодирования и один блок управляющей логики.

Главное условие: выполняемые программы должны иметь упорядоченную структуру с большой степенью параллелизма

Матричные компьютеры

SIMD-процессор (Single Instruction-stream Multiple Data-stream — один поток команд с несколькими потоками данных) состоит из большого числа сходных процессоров, которые выполняют одну и ту же последовательность команд применительно к разным наборам данных.
Первым в мире таким процессором был ILLIAC IV (университет Иллинойс) [Bouknight et al., 1972].

Применение: современные графические процессоры (GPU) широко используют SIMD-обработку для обеспечения высокой вычислительной мощности при относительно небольшом количестве транзисторов.

SIMD-ядро графического процессора Fermi

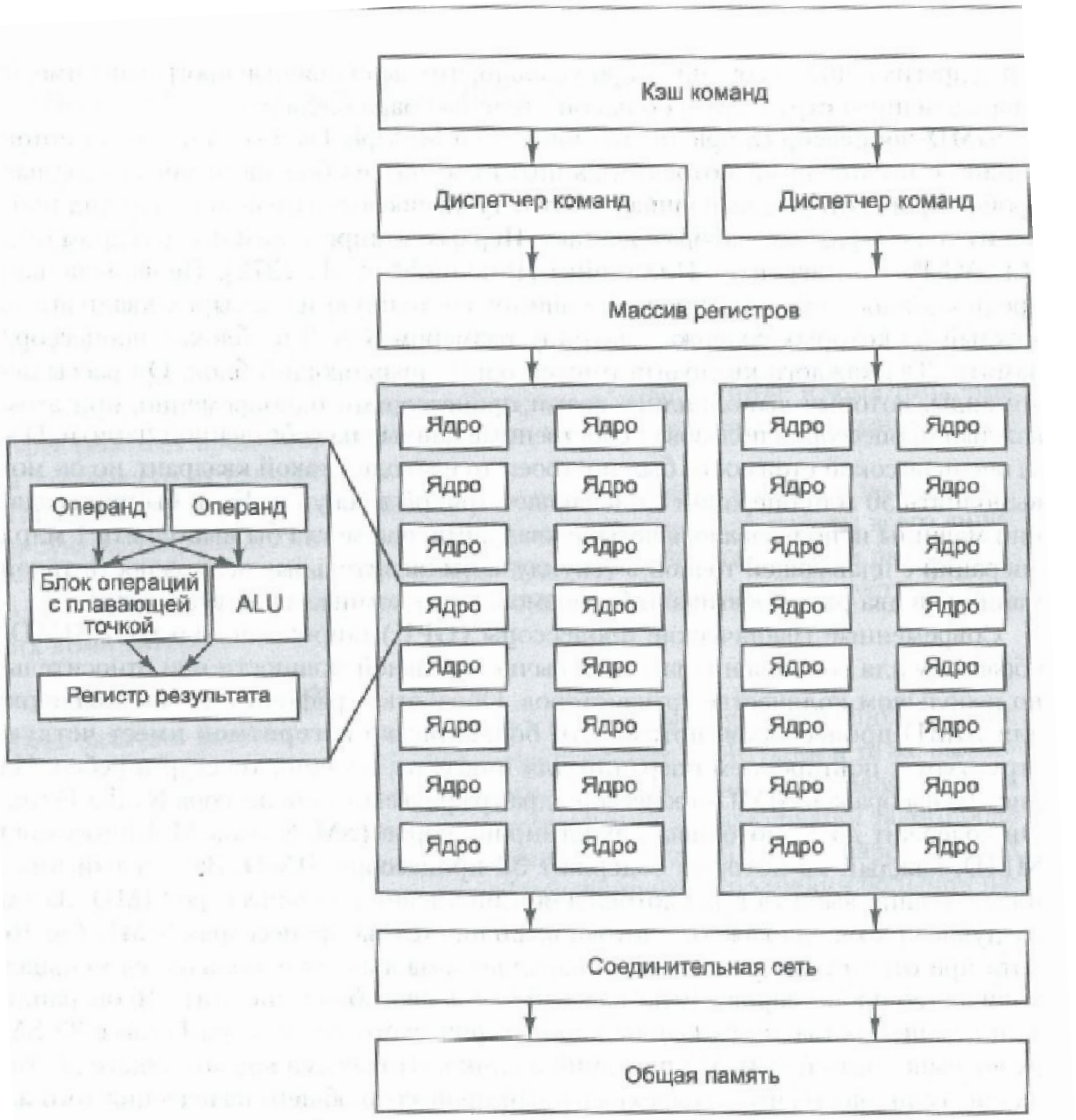
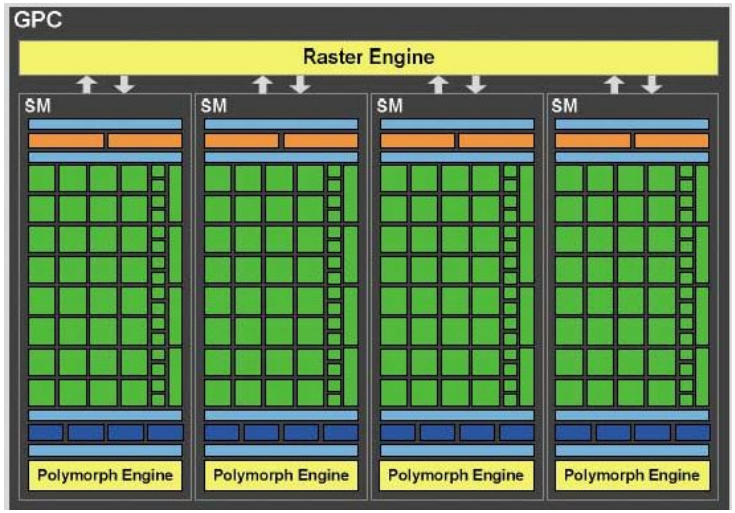


Рис. 2.6. SIMD-ядро графического процессора Fermi

Содержит до 16 потоковых мультипроцессоров (SM, Stream Multiprocessor) SIMD, каждый из которых содержит 32 процессора SIMD.

За каждый цикл планировщик выбирает два потока для выполнения на процессоре SIMD. Затем следующая команда каждого потока выполняется на процессорах SIMD (до 16, хотя при отсутствии достаточного параллелизма данных используется меньшее количество процессоров). Если каждый поток способен выполнить 16 операций за цикл, полностью загруженное ядро графического процессора Fermi с 32 SM будет выполнять целых 512 операций за цикл.

Каждый GPC-кластер фактически представляет собой отдельный графический процессор, за тем лишь исключением, что не имеет выделенного доступа к памяти



Каждый отдельный кластер GPC представляет собой совокупность четырех потоковых мультипроцессоров (Streaming Multiprocessor, SM)
Отдельный потоковый мультипроцессор SM включает 32 процессорных ядра

В одном кластере насчитывается четыре потоковых мультипроцессора, а всего в чипе имеется четыре кластера, то получим 512 процессорных ядер

Векторный процессор (vector processor)

Векторный процессор - это процессор, в котором операндами некоторых команд могут выступать упорядоченные массивы данных - векторы. Отличается от скалярных процессоров, которые могут работать только с одним операндом в единицу времени.

Векторный процессор (vector processor) очень похож на SIMD-процессор: эффективен при выполнении последовательности операций над парами элементов данных, однако все операции сложения выполняются в одном блоке суммирования, который имеет конвейерную структуру

- ✔ Оба типа процессоров работают с массивами данных и выполняют одни и те же команды, которые, например, попарно складывают элементы двух векторов.
- ✔ Но! У SIMD-процессора столько же суммирующих устройств, сколько элементов в массиве, а векторный процессор содержит векторный регистр, состоящий из набора традиционных регистров.

Процессорная плата векторного компьютера Cray YMP

Векторный процессор (vector processor)

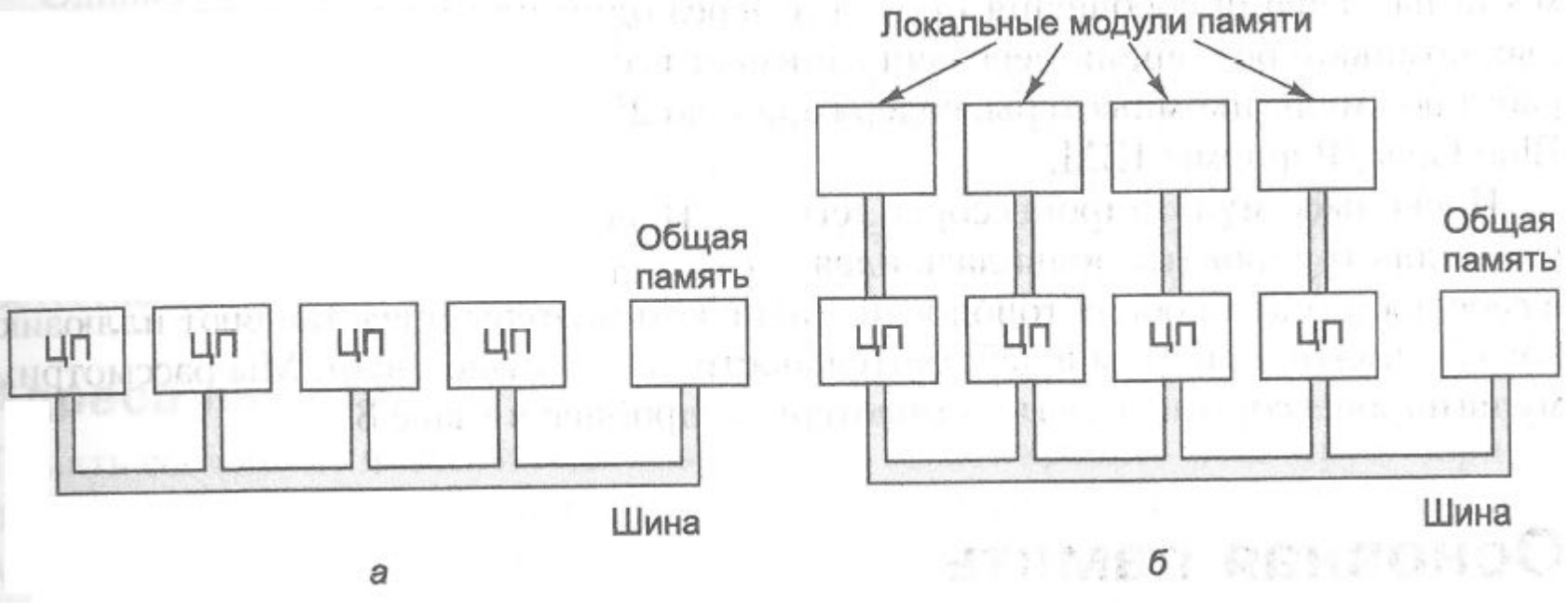
1. Регистры загружаются из памяти единственной командой, которая фактически делает это последовательно.
2. Команда сложения попарно складывает элементы двух таких векторов, загружая их из двух векторных регистров в суммирующее устройство с конвейерной структурой.
3. В результате из суммирующего устройства выходит другой вектор, который либо помещается в векторный регистр, либо сразу используется в качестве операнда при выполнении другой операции с векторами.

✔ **Команды SSE (Streaming SIMD Extension)** в архитектуре Intel Core используют эту модель расширения для ускорения вычислений с высокой степенью упорядоченности — например, обработки мультимедийных и научных данных.

Мультипроцессоры

- Элементы процессора, распараллеленного по данным, связаны между собой, поскольку их работу контролирует единый блок управления.
- Система из нескольких параллельных процессоров, имеющих общую память, называется **мультипроцессором**.
- Поскольку каждый процессор может записывать информацию в любую часть памяти и считывать информацию из любой части памяти, чтобы не допустить каких-либо нестыковок, их работа должна согласовываться программным обеспечением.
- Когда два или несколько процессоров имеют возможность тесного взаимодействия эти процессоры называют сильно связанными (tightly coupled).

Мультипроцессоры



Мультипроцессор с единственной шиной и общей памятью (а);

Мультипроцессор с собственной локальной памятью для каждого процессора (б)

Мультипроцессоры имеют преимущество перед другими видами параллельных компьютеров, поскольку с единой общей памятью очень легко работать.

Мультипроцессоры

- Мультипроцессоры с небольшим числом процессоров (< 256) разрабатывать достаточно просто.
- Создание больших мультипроцессоров представляет определенные трудности. Сложность заключается в том, чтобы связать все процессоры с общей памятью.

Мультикомпьютеры

Системы без общей памяти, состоящие из большого числа взаимосвязанных компьютеров, у каждого из которых имеется собственная память называются **мультикомпьютерами** (например Blue Gene/P фирмы IBM)

- В мультикомпьютерах процессоры являются слабо связанными.
- Процессоры мультикомпьютера отправляют друг другу сообщения (что-то вроде электронной почты, но гораздо быстрее).
- Каждый компьютер не обязательно соединять со всеми другими, поэтому обычно в качестве топологий используются двух- и трехмерные решетки, а также деревья и кольца.

Хотя на пути до места назначения сообщения проходят через один или несколько промежуточных компьютеров, время передачи занимает всего несколько микросекунд

Мультипроцессоры легче программировать, а мультикомпьютеры — конструировать => появилась идея создания гибридных систем, сочетающих в себе достоинства обеих топологий.