ВВЕДЕНИЕ

Стремительное развитие науки и проникновение человеческой мысли во все новые области вместе с решением поставленных прежде проблем постоянно порождает поток вопросов и ставит новые, как правило более сложные, задачи. Во времена первых компьютеров казалось, что увеличение их быстродействия в 100 раз позволит решить большинство проблем, однако гигафлопная производительность современных суперЭВМ сегодня является явно недостаточной для многих ученых. Электро- и гидродинамика, сейсморазведка и прогноз погоды, моделирование химических соединений, исследование виртуальной реальности - вот далеко не полный список областей науки, исследователи которых используют каждую возможность ускорить выполнение своих программ.

Наиболее перспективным и динамичным направлением увеличения скорости решения прикладных задач является широкое внедрение идей параллелизма в работу вычислительных систем. К настоящему времени спроектированы и опробованы сотни различных компьютеров, использующих в своей архитектуре тот или иной вид параллельной обработки данных. В научной литературе и технической документации можно найти более десятка различных названий, характеризующих лишь общие принципы функционирования параллельных машин: векторно-конвейерные, массивно-параллельные, компьютеры с широким командным словом, систолические массивы, гиперкубы, спецпроцессоры и мультипроцессоры, иерархические и кластерные компьютеры, dataflow, матричные ЭВМ и многие другие. Если же к подобным названиям для полноты описания добавить еще и данные о таких важных параметрах, как, например, организация памяти, топология связи между процессорами, синхронность работы отдельных устройств или способ исполнения арифметических операций, то число различных архитектур станет и вовсе необозримым.

Попытки систематизировать все множество архитектур начались после опубликования М.Флинном первого варианта классификации вычислительных систем в конце 60-х годов и непрерывно продолжаются по сей день. Ясно, что навести порядок в хаосе очень важно для лучшего понимания исследуемой предметной области, однако нахождение удачной классификации может иметь

целый ряд существенных следствий.

В самом деле, вспомним открытый в 1869 году Д.И.Менделеевым периодический закон. Выписав на карточках названия химических элементов и указав их важнейшие свойства, он сумел найти такое расположение, при котором четко прослеживалась закономерность в изменении свойств элементов, расположенных в каждом столбце и в каждой строке. Теперь, зная положение какого-либо элемента в таблице, он мог с большой степенью точности описывать его свойства, не проводя с ним никаких непосредственных экспериментов. Другим, поистине фантастическим следствием, явилось то, что данный закон сразу указал на несколько "белых пятен" в таблице и позволил предсказать существование (!) и свойства (!!) неизвестных до тех пор элементов. В 1875 году французский ученый Буабодран, изучая спектры минералов, открыл предсказанный Менделеевым галлий и впервые подробно описал его свойства. В свою очередь Менделеев, никогда прежде не видевший данного химического элемента, не только смог указать на ошибку в определении плотности, но и вычислил ее правильное значение.

Существующая классификация растительного и животного мира, в отличие от периодического закона, носит скорее описательный характер. С ее помощью намного сложнее предсказывать существование нового вида, однако знание того, что исследуемый экземпляр принадлежит такому-то роду/семейству/отряду/классу позволяет оправданно предположить наличие у него вполне определенных свойств.

Подобную классификацию хотелось бы найти и для архитектур параллельных вычислительных систем. Основной вопрос - что заложить в основу классификации, может решаться по-разному, в зависимости от того, для кого данная классификация создается и на решение какой задачи направлена. Так, часто используемое деление компьютеров на персональные ЭВМ, рабочие станции, мини--ЭВМ, большие универсальные ЭВМ, минисупер-ЭВМ и супер-ЭВМ, позволяет, быть может, примерно прикинуть стоимость компьютера. Однако она не приближает пользователя к пониманию того, что от него потребуется для написания программы, работающий на пределе производительности параллельного компьютера, т.е. того, ради чего он и решился его использовать. Как это ни странно, но от обилия разных параллельных компьютеров страдает, прежде всего, конечный пользователь, для которого, вроде бы, они и создавались: он вынужден каждый раз подбирать наиболее эффективный алгоритм, он испытывает на себе "прелести" параллельного программирования и отладки,

решает проблемы переносимости и затем все повторяется заново.

Хотелось бы, чтобы такая классификация помогла ему разобраться с тем, что представляет собой каждая архитектура, как они взаимосвязаны между собой, что он должен учитывать для написания действительно эффективных программ или же на какой класс архитектур ему следует ориентироваться для решения требуемого класса задач. Одновременно удачная классификация могла бы подсказать возможные пути совершенствования компьютеров и в этом смысле она должна быть достаточно содержательной. Трудно рассчитывать на нахождение нетривиальных "белых пятен", например, в классификации по стоимости, однако размышления о возможной систематике с точки зрения простоты и технологичности программирования могут оказаться чрезвычайно полезными для определения направлений поиска новых архитектур.

В данной работе не ставилась задача сразу предложить что-то конкретное. Она носит скорее обзорный характер и ее основная задача - это собрать в одном месте накопленый к настоящему времени материал. В работу включены не все найденные классификации, а описаны лишь те, в которых впервые введены какие-либо новые существенные понятия.

1 Классификация Шора

Классификация Дж.Шора, появившаяся в начале 70-х годов, интересна тем, что представляет собой попытку выделения типичных способов компоновки вычислительных систем на основе фиксированного числа базисных блоков: устройства управления, арифметико-логического устройства, памяти команд и памяти данных. Дополнительно предполагается, что выборка из памяти данных может осуществляться словами, то есть выбираются все разряды одного слова, и/или битовым слоем - по одному разряду из одной и той же позиции каждого слова (иногда эти два способа называют горизонтальной и вертикальной выборками соответственно). Конечно же, при анализе данной классификации надо делать скидку на время ее появления, так как предусмотреть невероятное разнообразие параллельных систем настоящего времени было в принципе невозможно. Итак, согласно классификации Шора все компьютеры разбиваются на шесть классов, которые он так и называет: машина типа I, II и т.д.

Машина I (рис. 1.5) - это вычислительная система, которая содержит устройство управления, арифметико-логическое устройство, память команд и память данных с пословной выборкой. Считывание данных осуществляется выборкой всех разрядов

некоторого слова для их параллельной обработки в арифметико-логическом устройстве. Состав АЛУ специально не оговаривается, что допускает наличие нескольких функциональных устройств, быть может конвейерного типа. По этим соображениям в данный класс попадают как классические последовательные машины (IBM 701, PDP-11, VAX 11/780), так и конвейерные скалярные (CDC 7600) и векторно-конвейерные (CRAY-1).

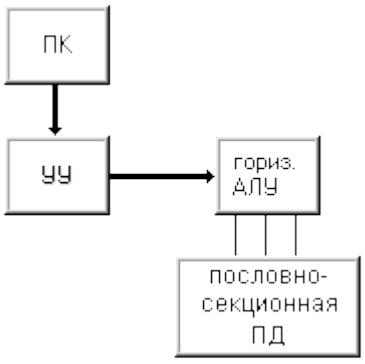

Рисунок 1.5 – Машина I

Если в машине I осуществлять выборку не по словам, а выборкой содержимого одного разряда из всех слов, то получим **машину II (рис. 1.6)** . Слова в памяти данных по прежнему располагаются горизонтально, но доступ к ним осуществляется иначе. Если в машине I происходит последовательная обработка слов при параллельной обработке разрядов, то в машине II - последовательная обработка битовых слоев при параллельной обработке множества слов.

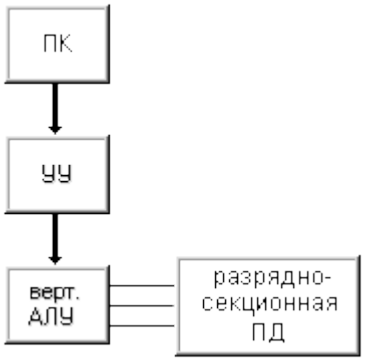

Рисунок 1.6 – Машина II

Структура машины II лежит в основе ассоциативных компьютеров (например, центральный процессор машины STARAN), причем фактически такие компьютеры имеют не одно арифметико-логическое устройство, а множество сравнительно простых устройств поразрядной обработки. Другим примером служит матричная система ICL DAP, которая может одновременно обрабатывать по одному разряду из

4096 слов.

Если объединить принципы построения машин I и II, то получим **машину III** (рис. 1.7). Эта машина имеет два арифметико-логических устройства - горизонтальное и вертикальное, и модифицированную память данных, которая обеспечивает доступ как к словам, так и к битовым слоям. Впервые идею построения таких систем в 1960 году выдвинул У.Шуман , называвший их ортогональными (если память представлять как матрицу слов, то доступ к данным осуществляется в направлении, "ортогональном" традиционному - не по словам (строкам), а по битовым слоям (столбцам)). В принципе, как машину STARAN, так и ICL DAP можно запрограммировать на выполнение функций машины III, но поскольку они не имеют отдельных АЛУ для обработки слов и битовых слоев, отнести их к данному классу нельзя. Полноправными представителями машин класса III являются вычислительные системы семейства OMEN-60 фирмы Sanders Associates, построенные в прямом соответствии с концепцией ортогональной машины.

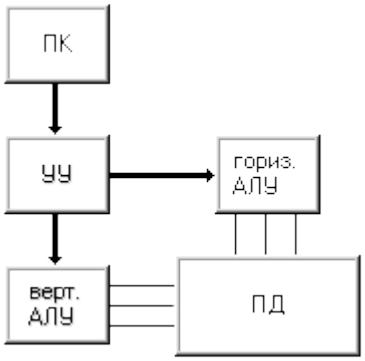

Рисунок 1.7 – Машина III

Если в машине I увеличить число пар арифметико-логическое устройство <=> память данных (иногда эту пару называют процессорным элементом) то получим **машину IV** (рис. 1.8). Единственное устройство управления выдает команду за командой сразу всем процессорным элементам. С одной стороны, отсутствие соединений между процессорными элементами делает дальнейшее наращивание их числа относительно простым, но с другой, сильно ограничивает применимость машин этого класса. Такую структуру имеет вычислительная система PEPE, объединяющая 288 процессорных элементов.

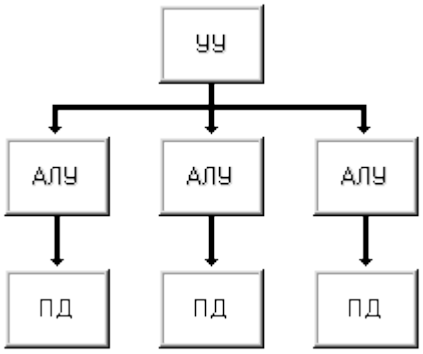

Рисунок 1.8 – Машина IV

Если ввести непосредственные линейные связи между соседними процессорными элементами машины IV, например в виде матричной конфигурации, то получим схему **машины V** (рис. 1.9). Любой процессорный элемент теперь может обращаться к данным как в своей памяти, так и в памяти непосредственных соседей. Подобная структура характерна, например, для классического матричного компьютера ILLIAC IV.

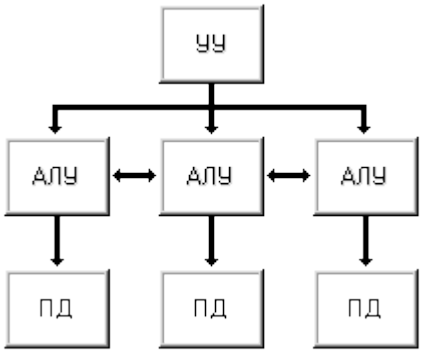

Рисунок 1.9 – Машина V

Заметим, что все машины с I-ой по V-ю придерживаются концепции разделения памяти данных и арифметико-логических устройств, предполагая наличие шины данных или какого-либо коммутирующего элемента между ними. **Машина VI** (рис. 1.10), названная матрицей с функциональной памятью (или памятью с встроенной логикой), представляет собой другой подход, предусматривающий распределение логики процессора по всему запоминающему устройству. Примерами могут служить как простые ассоциативные запоминающие устройства, так и сложные ассоциативные процессоры.

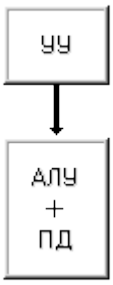

Рисунок 1.10 – Машина VI

2 ОРГАНИЗАЦИЯ КОМПЬЮТЕРНЫХ СИСТЕМ

2.1 Общие сведения

Цифровой компьютер состоит из связанных между собой процессоров, памяти и устройств ввода-вывода.

На рис. 2.1 показано устройство обычного компьютера. Центральный процессор — это мозг компьютера. Его задача — выполнять программы, находящиеся в основной памяти. Он вызывает команды из памяти, определяет их тип, а затем выполняет их одну за другой. Компоненты соединены шиной, представляющей собой набор параллельно связанных проводов, по которым передаются адреса, данные и сигналы управления. Шины могут быть внешними (связывающими процессор с памятью и устройствами ввода-вывода) и внутренними.

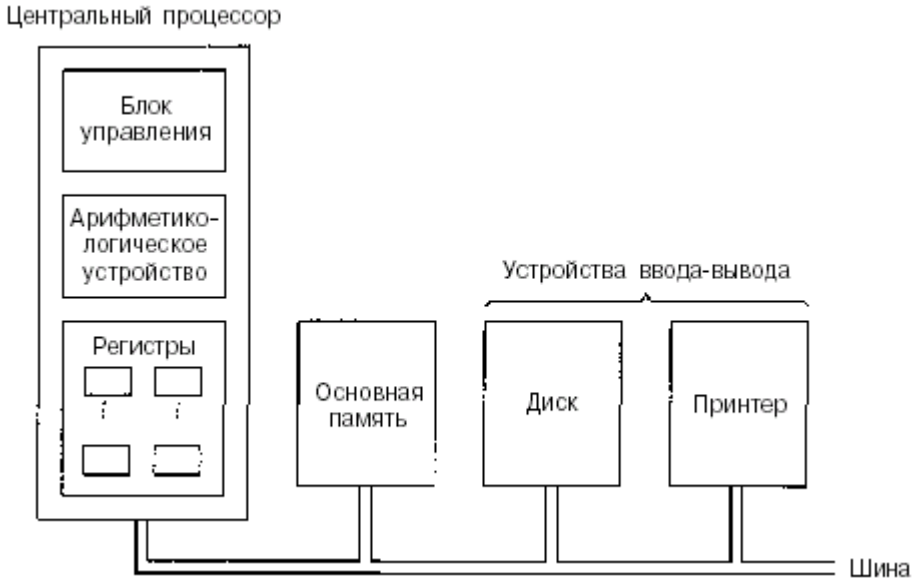

Рисунок 2.1 – . Схема устройства компьютера с одним центральным процессором и двумя устройствами ввода-вывода

Процессор состоит из нескольких частей. 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).

Внутри центрального процессора находится память для хранения промежуточных результатов и некоторых команд управления. Эта память состоит из нескольких регистров, каждый из которых выполняет определенную функцию. Обычно все регистры одинакового размера. Каждый регистр содержит одно число, которое ограничивается размером регистра. Регистры считываются и записываются очень быстро, поскольку они находятся внутри центрального процессора.

Самый важный регистр — счетчик команд, который указывает, какую команду нужно выполнять дальше. Название «счетчик команд» не соответствует действительности, поскольку он ничего не считает, но этот термин употребляется повсеместно. Еще есть регистр команд, в котором находится команда, выполняемая в данный момент. У большинства компьютеров имеются и другие регистры, одни из них многофункциональны, другие выполняют только какие-либо специфические функции.

2.2 Устройство центрального процессора

Внутреннее устройство тракта данных типичного фон-неймановского процессорапоказано на рис. 2.2. Тракт данных состоит из регистров (обычно от 1 до 32), АЛУ (арифметико-логического устройства) и нескольких соединяющих шин. Содержимое регистров поступает во входные регистры АЛУ, которые на рис. 2.2 обозначены буквами A и B. В них находятся входные данные АЛУ, пока АЛУ производит вычисления. Тракт данных — важная составная часть всех компьютеров.

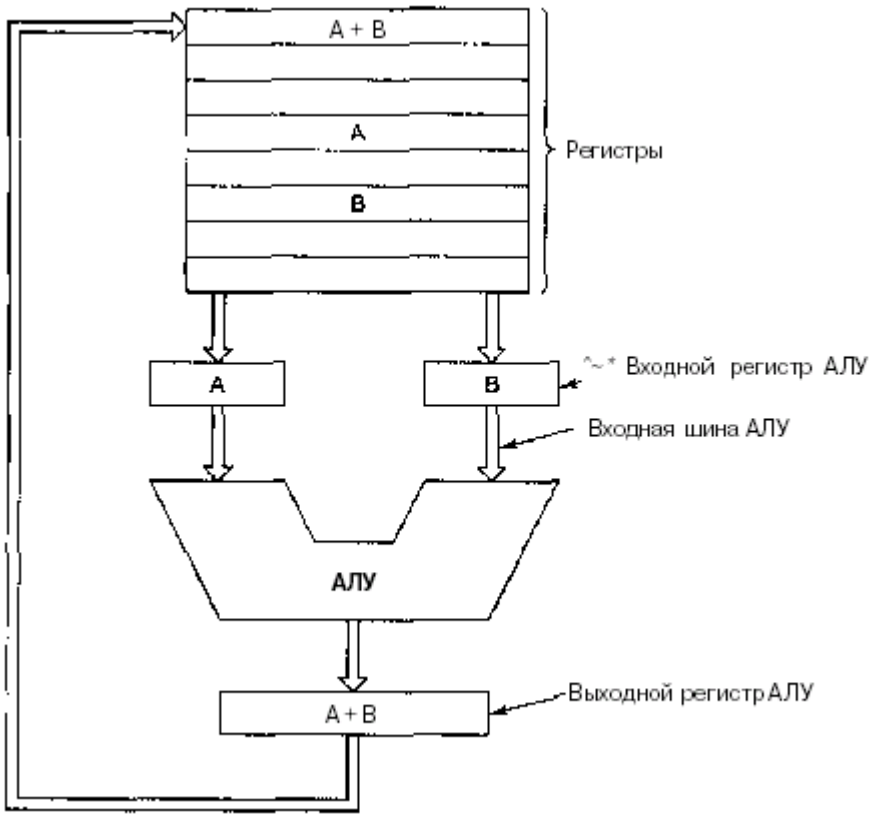

Рисунок 2.2 - Тракт данных в обычной фон-неймановской машине

АЛУ выполняет сложение, вычитание и другие простые операции над входными данными и помещает результат в выходной регистр. Этот выходной регистр может помещаться обратно в один из регистров. Он может быть сохранен в памяти, если это необходимо. На рис. 2.2 показана операция сложения. Однако входные и выходные регистры есть не у всех компьютеров.

Большинство команд можно разделить на две группы: команды типа регистр-память и типа регистр-регистр. Команды первого типа 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.) Словом может быть целое число. Другие команды этого типа помещают регистры обратно в память.

Команды второго типа вызывают два операнда из регистров, помещают их во входные регистры АЛУ, выполняют над ними какую-нибудь арифметическую или логическую операцию и переносят результат обратно в один из регистров. Этот процесс называется циклом тракта данных. В какой-то степени он определяет, что может делать машина. Чем быстрее происходит цикл тракта данных, тем быстрее

компьютер работает.

2.3 Выполнение команд

Центральный процессор выполняет каждую команду за несколько шагов:

1) вызывает следующую команду из памяти и переносит ее в регистр команд;

2) меняет положение счетчика команд, который теперь должен указывать наследующую 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.

Первые компьютеры содержали небольшое количество команд, и эти команды были простыми. Но поиски более мощных компьютеров привели, кроме всего прочего, к появлению более сложных команд. Вскоре разработчики поняли, что при наличии сложных команд программы выполняются быстрее, хотя выполнение отдельных команд занимает больше времени. В качестве примеров сложных команд можно назвать выполнение операций с плавающей точкой, обеспечение прямого доступа к элементам массива и т. п. Если обнаруживалось, что две определенные команды часто выполнялись последовательно одна за другой, то вводилась новая команда, заменяющая работу этих двух.

Сложные команды были лучше, потому что некоторые операции иногда перекрывались. Какие-то операции могли выполняться параллельно, для этого использовались разные части аппаратного обеспечения. Для дорогих компьютеров с высокой производительностью стоимость этого дополнительного аппаратного обеспечения была вполне оправданна. Таким образом, у дорогих компьютеров было гораздо больше команд, чем у дешевых. Однако развитие программного обеспечения и требования совместимости команд привели к тому, что сложные команды стали использоваться и в дешевых компьютерах, хотя там во главу угла

ставилась стоимость, а не скорость работы.

К концу 50-х годов компания IBM, которая лидировала тогда на компьютерном рынке, решила, что производство семейства компьютеров, каждый из которых выполняет одни и те же команды, имеет много преимуществ и для самой компании, и для покупателей. Чтобы описать этот уровень совместимости, компания IBM ввела термин архитектура. Новое семейство компьютеров должно было иметь одну общую архитектуру и много разных разработок, различающихся по цене и скорости, которые могли выполнять одну и ту же программу. Но как построить дешевый компьютер, который будет выполнять все сложные команды, предназначенные для высокоэффективных дорогостоящих машин?

Решением этой проблемы стала интерпретация. Эта технология, впервые предложенная Уилксом в 1951 году, позволяла разрабатывать простые дешевые компьютеры, которые, тем не менее, могли выполнять большое количество команд. В результате IBM создала архитектуру System/360, семейство совместимых компьютеров, различных по цене и производительности. Аппаратное обеспечение без интерпретации использовалось только в самых дорогих моделях.

Простые компьютеры с интерпретированными командами имели некоторые другие преимущества. Наиболее важными среди них были:

1) возможность фиксировать неправильно выполненные команды или даже восполнять недостатки аппаратного обеспечения;

2) возможность добавлять новые команды при минимальных затратах, даже после покупки компьютера;

3) структурированная организация, которая позволяла разрабатывать, проверять и документировать сложные команды.

В 70-е годы компьютерный рынок быстро разрастался, новые компьютеры могли выполнять все больше и больше функций. Спрос на дешевые компьютеры провоцировал создание компьютеров с использованием интерпретаторов. Возможность разрабатывать аппаратное обеспечение и интерпретатор для определенного набора команд вылилась в создание дешевых процессоров. Полупроводниковые технологии быстро развивались, преимущества низкой стоимости преобладали над возможностями более высокой производительности, и использование интерпретаторов при разработке компьютеров стало широко применимо. Интерпретация использовалась практически во всех компьютерах,

выпущенных в 70-е годы, от миникомпьютеров до самых больших машин.

К концу 70-х годов интерпретаторы стали применяться практически во всех моделях, кроме самых дорогостоящих машин с очень высокой производительностью (например, Cray-1 и компьютеров серии Control Data Cyber). Использование интерпретаторов исключало высокую стоимость сложных команд, и разработчики могли вводить все более и более сложные команды, в особенности различные способы определения используемых операндов.

Эта тенденция достигла пика своего развития в разработке компьютера VAX (производитель Digital Equipment Corporation), у которого было несколько сотен команд и более 200 способов определения операндов в каждой команде. К несчастью, архитектура VAX с самого начала разрабатывалась с использованием интерпретатора, а производительности уделялось мало внимания. Это привело к появлению большого количества команд второстепенного значения, которые трудно было выполнять сразу без интерпретации. Данное упущение стало фатальным как для VAX, так и для его производителя (компании DEC). Compaq купил DEC в 1998 году.

Хотя самые первые 8-битные микропроцессоры были очень простыми и содержали небольшой набор команд, к концу 70-х годов даже они стали разрабатываться с использованием интерпретаторов. В этот период основной проблемой для разработчиков стала возрастающая сложность микропроцессоров. Главное преимущество интерпретации заключалось в том, что можно было разработать простой процессор, а вся сложность сводилась к созданию интерпретатора. Таким образом, разработка сложного аппаратного обеспечения замещалась разработкой сложного программного обеспечения.

Успех Motorola 68000 с большим набором интерпретируемых команд и одновременный провал Zilog Z8000, у которого был столь же обширный набор команд, но не было интерпретатора, продемонстрировали все преимущества использования интерпретаторов при разработке новых машин. Успех Motorola 68000 был несколько неожиданным, учитывая, что Z80 (предшественник Zilog Z8000) пользовался большей популярностью, чем Motorola 6800 (предшественник Motorola 68000). Конечно, важную роль здесь играли и другие факторы, например то, что Motorola много лет занималась производством микросхем, а Exxon (владелец Zilog) долгое время был нефтяной компанией.

Еще один фактор в пользу интерпретации — существование быстрых постоянных запоминающих устройств (так называемых командных ПЗУ) для хранения интерпретаторов. Предположим, что для выполнения обычной интерпретируемой команды Motorola 68000 интерпретатору нужно выполнить 10 команд, которые называются микрокомандами, по 100 не каждая, и произвести 2 обращения к оперативной памяти по 500 не каждое. Общее время выполнения команды составит следовательно, 2000 не, всего лишь в два раза больше, чем в лучшем случае могло бы занять непосредственное выполнение этой команды без интерпретации. А если бы не было специального быстродействующего постоянного запоминающего устройства, выполнение этой команды заняло бы целых 6000 не. Таким образом, важность наличия командных ПЗУ очевидна.

2.4 RISC и CISC

В конце 70-х годов проводилось много экспериментов с очень сложными командами, появление которых стало возможным благодаря интерпретации. Разработчики пытались уменьшить пропасть между тем, что компьютеры способны делать и тем, что требуют языки высокого уровня. Едва ли кто-нибудь тогда думал о разработке более простых машин, так же как сейчас мало кто занимается разработкой менее мощных операционных систем, сетей, редакторов и т. д. (к несчастью).

В компании IBM группа разработчиков во главе с Джоном Коком противостояла этой тенденции; они попытались воплотить идеи Сеймура Крея, создав экспериментальный высокоэффективный мини-компьютер 801. Хотя IBM не занималась сбытом этой машины, а результаты эксперимента были опубликованы только через несколько лет, весть быстро разнеслась по свету, и другие производители тоже занялись разработкой подобных архитектур.

В 1980 году группа разработчиков в университете Беркли во главе с Дэвидом Паттерсоном и Карло Секвином начала разработку процессоров VLSI без использования интерпретации. Для обозначения этого понятия они придумали термин RISC и назвали новый процессор RISC I, вслед за которым вскоре был выпущен RISC II. Немного позже, в 1981 году, Джон Хеннеси в Стенфорде разработал и выпустил другую микросхему, которую он назвал MIPS. Эти две микросхемы развились в коммерчески важные продукты SPARC и MIPS соответственно.

Новые процессоры существенно отличались от коммерческих процессоров того времени. Поскольку они не были совместимы с существующей продукцией, разработчики вправе были включать туда новые наборы команд, которые могли бы увеличить общую производительность системы. Так как основное внимание уделялось простым командам, которые могли быстро выполняться, разработчики вскоре осознали, что ключом к высокой производительности компьютера была разработка команд, к выполнению которых можно быстро приступать. Сколько времени занимает выполнение одной команды, было не так важно, как то, сколько команд может быть начато в секунду.

В то время как разрабатывались эти простые процессоры, всеобщее внимание привлекало относительно небольшое количество команд (обычно их было около 50). Для сравнения: число команд в DEC VAX и больших IBM в то время составляло от 200 до 300. RISC — это сокращение от Reduced Instruction Set Computer — компьютер с сокращенным набором команд. RISC противопоставлялся CISC (Complex Instruction Set Computer — компьютер с полным набором команд). В качестве примера CISC можно привести VAX, который доминировал в то время в научных компьютерных центрах. На сегодняшний день мало кто считает, что главное различие RISC и CISC состоит в количестве команд, но название сохраняется до сих пор.

С этого момента началась грандиозная идеологическая война между сторонниками RISC и разработчиками VAX, Intel и больших IBM. По их мнению, наилучший способ разработки компьютеров — включение туда небольшого количества простых команд, каждая из которых выполняется за один цикл тракта данных, то есть берет два регистра, производит над ними какую-либо арифметическую или логическую операцию (например, сложения или логическое И) и помещает результат обратно в регистр. В качестве аргумента они утверждали, что даже если RISC должна выполнять 4 или 5 команд вместо одной, которую выполняет CISC, притом что команды RISC выполняются в 10 раз быстрее (поскольку они не интерпретируются), он выигрывает в скорости. Следует также отметить, что к этому времени скорость работы основной памяти приблизилась к скорости специальных управляющих постоянных запоминающих устройств, потому недостатки интерпретации были налицо, что повышало популярность компьютеров RISC.

Учитывая преимущества производительности RISC, можно было бы предположить, что такие компьютеры, как Alpha компании DEC, стали доминировать над компьютерами CISC (Pentium и т. д.) на рынке. Однако ничего подобного не

произошло. Возникает вопрос: почему?

Во-первых, компьютеры RISC были несовместимы с другими моделями, а многие компании вложили миллиарды долларов в программное обеспечение для продукции Intel. Во-вторых, как ни странно, компания Intel сумела воплотить те же идеи в архитектуре CISC. Процессоры Intel, начиная с 486-го, содержат ядро RISC, 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 работа происходит не так быстро, как у RISC, данная архитектура имеет ряд преимуществ, поскольку позволяет использовать старое программное обеспечение без изменений.

2.5 Принципы разработки современных компьютеров

Прошло уже более двадцати лет с тех пор, как были сконструированы первые компьютеры RISC, однако некоторые принципы разработки можно перенять, учитывая современное состояние технологий аппаратного обеспечения. Если происходит очень резкое изменение в технологиях (например, новый процесс производства делает время цикла памяти в 10 раз меньше, чем время цикла центрального процессора), меняются все условия. Поэтому разработчики всегда должны учитывать возможные технологические изменения, которые могут повлиять на баланс между компонентами компьютера.

Существует ряд принципов разработки, иногда называемых принципами RISC, которым по возможности стараются следовать производители универсальных процессоров. Из-за некоторых внешних ограничений, например требования совместимости с другими машинами, приходится время от времени идти на компромисс, но эти принципы — цель, к которой стремится большинство разработчиков.

Все команды непосредственно выполняются аппаратным обеспечением. Все обычные команды непосредственно выполняются аппаратным обеспечением. Они не интерпретируются микрокомандами. Устранение уровня интерпретации обеспечивает высокую скорость выполнения большинства команд. В компьютерах типа CISC более сложные команды могут разбиваться на несколько частей, которые затем выполняются как последовательность микрокоманд. Эта дополнительная операция снижает скорость работы машины, но она может быть

применима для редко встречающихся команд.

Компьютер должен начинать выполнение большого числа команд. В современных компьютерах используется много различных способов для увеличения производительности, главное из которых — возможность обращаться к как можно большему количеству команд в секунду. Процессор 500-MIPS способен приступать к выполнению 500 млн команд в секунду, и при этом не имеет значения, сколько времени занимает само выполнение этих команд (MIPS — это сокращение от Millions of Instructions Per Second — «миллионы команд в секунду».) Этот принцип предполагает, что параллелизм может играть главную роль в улучшении производительности, поскольку приступать к большому количеству команд за короткий промежуток времени можно только в том случае, если одновременно может выполняться несколько команд.

Хотя команды некоторой программы всегда расположены в определенном порядке, компьютер может приступать к их выполнению и в другом порядке (так как необходимые ресурсы памяти могут быть заняты) и, кроме того, может заканчивать их выполнение не в том порядке, в котором они расположены в программе. Конечно, если команда 1 устанавливает регистр, а команда 2 использует этот регистр, нужно действовать с особой осторожностью, чтобы команда 2 не считывала регистр до тех пор, пока он не будет содержать нужное значение. Чтобы не допустить подобных ошибок, необходимо вводить большое количество соответствующих записей в память, но производительность все равно становится выше благодаря возможности выполнять несколько команд одновременно.

Команды должны легко декодироваться. Предел количества вызываемых команд в секунду зависит от процесса декодирования отдельных команд. Декодирование команд осуществляется для того, чтобы определить, какие ресурсы им необходимы и какие действия нужно выполнить. Полезны любые средства, которые способствуют упрощению этого процесса. Например, используются регулярные команды с фиксированной длиной и с небольшим количеством полей. Чем меньше разных форматов команд, тем лучше.

К памяти должны обращаться только команды загрузки и сохранения. Один из самых простых способов разбивания операций на отдельные шаги — потребовать, чтобы операнды для большинства команд брались из регистров и возвращались туда же. Операция перемещения операндов из памяти в регистры может осуществляться в разных командах. Поскольку доступ к памяти занимает много

времени, а подобная задержка нежелательна, работу этих команд могут выполнять другие команды, если они не делают ничего, кроме передвижения операндов между регистрами и памятью. Из этого наблюдения следует, что к памяти должны обращаться только команды загрузки и сохранения (LOAD и STORE).

Должно быть большое количество регистров. Поскольку доступ к памяти происходит довольно медленно, в компьютере должно быть много регистров (по крайней мере 32). Если слово однажды вызвано из памяти, при наличии большого числа регистров оно может содержаться в регистре до тех пор, пока будет не нужно. Возвращение слова из регистра в память и новая загрузка этого же слова в регистр нежелательны. Лучший способ избежать излишних перемещений — наличие достаточного количества регистров.

2.6 Параллелизм на уровне команд

Разработчики компьютеров стремятся к тому, чтобы улучшить производительность машин, над которыми они работают. Один из способов заставить процессоры работать быстрее — увеличение их скорости, однако при этом существуют некоторые технологические ограничения, связанные с конкретным историческим периодом. Поэтому большинство разработчиков для достижения лучшей производительности при данной скорости работы процессора используют параллелизм (возможность выполнять две или более операций одновременно).

Существует две основные формы параллелизма: параллелизм на уровне команд и параллелизм на уровне процессоров. В первом случае параллелизм осуществляется в пределах отдельных команд и обеспечивает выполнение большого количества команд в секунду. Во втором случае над одной задачей работают одновременно несколько процессоров. Каждый подход имеет свои преимущества.

2.6.1 Конвейеры

Уже много лет известно, что главным препятствием высокой скорости выполнения команд является их вызов из памяти. Для разрешения этой проблемы разработчики придумали средство для вызова команд из памяти заранее, чтобы ониимелись в наличии в тот момент, когда будут необходимы. Эти команды помещались в набор регистров, который назывался буфером выборки с упреждением. Таким образом, когда была нужна определенная команда, она вызывалась прямо из буфера, и не нужно было ждать, пока она считается из памяти. Эта идея использовалась еще при разработке IBM Stretch, который был сконструирован в 1959 году.

В действительности процесс выборки с упреждением подразделяет выполнение команды на два этапа: вызов и собственно выполнение. Идея конвейера еще больше продвинула эту стратегию вперед. Теперь команда подразделялась уже не на два, а на несколько этапов, каждый из которых выполнялся определенной частью аппаратного обеспечения, причем все эти части могли работать параллельно.

На рис. 2.3а изображен конвейер из 5 блоков, которые называются стадиями. Стадия С1 вызывает команду из памяти и помещает ее в буфер, где она хранится до тех пор, пока не будет нужна. Стадия С2 декодирует эту команду, определяя ее тип и тип операндов, над которыми она будет производить определенные действия. Стадия С3 определяет местонахождение операндов и вызывает их или из регистров, или из памяти. Стадия С4 выполняет команду, обычно путем провода операндов через тракт данных. И наконец, стадия С5 записывает результат обратно в нужный регистр.

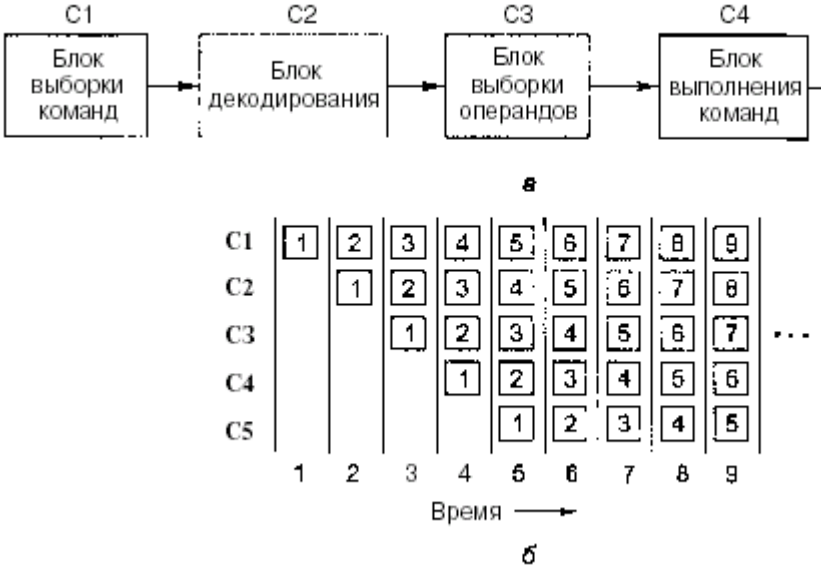

Рисунок 2.3 - Конвейер из 5 стадий (а); состояние каждой стадии в зависимости от количества пройденных циклов (б). Показано 9 циклов

На рис. 2.3, б мы видим, как действует конвейер во времени. Во время цикла 1 стадия С1 работает над командой 1, вызывая ее из памяти. Во время цикла 2 стадия С2 декодирует команду 1, в то время как стадия С1 вызывает из памяти команду 2. Во время цикла 3 стадия С3 вызывает операнды для команды 1, стадия С2 декодирует команду 2, а стадия С1 вызывает третью команду. Во время цикла 4 стадия С4 выполняет команду 1, С3 вызывает операнды для команды 2, С2 декодирует команду 3, а С1 вызывает команду 4. Наконец, во время пятого цикла

С5 записывает результат выполнения команды 1 обратно в регистр, тогда как другие стадии работают над следующими командами.

Конвейеры позволяют найти компромисс между временем ожидания (сколько времени занимает выполнение одной команды) и пропускной способностью процессора (сколько миллионов команд в секунду выполняет процессор). Если время цикла составляет T не, а конвейер содержит n стадий, то время ожидания составит nT не, а пропускная способность — 1000/T млн команд в секунду.

2.6.2 Суперскалярные архитектуры

Один конвейер — хорошо, а два — еще лучше. Одна из возможных схем процессора с двойным конвейером показана на рис. 2.4. В основе разработки лежит конвейер, изображенный на рис. 2.3. Здесь общий отдел вызова команд берет из памяти сразу по две команды и помешает каждую из них в один из конвейеров. Каждый конвейер содержит АЛУ для параллельных операций. Чтобы выполняться параллельно, две команды не должны конфликтовать при использовании ресурсов (например, регистров), и ни одна из них не должна зависеть от результата выполнения другой. Как и в случае с одним конвейером, либо компилятор должен следить, чтобы не возникало неприятных ситуаций (например, когда аппаратное обеспечение выдает некорректные результаты, если команды несовместимы), либо же конфликты выявляются и устраняются прямо во время выполнения команд благодаря использованию дополнительного аппаратного обеспечения.

Сначала конвейеры (как двойные, так и одинарные) использовались только в компьютерах RISC. У 386-го и его предшественников их не было. Конвейеры в процессорах компании Intel появились только начиная с 486-й модели1.486-й процессор содержал один конвейер, а Pentium — два конвейера из пяти стадий. Похожая схема изображена на рис. 2.4, но разделение функций между второй и третьей стадиями (они назывались декодирование 1 и декодирование 2) было немного другим. Главный конвейер (u-конвейер) мог выполнять произвольные команды. Второй конвейер (v-конвейер) мог выполнять только простые команды с целымичислами, а также одну простую команду с плавающей точкой (FXCH).

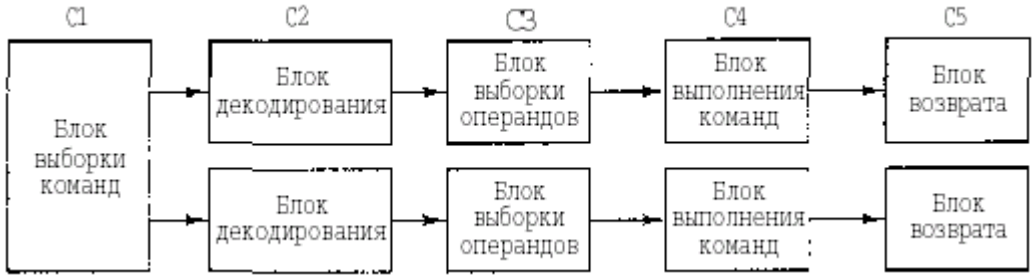

Рисунок 2.4 - Двойной конвейер из пяти стадий с общим отделом вызова команд

Имеются сложные правила определения, является ли пара команд совместимой для того, чтобы выполняться параллельно. Если команды, входящие в пару, были сложными или несовместимыми, выполнялась только одна из них (в и-конвейере). Оставшаяся вторая команда составляла затем пару со следующей командой. Команды всегда выполнялись по порядку. Таким образом, Pentium содержал особые компиляторы, которые объединяли совместимые команды в пары и могли порождать программы, выполняющиеся быстрее, чем в предыдущих версиях. Измерения показали, что программы, производящие операции с целыми числами, на компьютере Pentium выполняются почти в два раза быстрее, чем на 486-м, хотя у него такая же тактовая частота. Вне всяких сомнений, преимущество в скорости появилось благодаря второму конвейеру.

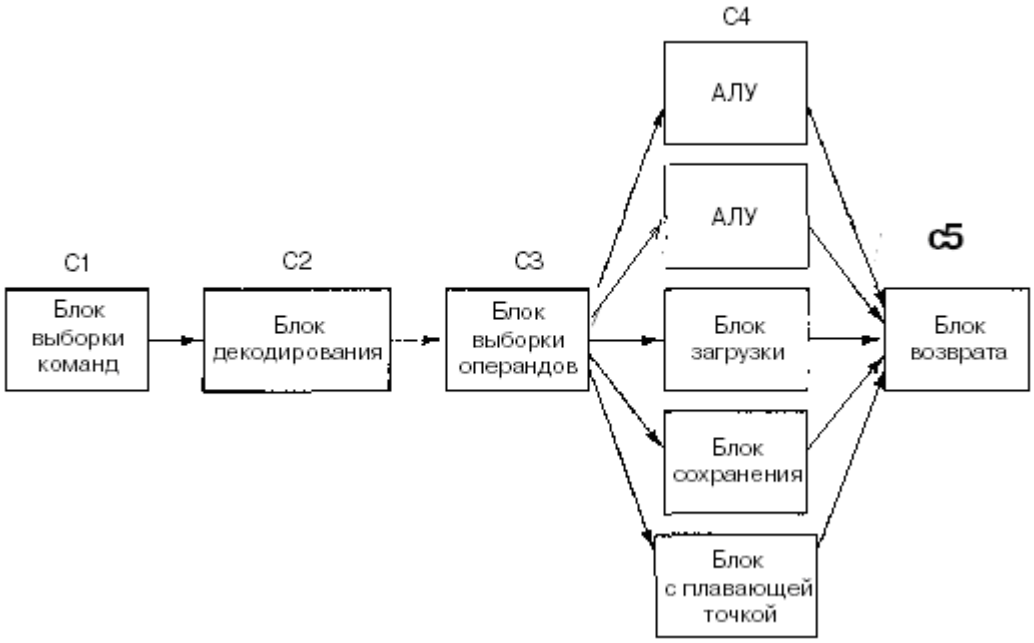

Рисунок 2.5 - Суперскалярный процессор с пятью функциональными блоками

Переход к четырем конвейерам возможен, но это потребовало бы создания громоздкого аппаратного обеспечения. Вместо этого используется другой подход.

Основная идея — один конвейер с большим количеством функциональных блоков, как показано па рис. 2.5. Pentium II, к примеру, имеет сходную структуру. В 1987 году для обозначения этого подхода был введен термин суперскалярная архитектура. Однако подобная идея нашла воплощение еще более 30 лет назад в компьютере CDC 6600. CDC 6600 вызывал команду из памяти каждые 100 не и помещал ее в один из 10 функциональных блоков для параллельного выполнения. Пока команды выполнялись, центральный процессор вызывал следующую команду.

Стадия 3 выпускает команды значительно быстрее, чем стадия 4 способна их выполнять. Если бы стадия 3 выпускала команду каждые 10 не, а все функциональные блоки выполняли бы свою работу также за 10 не, то на четвертой стадии всегда функционировал бы только один блок, что сделало бы саму идею конвейера бессмысленной. В действительности большинству функциональных блоков четвертой стадии для выполнения команды требуется значительно больше времени, чем занимает один цикл (это блоки доступа к памяти и блок выполнения операций с плавающей точкой). Как видно из рис. 2 5, на четвертой стадии может быть несколько АЛУ.

2.7 Параллелизм на уровне процессоров

Спрос на компьютеры, работающие все с более и более высокой скоростью, не прекращается. Астрономы хотят выяснить, что произошло в первую микросекунду после большого взрыва, экономисты хотят смоделировать всю мировую экономику, подростки хотят играть в 3D интерактивные игры со своими виртуальными друзьями через Интернет. Скорость работы процессоров повышается, но у них постоянно возникают проблемы с быстротой передачи информации, поскольку скорость распространения электромагнитных волн в медных проводах и света в оптико-волоконных кабелях по-прежнему остается 20 см/нс, независимо от того, насколько умны инженеры компании Intel. Кроме того, чем быстрее работает процессор, тем сильнее он нагревается, и нужно предохранять его от перегрева.

Параллелизм на уровне команд помогает в какой-то степени, но конвейеры и суперскалярная архитектура обычно увеличивают скорость работы всего лишь в 5-10 раз. Чтобы улучшить производительность в 50, 100 и более раз, нужно разрабатывать компьютеры с несколькими процессорами.

2.7.1 Векторные компьютеры

Многие задачи в физических и технических науках содержат векторы, в противном случае они имели бы очень сложную структуру. Часто одни и те же вычисления

выполняются над разными наборами данных в одно и то же время. Структура этих программ позволяет повышать скорость работы благодаря параллельному выполнению команд. Существует два метода, которые используются для быстрого выполнения больших научных программ. Хотя обе схемы во многих отношениях схожи, одна из них считается расширением одного процессора, а другая — параллельным компьютером.

Массивно-параллельный процессор (array processor) состоит из большого числа сходных процессоров, которые выполняют одну и ту же последовательность команд применительно к разным наборам данных. Первым в мире таким процессором был ILLIAC IV (Университет Иллинойса). Он изображен на рис. 2.6. Первоначально предполагалось сконструировать машину, состоящую из четырех секторов, каждый из которых содержит решетку 8x8 элементов процессор/память. Для каждого сектора имелся один блок контроля. Он рассылал команды, которые выполнялись всеми процессорами одновременно, при этом каждый процессор использовал свои собственные данные из своей собственной памяти (загрузка данных происходила во время инициализации). Из-за очень высокой стоимости был построен только один такой сектор, но он мог выполнять 50 млн операций с плавающей точкой в секунду. Если бы при создании машины использовалось четыре сектора и она могла бы выполнять 1 млрд операций с плавающей точкой в секунду, то мощность такой машины в два раза превышала бы мощность компьютеров всего мира.

Рисунок 2.6 - Массивно-параллельный процессор ILLIAC IV

Для программистов векторный процессор (vector processor) очень похож на массивно-параллельный процессор (array processor). Как и массивно-параллельный

процессор, он очень эффективен при выполнении последовательности операций над парами элементов данных. Но, в отличие от первого (array processor), все операции сложения выполняются в одном блоке суммирования, который имеет конвейерную структуру. Компания Cray Research, основателем которой был Сеймур Крей, выпустила много векторных процессоров, начиная с модели Cray-1 (1974) и по сей день. Cray Research в настоящее время входит в состав SGI.

Оба типа процессоров работают с массивами данных. Оба они выполняют одни и те же команды, которые, например, попарно складывают элементы для двух векторов. Но если у массивно-параллельного процессора (array processor) есть столько же суммирующих устройств, сколько элементов в массиве, векторный процессор (vector processor) содержит векторный регистр, который состоит из набора стандартных регистров. Эти регистры последовательно загружаются из памяти при помощи одной команды. Команда сложения попарно складывает элементы двух таких векторов, загружая их из двух векторных регистров в суммирующее устройство с конвейерной структурой. В результате из суммирующего устройства выходит другой вектор, который или помещается в векторный регистр, или сразу используется в качестве операнда при выполнении другой операции с векторами.

Массивно-параллельные процессоры (array processor) выпускаются до сих пор, но занимают незначительную сферу компьютерного рынка, поскольку они эффективны при решении только таких задач, которые требуют одновременного выполнения одних и тех же вычислений над разными наборами данных. Массивно-параллельные процессоры (array processor) могут выполнять некоторые операции гораздо быстрее, чем векторные компьютеры (vector computer), но они требуют большего количества аппаратного обеспечения, и для них сложно писать программы. Векторный процессор (vector processor), с другой стороны, можно добавлять к обычному процессору. В результате те части программы, которые могут быть преобразованы в векторную форму, выполняются векторным блоком, а остальная часть программы — обычным процессором.

2.7.2 Мультипроцессоры

Элементы массивно-параллельного процессора связаны между собой, поскольку их работу контролирует один блок управления. Система нескольких параллельных процессоров, разделяющих общую память, называется мультипроцессором. Поскольку каждый процессор может записывать или считывать информацию из любой части памяти, их работа должна согласовываться программным

обеспечением, чтобы не допустить каких-либо пересечений.

Возможны разные способы воплощения этой идеи. Самый простой из них — наличие одной шины, соединяющей несколько процессоров и одну общую память. Схема такого мультипроцессора показана на рис. 2.7а. Такие системы производят многие компании.

Нетрудно понять, что при наличии большого числа быстро работающих процессоров, которые постоянно пытаются получить доступ к памяти через одну и ту же шину, будут возникать конфликты. Чтобы разрешить эту проблему и повысить производительность компьютера, были разработаны различные модели. Одна из них изображена на рис. 2.7б. В таком компьютере каждый процессор имеет свою собственную локальную память, которая недоступна для других процессоров. Эта память используется для программ и данных, которые не нужно разделять между несколькими процессорами. При доступе к локальной памяти главная шина не используется, и, таким образом, поток информации в этой шине снижается. Возможны и другие варианты решения проблемы (например, кэш-память).

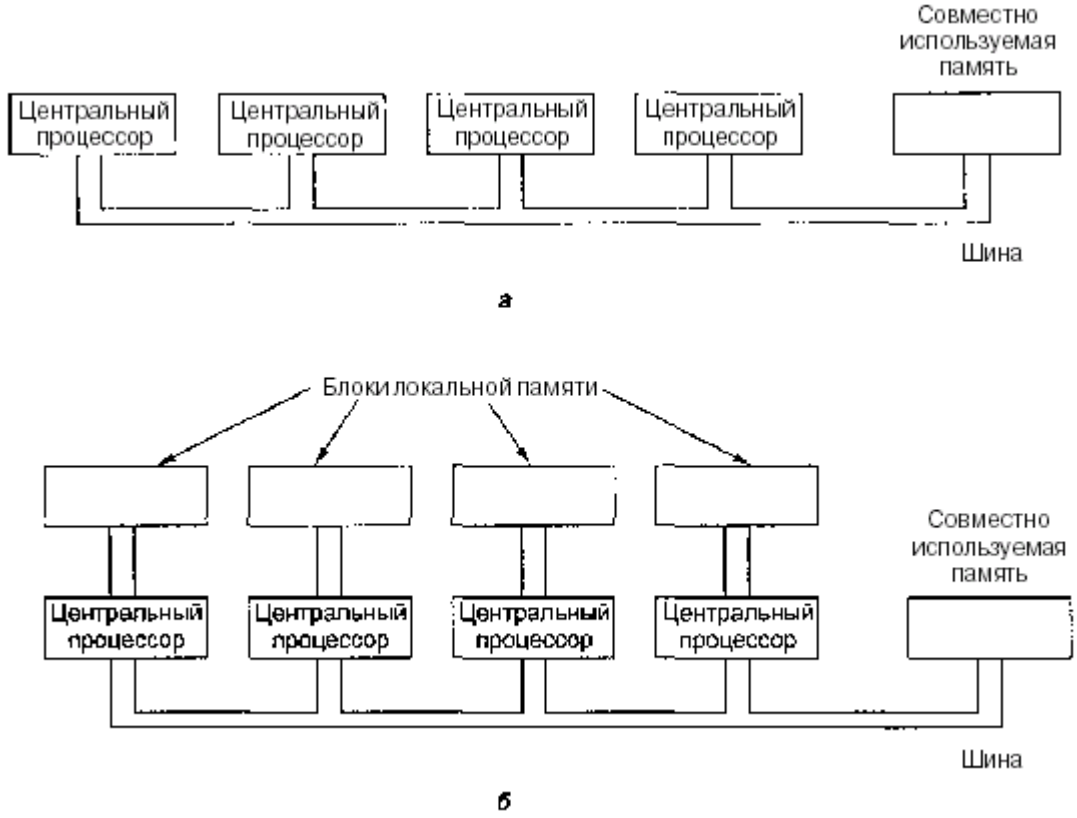

Рис. 2.7. Мультипроцессор с одной шиной и одной общей памятью (а); мультипроцессор,в котором для каждого процессора имеется собственная

локальная память (б)

Мультипроцессоры имеют преимущество перед другими видами параллельных компьютеров, поскольку с единой разделенной памятью очень легко работать. Например, представим, что программа ищет раковые клетки на сделанном через микроскоп снимке ткани. Фотография в цифровом виде может храниться в общей памяти, при этом каждый процессор обследует какую-нибудь определенную область фотографии. Поскольку каждый процессор имеет доступ к общей памяти, обследование клетки, которая начинается в одной области и продолжается в другой, не представляет трудностей.

2.7.3 Мультикомпьютеры

Мультипроцессоры с небольшим числом процессоров (< 64) сконструировать довольно легко, а вот создание больших мультипроцессоров представляет некоторые трудности. Сложность заключается в том, чтобы связать все процессоры с памятью. Чтобы избежать таких проблем, многие разработчики просто отказались от идеи разделенной памяти и стали создавать системы, состоящие из большого числа взаимосвязанных компьютеров, у каждого из которых имеется своя собственная память, а общей памяти нет. Такие системы называются мультикомпьютерами.

Процессоры мультикомпьютера отправляют друг другу послания (это несколько похоже на электронную почту, но гораздо быстрее). Каждый компьютер не обязательно связывать со всеми другими, поэтому обычно в качестве топологий используются 2D, 3D, деревья и кольца. Чтобы послания могли дойти до места назначения, они должны проходить через один или несколько промежуточных компьютеров. Тем не менее время передачи занимает всего несколько микросекунд. Сейчас создаются и запускаются в работу мультикомпьютеры, содержащие около 10 000 процессоров.

Поскольку мультипроцессоры легче программировать, а мультикомпьютеры — конструировать, появилась идея создания гибридных систем, которые сочетают в себе преимущества обоих видов машин. Такие компьютеры представляют иллюзию разделенной памяти, при этом в действительности она не конструируется и не требует особых денежных затрат.

ВЫВОДЫ

В данной научной работе был произведён анализ существующих подходов к классификации архитектур вычислительных систем, рассмотрены такие эффективные методы повышения производительности вычислительных систем, как параллельные вычисления и мультитрединг.

Работа рассчитана на продолжение исследований в этом направлении, целью которых является создание программного обеспечения формирования фазы определения для заданной системы команд.

Сама разработка вышеупомянутого программного обеспечения будет осуществляться в последующем при написании дипломного проекта, где и будут использованы результаты научно-исследовательской работы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Таненбаум Э. Архитектура компьютера, 4-е изд. – Спб.: Питер, 2003. – 700 с.;

2. М. Кузьминский, «Открытые системы», 1999, № 5-6, стр. 8;

3. М. Кузьминский, «Отрытые системы», 1999, № 9-10, стр. 8;

4. Головкин Б.А. Параллельные вычислительные системы. М.: Наука. 1980. 520 с.;

5. Методические указания по оформлению студенческих работ для студентов специальностей 7.080403 "Программное обеспечение автоматизированных систем" и 7.080404 "Интеллектуальные системы принятия решений" / Утв. Л.А. Белозерский и др. – Донецк: ДонГИИИ, 2001. – 52 с.